

TIBCO EBX®
Product Documentation

Version 6.2.3
April 2026

TIBCO®

Table of contents

User Guide

Introduction

1. How TIBCO EBX® works....17
2. Using the user interface....21
3. New user interface preview....27
4. Glossary....33

Data models

5. Introduction to data models....44

Implementing data models

6. Creating a data model....49
7. Configuring the data model....51
8. Implementing the data model structure....55
9. Properties of data model elements....61
10. Data validation controls on elements....75
11. Data model extensions....83
12. Working with an existing data model....95

Publishing and versioning data models

13. Publishing a data model....99
14. Versioning an embedded data model....101

Dataspaces

15. Introduction to dataspaces....104
16. Creating a dataspace....107
17. Working with existing dataspaces....109
18. Snapshots....117

Datasets

19. Introduction to datasets....122
20. Creating a dataset....125
21. Viewing data....127
22. Editing data....149

File import and export services

23. XML import and export....153
24. CSV import and export....159
25. Working with existing datasets....165
26. Dataset inheritance....169

Collaborative features (Team Up)

27. Introduction to collaborative features (Team Up)....174
28. Comments....177
29. Ratings....183
30. Tags....187

Workflow models

31. Introduction to workflow models 194
32. Creating and implementing a workflow model 199
33. Configuring the workflow model 213
34. Publishing workflow models 221

Data workflows

35. Introduction to data workflows 224
36. Using the Data Workflows area user interface 225
37. Work items 231

Managing data workflows

38. Launching and monitoring data workflows 237
39. Administration of data workflows 239

Data services

40. Introduction to data services 244
41. Generating data service WSDLs 247

EBX® Metadata Management

42. Introduction to metadata management 252
43. Key terms 256
44. Accessing the metadata perspective 259

User Interface

45. Overview of the User Interface 261

Navigation pane

46. Navigation pane overview 265

Metadata catalog

47. Catalog overview 269
48. Creating applications 271
49. Creating data elements 275
50. Provisioning 277
51. Regular expressions 287

Governance

52. Governance overview 293
53. Defining a business term 295

Metadata

54. Metadata section overview 297
55. Managing assets 299

Technology

56. Technology overview 305
57. Create an instance 307
58. Create a system 309
59. Create a system type 313
60. Create an infrastructure 317
61. Create a provider 319

Privacy

62. Privacy overview 321

63. Processing activities....323
64. Consents....325
65. Data breaches....327
66. Requests....329
67. Workspace....331
68. Home page....335
69. Custom views....357

Advanced data services

70. Advanced data services overview....363
71. Metadata harvesting....365
72. Asset sampling....379
73. Asset classification....383
74. Lineage....389

Collaboration

75. Collaboration overview....401
76. Metadata comments....403
77. Metadata ratings....407
78. Tag metadata....409
79. Managing metadata with collaborative workflows....413

Reference Manual

Integration

80. Overview of integration and extension....417
81. Using TIBCO EBX® as a Web Component....419
82. Built-in user services....427
83. Supported XPath syntax....441

Localization

84. Labeling and localization....450
85. Extending TIBCO EBX® internationalization....453

Persistence

86. Overview of persistence....458
87. History....461
88. Replication....469
89. Data model evolutions....475

Other

90. Inheritance and value resolution....480
91. Permissions....487
92. Criteria editor....509
93. Search....511

Administration Guide

94. Administration overview....518

Installation & configuration

95. Supported environments....524
96. Disk requirements....531
97. Jakarta EE deployment....533

Installation notes for EBX®

98. Installation notes for JBoss EAP 8.0.X....545
99. Installation notes for WebSphere Application Server Liberty 23.X....551
100. Installation notes for Tomcat 10.1.X....557

Installation notes for the EBX® Metadata Management application

101. Installation notes for EBX® Container Edition....563
102. Installation notes for on-prem Tomcat 10.1.X....565
103. TIBCO EBX® main configuration file....571
104. Initialization and first-launch assistant....603
105. Managing TIBCO EBX® add-ons....605
106. User authentication....609
107. Performance and tuning....615
108. Configuration notes....625

EBX® Container Edition

109. Building the image....628
110. Running the image....633
111. Customizing the image....653

Technical administration

112. Repository administration....660
113. UI administration....671
114. UI – Workflow launcher....689
115. Users and roles directory....703
116. Data model administration....709
117. Database mapping administration....711
118. External services configuration....715
119. Workflow management....719
120. Task scheduler....723
121. Background tasks....729
122. Audit trail....733
123. Collaborative features (Team Up)....737
124. AI Assistant....741
125. Other....747

Distributed Data Delivery (D3)

126. Introduction to D3....750
127. D3 broadcasts and delivery dataspaces....755
128. D3 JMS Configuration....759
129. D3 administration....767

130. Staging....777

131. New user interface preview 791

EBX® Metadata Management application

132. Metadata Administration Perspective 796
133. User roles 799
134. Permissions 801
135. User interface configuration 803
136. Workflows 805
137. Web Services and API 811

Metadata Agent

138. Installing the TIBCO® Metadata Agent 815
139. TIBCO TDV Metadata Adapter installation 825
140. TIBCO® EBX® Metadata Adapter installation 831
141. TIBCO® Metadata Agent Framework 835
142. Administration of TIBCO® Metadata Agent 841
143. Technical Users 843
144. TIBCO® Metadata Agents 845
145. Items managed by Metadata Agent 851
146. Harvesting configurations 859

Security Guide

147. Security Best Practices.....870

Developer Guide

Introduction

148. Packaging TIBCO EBX® modules.... 879
149. Mapping to Java.... 885
150. Tools for Java developers.... 891
151. Terminology changes.... 893

Data model

152. Introduction.... 896
153. Data types.... 899
154. Tables and relationships.... 913
155. Constraints.... 935
156. Triggers and functions.... 951
157. Labels and messages.... 955
158. Additional properties.... 961
159. Data services.... 969
160. Toolbars.... 971
161. Custom forms.... 973
162. Publication REST API.... 1011
163. Workflow model.... 1013

User interface

164. Interface customization.... 1024

User services

165. Overview.... 1027
166. Quick start.... 1031
167. Implementing a user service.... 1035
168. Declaring a user service.... 1049
169. Development recommendations.... 1055

SOAP data services

170. Introduction.... 1060
171. WSDL generation.... 1071
172. SOAP operations.... 1079

REST data services

173. Introduction.... 1114
174. Built-in RESTful services.... 1125

JSON Formats

175. Introduction.... 1191
176. Extended.... 1193
177. Compact.... 1223
178. Common.... 1233
179. Others.... 1243

SQL in EBX®

180. Introduction.... 1254
181. Comparison operators.... 1264

182. Arithmetic operators and functions....1268
183. Logical operators....1272
184. String operators and functions....1276
185. Date and time functions....1280
186. EBX® SQL functions....1283
187. REST Toolkit....1285
188. Record permission....1301

EBX® Script

189. Reference....1318

Usage

190. Function field....1343
191. Script task....1345
192. Table trigger....1351
193. Dataset trigger....1355
194. Scripted REST service....1359
195. REST service....1365
196. Role mapper....1373
197. Constraint validation....1377

API

198. Unit summary....1386
199. Unit default....1389
200. Unit core.complex....1393
201. Unit core.data....1398
202. Unit core.date....1420
203. Unit core.datetime....1426
204. Unit core.dictionary....1434
205. Unit core.http....1442
206. Unit core.json....1455
207. Unit core.list....1470
208. Unit core.locale....1477
209. Unit core.log....1479
210. Unit core.math....1482
211. Unit core.message....1496
212. Unit core.profile....1503
213. Unit core.resource....1505
214. Unit core.set....1510
215. Unit core.string....1518
216. Unit core.time....1533
217. Unit core.uri....1537
218. Unit core.validation....1539
219. Unit core.rest....1545

EBX® Script IDE

220. Overview of the EBX® Script IDE....1548
221. How to guide....1549

Tutorials

222. Creating functions....1553
223. Creating triggers....1559

224. Creating a workflow script..... 1563
225. Creating scripted REST services tutorial..... 1569
226. User interface reference..... 1579

Migration Guide

227. Migration introduction..... 1586
228. Environment and hardware requirements..... 1589
229. Changes to supported add-ons..... 1593
230. Step 1: Adapting custom code..... 1599
231. Step 2: Preparing the repository..... 1603
232. Step 3: Deploying..... 1611
233. Java API changes..... 1613

Notices

234. Documentation and Support....1616
235. Legal and Third-Party....1619

User Guide

Introduction

CHAPTER 1

How TIBCO EBX® works

This chapter contains the following topics:

1. Product overview
2. EBX® architecture

1.1 Product overview

Master Data Management (MDM) is a way to model, manage and ultimately govern shared data. When data needs to be shared by various IT systems, as well as different business teams, having a single governed version of master data is crucial.

With EBX®, business and IT users can collaborate on a single, unified solution in order to design data models and manage master data content.

EBX® is an MDM software that allows modeling any type of master data and implementing governance using the rich features included, such as collaborative workflows, data authoring, hierarchy management, version control, and role-based security.

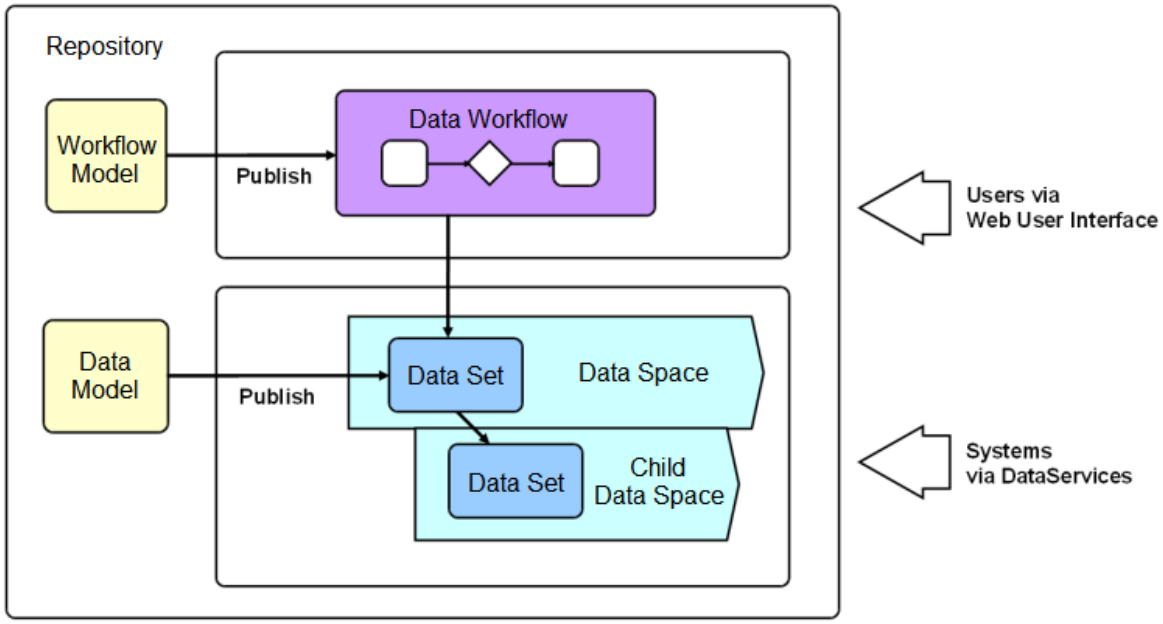
An MDM project using EBX® starts with the creation of a *data model*. This is where tables, fields, links and business rules related to the master data are defined. Examples of modeled data include product catalogs, financial hierarchies, lists of suppliers or simple reference tables.

The data model can then be published to make it available to *datasets*, which store the actual master data based on the structure defined in the data model. Datasets are organized and contained within *dataspaces*, containers that isolate updates from one another. Dataspaces allow working on parallel versions of data without the modifications impacting other versions.

Workflows are an invaluable feature for performing controlled change management or data approval. They provide the ability to model a step-by-step process involving multiple users, both human and automated.

Workflow models detail the tasks to be performed, as well as the parties associated with the tasks. Once a workflow model is published, it can be executed as *data workflows*. Data workflows can notify users of relevant events and outstanding work in a collaborative context.

Data services help integrate EBX® with third-party systems (middleware), by allowing external systems to access data in the repository, or to manage dataspaces and workflows through web services.



See also

Data modeling [p 34]

Datasets [p 36]

Dataspaces [p 38]

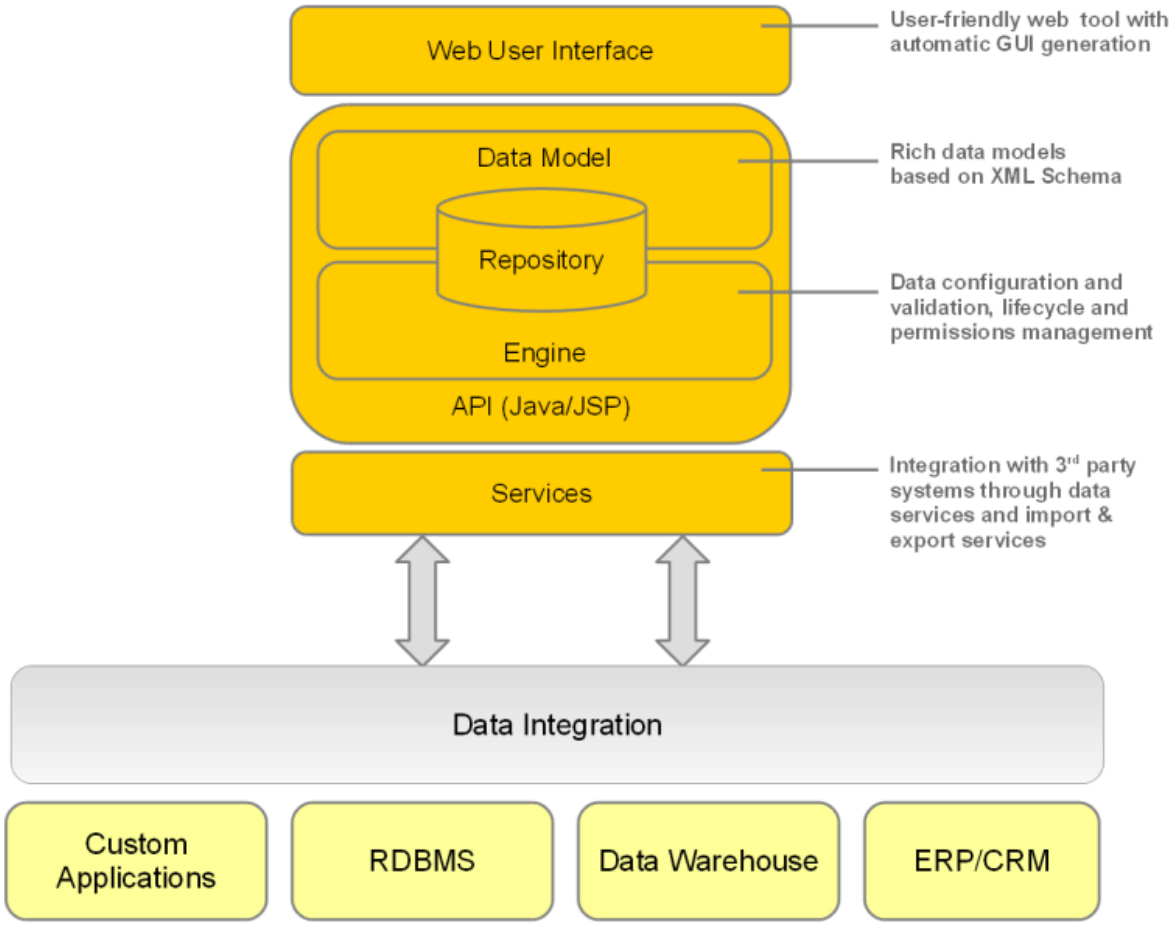
Workflow modeling [p 39]

Data workflows [p 40]

Data services [p 41]

1.2 **EBX® architecture**

The following diagram illustrates the EBX® architecture.



CHAPTER 2

Using the user interface

This chapter contains the following topics:

1. Overview
2. Advanced perspective
3. Perspectives
4. User pane
5. User interface features
6. Where to find EBX® help

2.1 Overview

The general layout of TIBCO EBX® workspaces is entirely customizable by a perspective administrator.

If several customized perspectives have been created, the tiles icon 'Select perspective' allows the user to switch between available perspectives.

The advanced perspective is accessible by default.

See also *UI administration* *[p 671]*

2.2 Advanced perspective

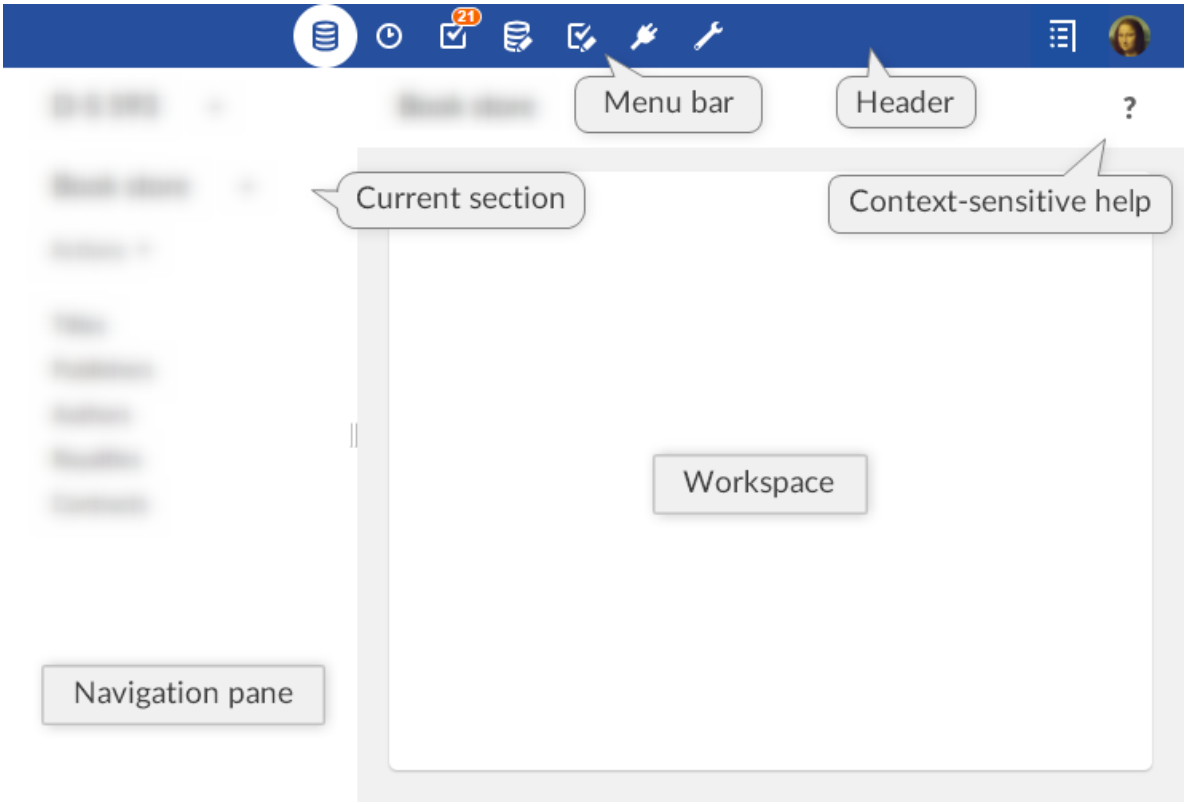
By default, the EBX® advanced perspective is available to all users, but its access can be restricted to selected profiles. The view is separated into several general areas, referred to as the following in the documentation:

> **Note**
>
> The advanced perspective is still accessible to users through explicit selection (for example through a Web component). Unlike other perspectives, it can only be "hidden" in the user interface so that users cannot apply it themselves.

- **Header:** Displays the avatar of the user currently logged in and the perspective selector. Clicking on the user's avatar gives access to the user pane.
- **Menu bar:** The functional categories accessible to the current user.

- **Navigation pane:** Displays context-dependent navigation options. For example: selecting a table in a dataset, or a work item in a workflow.
- **Workspace:** Main context-dependent work area of the interface. For example, the table selected in the navigation pane is displayed in the workspace, or the current work item is executed in the workspace.

The following functional areas are displayed according to the permissions of the current user: *Data*, *Dataspaces*, *Modeling*, *Data Workflow*, *Data Services*, and *Administration*.



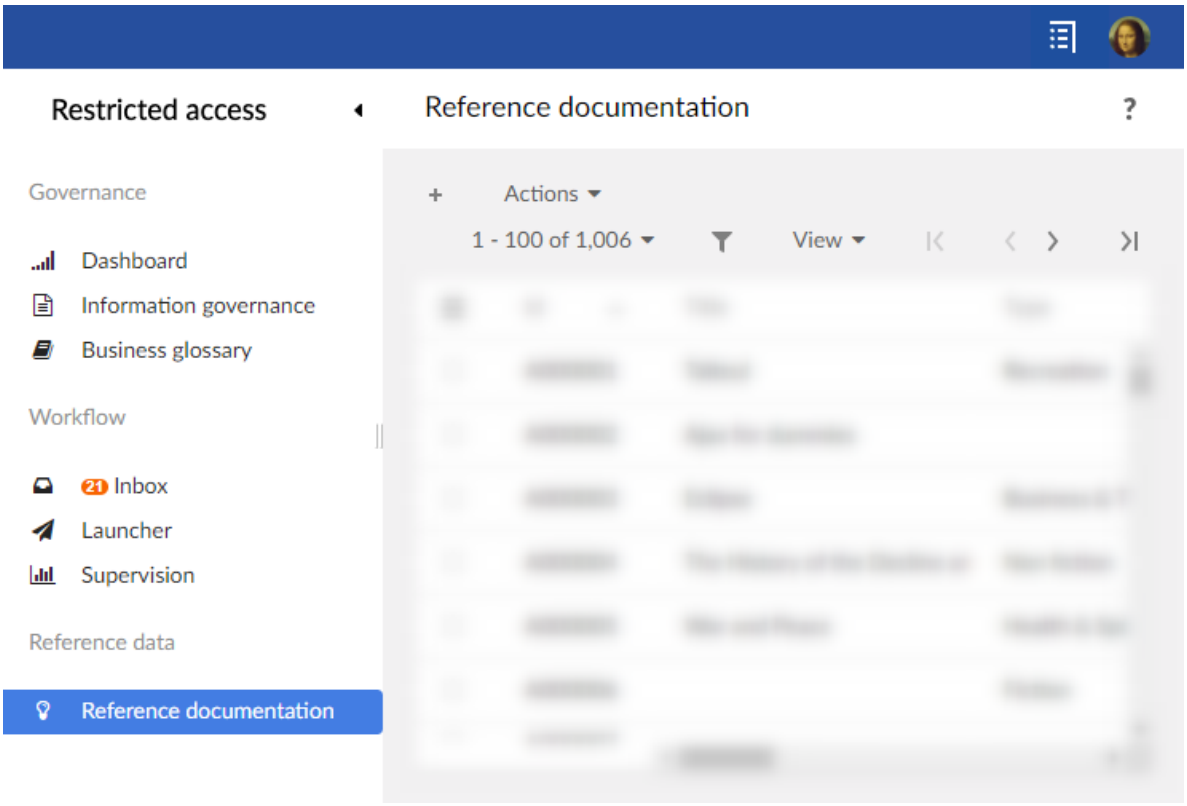
2.3 Perspectives

The EBX® perspectives are highly configurable views with a target audience. Perspectives offer a simplified user interface to business users and can be assigned to one or more profiles. This view is split into several general areas, referred to as the following in the documentation:

- **Header:** Displays the avatar of the user currently logged in and the perspective selector (when more than one perspective is available). Clicking on the user's avatar gives access to the user pane.
- **Navigation pane:** Displays the hierarchical menu as configured by the perspective administrator. It can be expanded or collapsed to access relevant entities and services related to the user's activity.
- **Workspace:** Main context-dependent work area of the interface.

Perspectives are configured by authorized users. For more information on how to configure a perspective, see perspective administration [p 672].

Example of a hierarchical menu:



Perspective modes

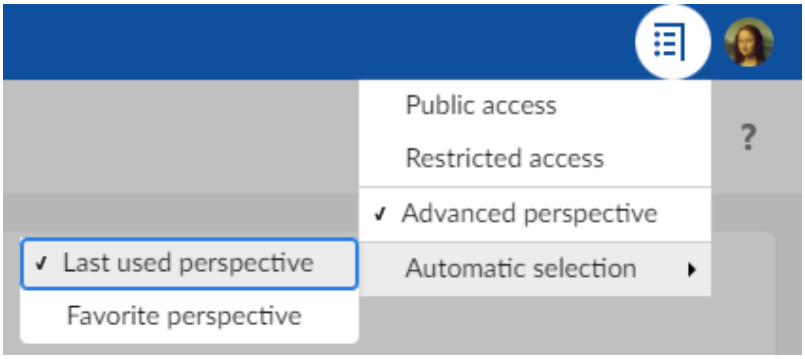
It’s possible now to choose between two modes of perspective.

According to the mode chosen by the user, the last visited perspective or the favorite one will be applied at connection.

Last used perspective:

This option allows the user to get the last selected perspective and it will be applied in the next log in.

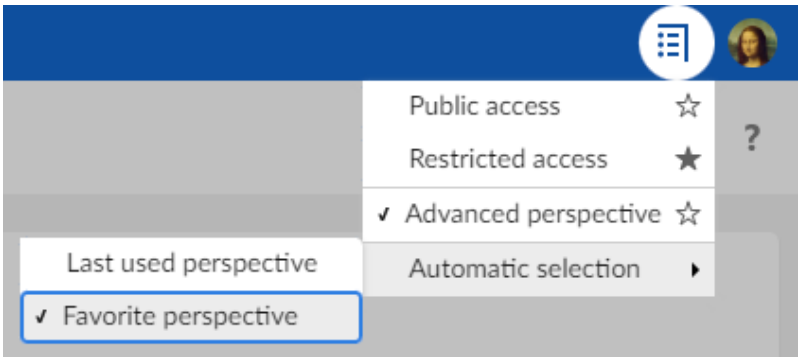
If the user chooses this mode, the stars next to the perspective name will disappear.



Favorite perspectives:

When more than one perspective is available to a user, it is possible to define one as their favorite perspective so that, when logging in, this perspective will be applied by default. To do so, an icon is available in the perspective selector next to each perspective:

- A full star indicates the favorite perspective. A click on it will remove the favorite perspective.
- An empty star indicates that the associated perspective is not the favorite one. A click on it will set this perspective as the favorite one.



See also *Recommended perspectives* *[p 684]*

2.4 User pane

General EBX® features are grouped in the user pane. It can be accessed by clicking on the avatar (or user's initials) in the upper right corner of any page.

The user pane is then displayed with the user avatar and gives access to the profile configuration (according to the user's rights), language selection, density selection and online documentation.

> **Attention**
> The logout button is located on the user pane.

Avatar

An avatar can be defined for each user. The avatar consists in a picture, defined using a URL path; or in two letters (the user's initials by default). The background color is set automatically and cannot be modified. Regarding the image that will be used, it has to be a square format but there is no size limitation.

> **Note**
>
> Avatars appear in the user pane, history and workflow interfaces.

The feature is also available through the Java method `UIComponentWriter.addUserAvatar`[API].

The avatar layout can be customized in the 'Ergonomics and layout' section of the 'Administration' area. It is possible to choose between the display of the avatar only, user name only, or to display both.

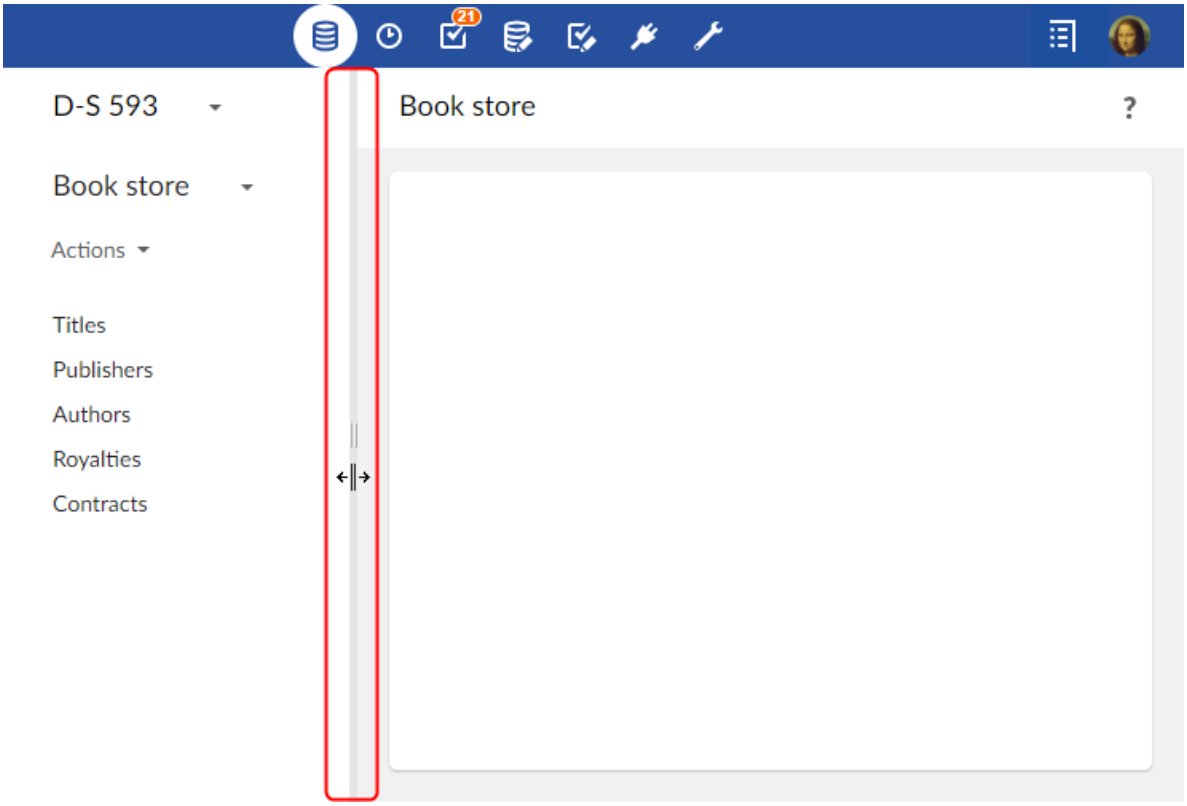
Density

Users can choose their display density mode between 'Compact' and 'Comfortable'. The display mode can be modified from the user pane.

2.5 User interface features

Resetting the navigation pane width

After having resized the width of the navigation pane, you can restore it to the default width by hovering over the border and double-clicking.



2.6 Where to find EBX® help

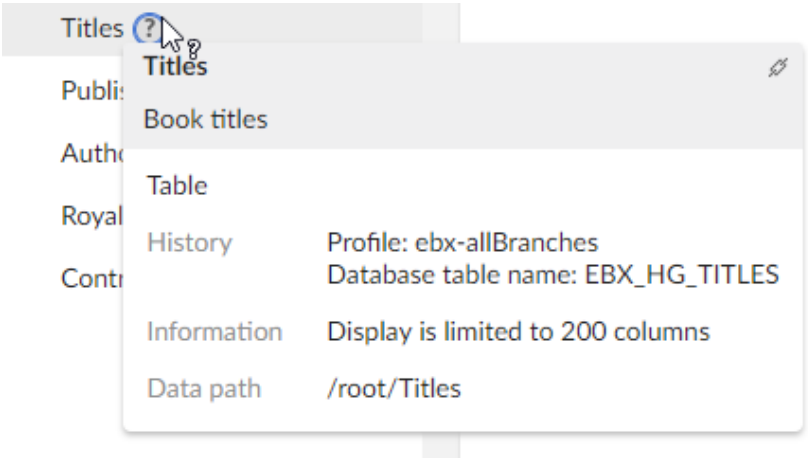
In addition to the full standalone product documentation accessible via the user pane [p 24], help is accessible in various forms within the interface.

Context-sensitive help

When browsing any workspace in EBX®, context-specific help is available by clicking on the question mark located to the right side of the second header. The corresponding chapter from the product documentation will be displayed.

Contextual help on elements

When you hover over an element for which contextual help has been defined, a question mark appears. Clicking on the question mark opens a panel with information on the element.



When a permalink to the element is available, a link button appears in the upper right corner of the panel.



CHAPTER 3

New user interface preview

This chapter contains the following topics:

1. Overview
2. About the preview
3. Accessing the new UI
4. Navigation basics
5. Using the new UI
6. Glossary

3.1 Overview

This chapter helps you understand and use the new user interface (UI) that is available as a preview. Use this guide to learn how to navigate the UI, complete common tasks, and understand key terms. This content is written for business users (data consumers, data stewards, and contributors). Administrative configuration is documented in the *Administration Guide > New user interface preview*.

The new UI is designed to support faster navigation, better context preservation, and more flexible ways of working with data, especially when moving between tables, records, and in-progress work.

> **Note**
>
> Use of the new UI is optional. It can be evaluated alongside the existing UI, which remains the recommended interface for production use.

3.2 About the preview

The new UI is provided as a preview to gather feedback and allow early adoption. Preview features may change before general availability.

Preview features can change between releases. Some features from the current UI may be missing or behave differently. Performance and accessibility are still being tuned.

Use the preview:

- To become familiar with upcoming navigation and interaction patterns.
- To perform day-to-day tasks if your administrator has enabled access.

Do not to use the preview:

- For critical operations that require stable, production-ready behavior.
- If your role depends on features not yet available in the preview.

> **Note**
>
> Your administrator controls availability of the preview UI.

Submitting feedback

Feedback on any aspect of EBX® is always welcome. This is especially true for the UI preview. Use the special *UI > UI Preview* category in the EBX® Ideas Portal to submit any feedback.

3.3 Accessing the new UI

Your administrator controls the availability of the preview UI. After they have deployed and enabled the UI, you can access it:

- From the **Perspectives** menu, using the **Homepage** option. Note that if the new UI's artifacts are not deployed, the **Homepage** option takes you to the default homepage.
- By appending the following to your EBX® URL: `/ebx-hub`.

3.4 Navigation basics

The new UI is composed of three primary components: the Dock, Workspace, and Cards. These components give you access to navigation and data operations.

The following image is a simplified illustration of the new UI and is intended to highlight the aforementioned components along with their descriptions in the list below:



1) Dock	The resizeable Dock is the persistent vertical area on the left side of the screen. Its main purpose is to enable navigation between **Home**, available perspectives, in-progress data operations, and open tables and records.
2) Workspace	The Workspace is the primary area displayed to the right of the Dock. When accessing the new UI, the Workspace defaults to **Home**. When you navigate using the Dock, the Workspace focus automatically updates to the selected view or component.
3) Cards	Cards are the individual units of content shown in the Workspace. They present information, actions, or shortcuts related to your work and provide a consistent way to interact with data and functionality throughout the new UI. As you use cards to navigate deeper, new columns of cards open to the right, showing more specific or detailed views related to your selection. This allows you to progressively explore data while keeping the previous context visible. The content of some of the cards that display on **Home** is configurable. After navigating away from **Home**, the

content shown in cards is determined by the perspective configuration.

3.5 Using the new UI

The following topics in this section introduce some of the ways in which you can interact with the new UI:

- Home configuration [p 30]: describes how to personalize your **Home**.
- Understanding view tuning [p 31]: highlights the consolidation of data view access, filtering, and search capabilities.
- Becoming familiar with drafts [p 31]: outlines the new concept of drafts when creating or editing data.

Configuring Home

Home is your personalized landing area in the new UI. You can configure it to surface the data, recent items, and components that matter to you the most.

When you edit **Home**, you can:

- Add or remove cards that provide the following features:
  - Shortcuts to frequently visited tables
  - Display open drafts
  - View record ratings
  - Show your workflow inbox
  - View your recently created and modified records
  - Count the number of records in specified tables
- Rearrange, resize, and configure cards.
- Change the layout of the Dock. This layout persists when navigating away from **Home**.

To configure **Home**:

1. If this is your first time accessing the new UI, click **Edit Home**. Alternatively click the **Edit** icon in the upper-right of the screen.
2. Once in edit mode, expand either **Dock** or **Home**, depending on the changes you want to make.
3. Edit the Dock by reordering display components.
4. Edit **Home** display by:
   - Dragging the desired cards to the grid.
   - Once cards are on the grid, you can resize and rearrange them. The information displayed on some cards is set by default, and others require configuration. See the next step for cards that require configuration.
5. Click a card's **Edit** icon to provide information required by the card.
6. After making changes, save and close.

Understanding view tuning

In the new UI, you can tune the view of a table by clicking the gear icon at the top of the card. View tuning aims to help you locate data by providing the following capabilities:

- Select and display any configured table views, including hierarchical views.
- Perform a search on the table data.
- Sort the records using one or more fields.

Becoming familiar with drafts

The new UI introduces the concept of drafts to create and edit data before committing to the database.

How drafts work:

- When you add a new record or edit an existing record, a new draft is created.
- Changes are saved to the draft as you make them.
- Drafts allow you to pause work, navigate elsewhere, or return later without committing partial changes.

Note that draft records do not display on a table's card since the changes have not yet been propagated to the database. To access in progress drafts, you can:

- Go to **Home** and use the **Drafts** card.
- Select an open draft when creating or editing a record.

3.6 **Glossary**

This section provides definitions for terms used in this document.

Dock	The persistent vertical bar on the left side of the screen that provides navigation to **Home**, configured perspectives, current data operations, open tables and records.
Workspace	The primary area displayed to the right of the dock. This is where users view, interact with, and manage data to complete their tasks.
Card	A card is a structured container in the workspace that presents information, actions, or tools. Cards provide a consistent way to view and interact with data across the product. Depending on permissions, cards may be editable, configurable, or read-only.
Draft	A draft represents a set of uncommitted changes that you can work on incrementally before saving them to the database. Drafts are created automatically when you add or edit data.
Home	Your personalized landing area in the new UI. Home displays cards such as shortcuts, drafts, and recent activity, and can be customized to highlight the information most relevant to your work.
Drafts card	A **Home** card that shows your in-progress drafts and allows you to return to unfinished data edits.
View tuning	Tools that allow you to adjust how data is displayed in a table, including selecting views, searching, and sorting records.

CHAPTER 4

Glossary

This chapter contains the following topics:

1. Governance
2. Data modeling
3. Datasets
4. Data management life cycle
5. History
6. Workflow modeling
7. Data workflows
8. Data services
9. Cross-domain

4.1 Governance

repository

A back-end storage entity containing all the data managed by TIBCO EBX®. The repository is organized into dataspaces.

See also dataspace [p 38].

profile

The generic term for a user or a role. Profiles are used in data workflows and for defining permission rules.

See also user [p 33], role [p 34].

Related Java API `Profile`[API].

user

An entity created in the repository in order for physical users or external systems to authenticate and access EBX®. Users may be assigned roles and have other account information associated with them.

See also user and roles directory [p 34], profile [p 33].

Related concept User and roles directory [p 703].

Related Java API `UserReference`[API].

role

A user classification, used for permission rules and data workflows, which can be assigned to users. Each user may belong to multiple roles.

Whenever a role profile is specified in EBX®, the behavior resulting from that designation is applied to all users that are members of that role. For example, in a workflow model, a role may be specified when defining to whom work items are offered. As a result, all users belonging to that role can receive the same work item offer.

See also user and roles directory [p 34], profile [p 33].

Related concept User and roles directory [p 703].

Related Java API `Role`[API].

administrator

A predefined role that has access to the technical administration and configuration of EBX®.

user and roles directory

A directory defining the methods available for authentication when accessing the repository, all available roles, and the users authorized to access the repository with their role assignments.

See also user [p 33], role [p 34].

Related concept User and roles directory [p 703].

Related Java API `Directory`[API], `DirectoryHandler`[API].

user session

A repository access context that is associated with a user after being authenticated against the user and roles directory.

Related concept User and roles directory [p 703].

Related Java API `Session`[API].

4.2 Data modeling

Main documentation section Data models [p 44]

data model

A structural definition of the data to be managed in the EBX® repository. A data model includes detailed descriptions of all included data, in terms of organization, data types, and semantic relationships. The purpose of data models is to define the structure and characteristics of datasets, which are instances of data models that contain the data being managed by the repository.

See also dataset [p 36].

Related concept Data models [p 44].

field

A data model element that is defined with a name and a simple datatype. A field can be included in the data model directly or as a column of a table. In EBX®, fields can be assigned basic constraints, such as length and size, as well as more complex validation rules involving computations. Automated value assignment using field inheritance or computations based on other data can also be defined for fields. Aggregated lists can be created by setting the cardinality of a field to allow multiple values in the same record. Fields can be arranged into groups to facilitate structural organization in the data model.

By default, fields are denoted by the icon  .

See also record [p 36], group [p 35], table (in data model) [p 35], validation rule [p 36], inheritance [p 37].

Related concepts Structure elements properties [p 61], Controls on data fields [p 75].

Related Java API `SchemaNode`[API].

The former name (prior to version 5) of "field" was "attribute".

primary key

A field or a composition of multiple fields used to uniquely identify the records in a table.

Primary keys are denoted by the icon  .

Related concept Tables definition [p 913].

foreign key

A field or a composition of multiple fields in one table whose field values correspond to the primary keys of another table. Foreign keys are used to reference records in one table from another table.

Foreign keys are denoted by the icon  .

See also primary key [p 35].

Related concept Foreign key [p 919].

table (in data model)

A data model element that is composed of fields and/or groups of fields. Every table must define at least one field to act as the unique identifier, or primary key, of records. A table in a data model can be used to create a reusable type based on the table's structure, which can then be used to create other elements of the same structure in the data model.

Tables are represented by the icon  .

See also record [p 36], primary key [p 35], reusable type [p 36].

group

A classification entity used to facilitate the organization of a data model. A group can be used to collect fields, other groups, and tables. If a group contains tables, the group cannot be included within another table, as the constraint that tables cannot be nested must be respected. A group can be used to create a reusable type based on the group's structure, which can then be used to create other elements of the same structure in the data model.

Groups are represented by the icon  .

See also reusable type [p 36].

Related Java API `SchemaNode`[API].

reusable type

A shared simple or complex type definition that can be used to define other elements in the data model.

validation rule

An acceptance criterion defined on a field or a table. Data is considered invalid if it does not comply with all imposed validation rules.

The former name (prior to version 5) of "validation rule" was "constraint".

data model assistant (DMA)

The EBX® user interface includes a tool that aids the implementation of data models. It allows defining the structure of data models, creating and editing elements, as well as configuring and publishing data models.

See also Data models [p 44].

4.3 Datasets

Main documentation section Datasets [p 122]

record

A set of field values in a table, uniquely identified by a primary key. A record is a row in the table. Each record follows the data structure defined in the data model. The data model drives the data types and cardinality of the fields found in records.

See also table (in dataset) [p 36], primary key [p 35].

The former name (prior to version 5) of "record" was "occurrence".

table (in dataset)

A set of records (rows) of the same structure containing data. Each record is uniquely identified by its primary key.

Tables are represented by the icon  .

See also record [p 36], primary key [p 35].

dataset

A data-containing instance of a data model. The structure and behavior of a dataset are based upon the definitions provided by the data model that it is implementing. Depending on its data model, a dataset contains data in the form of tables, groups, and fields.

Datasets are represented by the icon  .

See also table (in dataset) [p 36], field [p 35], group [p 35], views [p 37].

Related concept Datasets [p 122].

The former name (prior to version 5) of "dataset" was "adaptation instance".

inheritance

A mechanism by which data can be acquired by default by one entity from another entity. In EBX®, there are two types of inheritance: dataset inheritance and field inheritance.

When enabled, dataset inheritance allows a child dataset to acquire default data values from its parent dataset. This feature can be useful when designing a data model where data declared in a parent scope will be used with the same value by default in nested child scopes. Values that are inherited from the parent can be overridden by the child. By default, dataset inheritance is disabled. It can be enabled during the data model definition.

Inheritance from the parent dataset is represented by the icon  .

Field inheritance is defined in the data model to automatically fetch a field value from a record in another table.

Inherited fields are represented by the icon  .

Related concept Inheritance and value resolution [p 480].

views

A customizable display configuration that may be applied to viewing tables. A view can be defined for a given user or role, in order to specify whether records are displayed in a tabular, hierarchical or tile format, as well as to set record filtering criteria.

The hierarchical view type offers a tree-based representation of the data in a table. Nodes in the tree can represent either field values or records. A hierarchical view can be useful for showing the relationships between the model data. When creating a view that uses the hierarchical format, dimensions can be selected to determine the structural representation of data. In a hierarchical view, it is possible to navigate through recursive relationships, as well as between multiple tables using foreign key relationships.

See also

Views [p 133]

Hierarchies [p 135]

recommended view

A recommended view can be defined by the dataset owner for each target profile. When a user logs in with no view specified, their recommended view (if any) is applied. Otherwise, the default view is applied.

The 'Manage recommended views' action allows defining assignment rules for recommended views depending on users and roles.

Related concept Recommended views [p 145].

favorite view

When displaying a table, the user can choose to define the current as their favorite view through the 'View' menu toolbar.

Once it has been set as the favorite, the view will be automatically applied each time this user accesses the table.

Related concept 'View' menu toolbar [p 146].

4.4 Data management life cycle

Main documentation section Dataspaces [p 104]

dataspace

A container entity composed of datasets. It is used to isolate different versions of datasets or to organize them.

Child dataspaces may be created based on a given parent dataspace, initialized with the state of the parent. Datasets can then be modified in the child dataspaces in isolation from their parent dataspace as well as each other. The child dataspaces can later be merged back into their parent dataspace or compared against other dataspaces.

See also inheritance [p 37], repository [p 33], dataspace merge [p 38].

Related concept Dataspaces [p 104].

The former name (prior to version 5) of "dataspace" was "branch" or "snapshot".

reference dataspace

The root ancestor dataspace of all dataspaces in the EBX® repository. As every dataspace merge must consist of a child merging into its parent, the reference dataspace is never eligible to be merged into another dataspace.

See also dataspace [p 38], dataspace merge [p 38], repository [p 33].

dataspace merge

The integration of the changes made in a child dataspace since its creation into its parent dataspace. The child dataspace is closed after the merge has completed successfully. To perform a merge, all the differences identified between the source dataspace and the target dataspace must be reviewed, and conflicts must be resolved. For example, if an element has been modified in both the parent and child dataspace since the creation of the child dataspace, the conflict must be resolved manually by deciding which version of the element should be kept as the result of the merge.

Related concept Merge [p 112].

snapshot

A static copy of a dataspace that captures its state and all of its content at a given point in time for reference purposes. A snapshot may be viewed, exported, and compared to other dataspaces, but it can never be modified directly.

Snapshots are represented by the icon  .

Related concept Snapshot [p 117]

The former name (prior to version 5) of "snapshot" was "version" or "home".

4.5 History

Main documentation section History [p 461]

historization

A mechanism that can be enabled at the table level to track modifications in the repository. Two history views are available when historization is activated: table history view and transaction history view. In all history views, most standard features for tables, such as export, comparison, and filtering, are available.

Activation of historization requires the configuration of a history profile. The historization of tables is not enabled by default.

See also table history view [p 39], transaction history view [p 39], history profile [p 39].

history profile

A set of preferences that specify which dataspaces should have their modifications recorded in the table history, and whether transactions should fail if historization is unavailable.

See also history profile [p 39].

table history view

A view containing a trace of all modifications that are made in a given table, including record creations, updates, and deletions. Each entry includes transactional information, such as a timestamp and the user performing the action, as well as the data at the conclusion of the transaction. This information can also be consulted at a record or dataset level.

Related technical reference History [p 461].

transaction history view

A view displaying the technical and authentication data of transactions, either globally at the repository level, or at the dataspace level. As a single transaction can perform multiple actions and affect multiple tables in one or more datasets, this view shows all the modifications that have occurred across the given scope for each transaction.

Related technical reference History [p 461].

4.6 **Workflow modeling**

Main documentation section Workflow models [p 194]

workflow model

A procedural definition of operations to be performed on data. A workflow model describes the complete path that the data must follow in order to be processed, including its states and associated actions to be taken by human users and automated scripts.

Related concept Workflow models [p 194].

The former name (prior to version 5) of "workflow model" was "workflow definition".

Workflow models are represented by the icon .

script task

A data workflow task performed by an automated process, with no human intervention. Common script tasks include dataspace creation, dataspace merges, and snapshot creation.

Script tasks are represented by the icon  .

See also workflow model [p 39].

user task

A data workflow task that is made up of one or more work items performed concurrently by human users. User task work items are offered or assigned to users, depending on the workflow model. The progression of a data workflow beyond a user task depends on the satisfaction of the task termination criteria defined in the workflow model.

User tasks are represented by the icon  .

See also workflow model [p 39].

workflow condition

A decision step in a data workflow. A data workflow condition describes the criteria used to decide which step will be executed next.

Workflow conditions are represented by the icon  .

sub-workflow invocation

A step in a data workflow that pauses the current data workflow and launches one or more other data workflows. If multiple sub-workflows are invoked by the same sub-workflow invocation step, they will be executed concurrently, in parallel.

wait task

A step in a data workflow that pauses the current workflow and waits for a specific event. When the event is received, the workflow is resumed and automatically goes to the next step.

data context

A set of data that may be shared between steps throughout a data workflow to ensure continuity between steps.

4.7 Data workflows

Main documentation section Data workflows [p 224]

workflow publication

An instance of a workflow model that has been made available for execution to users with the appropriate permissions.

The former name (prior to version 5) of "workflow publication" was "workflow".

data workflow

An executed instance of a workflow model, which runs the data processing steps that are defined in the model, including user tasks, script tasks, and conditions.

See also workflow model [p 39].

Related concept Data workflows [p 224].

The former name (prior to version 5) of "data workflow" was "workflow instance".

work list

A list of all published data workflows that the current user has the permissions to view. Users with the permissions to launch data workflows do so from their 'Work List'. All outstanding work items requiring action from the user appear under their published workflows in the work list. Additionally, if the user is the administrator of data workflows, they are able to view the state of execution of those data workflows in their 'Work List', and may intervene if necessary.

work item

An action that must be performed by a human user as a part of a user task.

Allocated work items are represented by the icon  .

See also user task [p 40].

token

Tokens are used during data workflow management, and are visible to repository administrators.

4.8 Data services

Main documentation section Data services [p 244]

data service

EBX® shares master data according to the Service-oriented architecture (SOA) by using XML web services. Since all data services are generated directly from models or built-in services they can be used to access part of the features available from the user interface.

Data services offer:

- a WSDL model-driven and built-in generator to build a communication interface. It can be produced through the user interface or the HTTP(S) connector for a client application. XML messages are communicated to the EBX® entry point.
- a SOAP connector or entry point component for SOAP messages which allows external systems interacting with the EBX® repository. This connector responds to requests coming from the WSDL produced by EBX®. This component accepts all SOAP XML messages corresponding to the EBX® WSDL generator.
- A RESTful connector, or entry point for the select operations, allows external systems interrogating the EBX® repository. After authenticating, it accepts the request defined in the URL and executes it according to the permissions of the authenticated user.

lineage

A mechanism by which access rights profiles are implemented for data services. Access rights profiles are then used to access data via WSDL interfaces.

Related concept: Generating a WSDL for lineage [p 249].

4.9 Cross-domain

node

A node is an element of a tree view or a graph. In EBX®, 'Node' can carry several meanings depending on the context of use:

- In the workflow model [p 39] context, a node is a workflow step or condition.
- In the data model [p 34] context, a node is a group, a table or a field.
- In the hierarchy [p 37] context, a node represents a value of a dimension.
- In an adaptation tree [p 37], a node is a dataset.
- In a dataset [p 36], a node is the node of the data model evaluated in the context of the dataset or the record.

Data models

CHAPTER 5

Introduction to data models

This chapter contains the following topics:

1. Overview
2. Using the Data Models area user interface

5.1 Overview

What is a data model?

The first step towards managing data in TIBCO EBX® is to develop a data model. The purpose of a data model is to provide the detailed structural definition of the data that will be managed in the repository, in terms of organization, data types, and semantic relationships.

In order to implement a data model in the repository, you will first create a new data model, then define the details of the structure of its component table, field, and group elements, as well as their behavior and properties. When you have completed the entry or import of your data model structure in the repository, you will publish it to make it available for use by datasets. Once you have a publication of your data model, you and other users can create datasets based upon it to contain the data that is managed by the EBX® repository.

Basic concepts used in data modeling

A basic understanding of the following terms is necessary to proceed with the creation of data models:

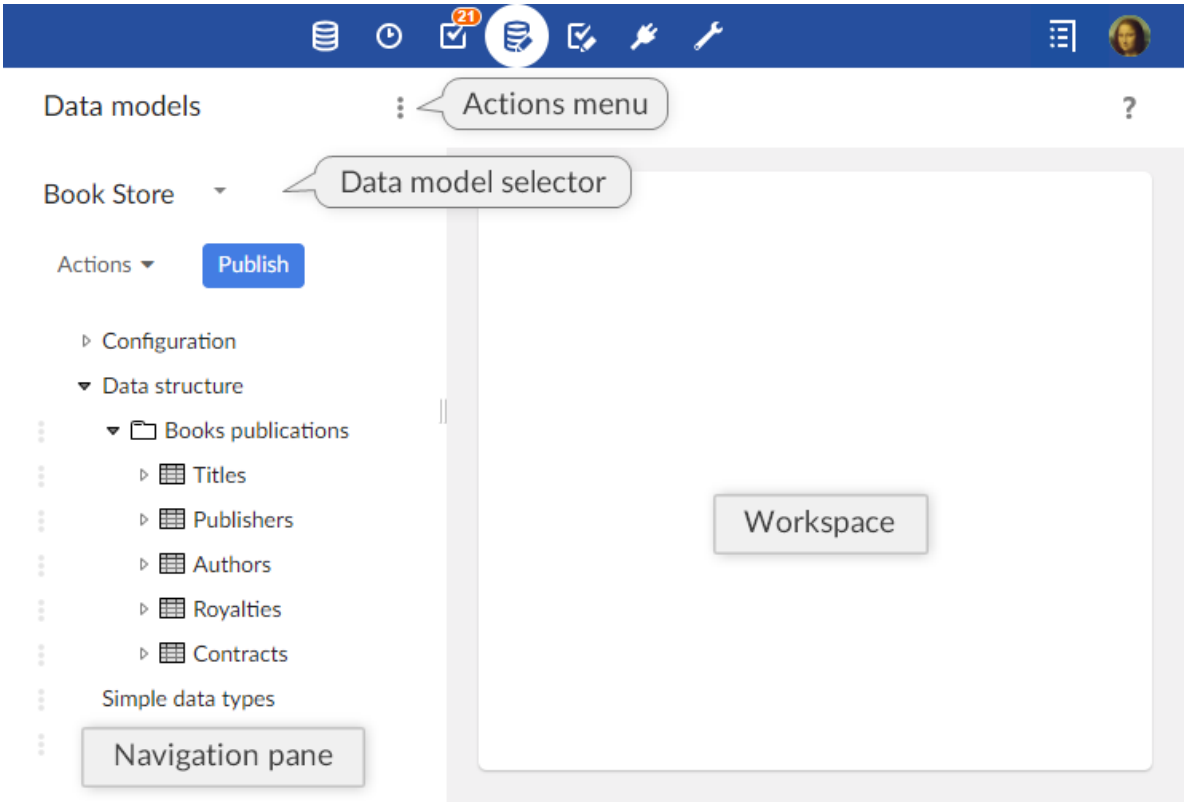
- field [p 35]
- primary key [p 35]
- foreign key [p 35]
- table (in data model) [p 35]
- group [p 35]
- reusable type [p 36]
- validation rule [p 36]

5.2 Using the Data Models area user interface

Navigating within the Data Model Assistant

Data models can be created, edited or imported, and published in the **Data Models** area of the user interface. The EBX® data model assistant (DMA) facilitates the development of data models.

> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective'.



The navigation pane is organized into the following sections:

Configuration	The technical configuration of the data model.
Global properties	Defines the global properties of the data model.
Included data models	Defines the data models included in the current model. All types defined in included data models can be reused in the current model.
Component library	Defines the Java components available in the model. These provide programmatic features that will be available for the model, such as programmatic constraints, functions, and UI beans.
Add-ons	Specifies which add-ons are used by the data model. These add-ons will have the capacity to enrich the current data model after the publication by adding properties and constraints to the elements of the data model.
Data structure	The structure of the data model. Defines the relationship between the elements of the data model and provides access to the definition of each element.
Simple data types	Simple reusable types defined in the current data model.
Complex data types	Complex reusable types defined in the current data model.
Included simple data types	Simple reusable types defined in an included external data model.
Included complex data types	Complex reusable types defined in an included external data model.
Extensions	Extensions available in the current data model.
Toolbars	The toolbars available to use in the data model.
User services	Declares the user services using the API available before release 5.8.0. From release 5.8.0, it is advised to use the new UserService API (these services are directly registered through the Java API, hence no declaration is required for them in the data model assistant)..

Entity mappings	Specifies entity names for the tables in the data model. These names are used as aliases in SQL requests or as suffixes in WSDL operations. This allows to refer to a table in SQL requests and data service operations using a unique name instead of its path.
Replications	This table defines the replication units of the data model. A replication unit allows the replication of a source table in the relational database, so that external systems can access this data by means of plain SQL requests and views.
Ajax components	Defines the available Ajax components in the model.
Java bindings	The bindings specify what Java types have to be generated from the model.

See also

Implementing the data model structure *[p 55]*

Configuring the data model *[p 51]*

Reusable types *[p 57]*

Data model extensions *[p 83]*

Data model element icons


field [p 35]


primary key [p 35]


foreign key [p 35]


table [p 35]


group [p 35]

Related concepts

Dataspaces *[p 104]*

Datasets *[p 122]*

CHAPTER 6

Creating a data model

This chapter contains the following topics:

1. Creating a new data model

6.1 Creating a new data model

To create a new data model, click the **Create** button in the pop-up, and follow through the wizard.

CHAPTER 7

Configuring the data model

This chapter contains the following topics:

1. Information associated with a data model
2. Permissions
3. Data model properties
4. Included data models
5. Add-ons used by the data model

7.1 Information associated with a data model

To view and edit the owner and documentation of your data model, select 'Information' from the **data model 'Actions'** [p 45] menu for your data model in the navigation pane.

> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective'.

Unique name	The unique name of the data model. This name cannot be modified once the data model has been created.
Owner	Specifies the data model owner, who will have permission to edit the data model's information and define its permissions.
Localized documentation	Localized labels and descriptions for the data model.

7.2 Permissions

To define the user permissions on your data model, select 'Permissions' from the **data model 'Actions'** [p 45] menu for your data model in the navigation pane.

The configuration of the permissions of a data model are identical to the options for the permissions of a dataset, as explained in Permissions [p 165].

7.3 Data model properties

In the navigation pane, under Configuration > Data model properties, you can access the following technical properties:

Module name	Defines the module that contains the resources that will be used by this data model. This is also the target module used by the data model publication if publishing to a module.
Module path	Physical location of the module on the server's file system.
Sources location	The source path used when configuring Java components in the 'Component library'. If this path is relative, it will be resolved using the 'Module path' as the base path.
Publication mode	Whether to publish the data model as an XML Schema Document within a module or as a publication completely embedded in the TIBCO EBX® repository. Embedded data models offer additional functionality such as versioning and rollback of publications. See Publication modes [p 99] for more information. **Model path in module**: Defines the target file for the data model generation. It must start with '/'.
Dataset inheritance	Specifies whether dataset inheritance is enabled for this data model. Dataset inheritance is disabled by default. See Dataset inheritance [p 169] for more information.
Documentation	Documentation of the data model defined by a Java class. This Java class can programmatically specify labels and descriptions for the elements of the data model. The labels and descriptions defined in this Java class are displayed in associated datasets in preference to the ones defined locally on an element. See Dynamic labels and descriptions [p 956] for more information.
Special extensions	Access permissions defined by programmatic rules in a Java class.
Disable auto-increment checks	Specifies whether to disable if the check of an auto-incremented field value in associated datasets regarding to the "max value" found in the table being updated. See Auto-incremented values [p 953] for more information.

Enable user services (old API)	Specifies if user services using the API available before release 5.8.0 can be declared. If 'No', the section 'Configuration > User services' is not displayed (except if at least service has been already declared in this section). From release 5.8.0, it is advised to use the new UserService Java API (these services are directly registered through the Java API, hence no declaration is required in the data model assistant). See `UserServiceDeclaration`[API] for more information.

7.4 Included data models

You can use data types in the current model that are defined in another data model by adding an entry for the other data model in the table under Configuration > Included data models.

When you access the record of an included model in this table, you will find technical information about the model under the **Information** tab. As an included data model could eventually have validation errors, for example, due to deleted Java resources, this view will provide information regarding those issues.

It is only possible to include data models that have no validation errors and have been defined and published as an embedded data model or packaged in a module.

The names of data types must be unique across both locally defined and included type definitions. That is, included data types must not have names that coincide with those of data types defined in the current data model or other included data models.

See also *Including external data models* *[p 911]*

7.5 Add-ons used by the data model

On any data model, it is possible to specify the *add-ons* used by the current data model. These add-ons will have the capacity to enrich the current data model after the publication by adding properties and constraints to the data model elements.

To define an add-on to be used by the data model through the user interface, create a new record in the 'Add-ons' table under the data model configuration in the navigation pane. A record of this table defines the following properties:

Name	Add-on public name.
Version	Add-on version.
Activated	Indicates if the add-on is activated. The add-on must be activated in order to be used.

CHAPTER 8

Implementing the data model structure

To work with the structural definition of your data model, select the data model you are working with in the navigation pane.

You can then access the structure of your data model in the navigation pane under 'Data structure', to define the structure of fields, groups, and tables.

This chapter contains the following topics:

1. Common actions and properties
2. Reusable types
3. Data model element creation details
4. Modifying existing elements

8.1 Common actions and properties

Adding elements to the data model

The following elements are available to describe the structure of your data model:

- fields
- groups
- tables
- primary keys
- foreign keys
- associations

Add a new element relative to any existing element in the data structure by clicking the down arrow to the right of the existing entry, and selecting an element creation option from the menu. Depending on whether the existing element is a field, group, or table, you have the choice of creating the new

element as a child of the existing element, or before or after the existing element at the same level. You can then follow the element creation wizard to create the new element.

> **Note**
>
> The element `root` is always added upon data model creation. If this element must be renamed, it can be deleted and recreated with a new name.

Names, labels, descriptions, and information

Whenever you create a new element in your data model, you must provide a name for the field 'Name' that is unique in its level of the data structure. This name is assigned once the element is created and cannot be changed subsequently.

You have the option to provide localized user-friendly labels to be displayed in the user interface instead of the unique name of the element, as well as brief localized descriptions of the element. Unlike the unique name, the labels and descriptions are modifiable after creation. According to the language preference of each user, TIBCO EBX® will display the corresponding localized label and description of the element.

Deleting elements of the data model

Any element can be deleted from the data structure using the down arrow corresponding to its entry.

When deleting a group or table that is not using a reusable type, the deletion is performed recursively, removing all its nested elements.

Duplicating existing elements

To duplicate an element, click the down arrow corresponding to its entry. You must provide a name for the duplicated element that is unique at its level of the data structure. All other properties are copied from the source element.

The duplicated element is added to the data model at the same level as the element from which it was copied, appended after the existing elements. If you are duplicating a table or group containing other elements, all nested elements are copied with their original names.

> **Note**
>
> If you duplicate a primary key field, the properties of the field are maintained, but the new field is not automatically added to the primary key.

Moving elements

To reorder an element within its current level of the data structure, click the down arrow corresponding to its entry and select 'Move'. Then, select the left-arrow button corresponding to the field *before which* you want to move the current element.

> **Note**
>
> It is not possible to move an element to a position outside of its level in the data structure.

8.2 Reusable types

Reusable types are shared element definitions that are created once and can be reused in different places in the data model.

> **Note**
>
> If you modify the definition of a reusable type in the 'Simple data types' or 'Complex data types' section, you will modify the structure of all elements based on that reusable type. The structure of a groups or table using a reusable type is shown as read-only. To edit the structure of the associated reusable type, you have to access the type from the 'Simple data types' or 'Complex data types' section.

Defining a reusable type

From the down arrow menu of 'Simple data types' and 'Complex data types' in the navigation pane, you can define simple and complex reusable types that will be available for creating more elements which share the same structural definition and properties. Alternatively, you can convert existing tables and groups into reusable types using their corresponding down arrow menus.

It is possible to see the elements that are using a reusable type by selecting 'References to this type' on the specific page of each data type, under 'Simple data types' and 'Complex data types' in the navigation pane. A table then displays all elements that are based on this type. If a data type is not used by any elements, you can select the 'Delete type' from its down arrow menu to delete the reusable type.

Using a reusable type

The structure of new elements can be defined using reusable types. To do so, select an existing reusable type in the element creation form. The created element will then share the type definition of the reusable type.

Including data types defined in other data models

You can also share reusable types between multiple data models. By configuring the inclusion of an external data model, you can use the data types defined in that data model to create new elements in the data structure the same way as using locally defined reusable types.

> **Note**
>
> As the names of data types must be unique across all locally defined as well as all included types, you cannot create new reusable types with the same name as a data type in an included data model. Similarly, you cannot include an external data model that defines a data type with the same name as a locally defined reusable type or a data type in another included data model.

Included data types appear in the sections 'Included simple data types' and 'Included complex data types' in the navigation panel. You can view the details of these included reusable types; however, they can only be edited locally in their original data models.

See Included data models [p 53] for more information.

8.3 Data model element creation details

Creating fields

When creating a new field, you must select its data type, which will define the data type of the values based upon this field. The data type of the field cannot be changed once the field has been created.

While creating a field, it is also possible to designate it as a foreign key, a mandatory field, and, if created under a table, a primary key.

Creating tables

While creating a table, you have the option to create the new table based on an existing reusable type. See Reusable types [p 57] for more information.

Every table requires specifying at least one primary key field, which you can create as a child element of the table from the navigation pane.

Creating groups

While creating a group, you have the option to create the new group based on an existing reusable type. See Reusable types [p 57] for more information.

Creating primary key fields

At least one primary key is required for every table. You can create a primary key field for a table by creating it as a child element under the table's entry in the 'Data structure' tree.

Besides creating a new field directly as a primary key, you can add any existing child field of a table to the definition of its primary key on the 'Primary key' tab of the table's 'Advanced properties'.

Creating or defining foreign key fields

Foreign key fields have the data type 'String'. You can create a foreign key field for a table by creating it as a child element under the table's entry in the 'Data structure' tree. You can also convert an existing field of type 'String' into a foreign key. To convert an existing field of type 'String' into a foreign key, enable 'Foreign key constraint' in the field's 'Advanced controls' and define the associated parameters.

Whether creating a foreign key directly or from an existing field, you must define the table that contains the records to be referenced.

Creating associations

An association allows defining semantic links between tables. You can create an association by creating it as a child element under the table's entry in the 'Data structure' tree and by selecting 'association' in the form for creating a new element. An association can only be defined inside a table. It is not possible to convert an existing field to an association.

When creating an association, you must specify the type of association. Several options are available:

- Inverse relationship of a *foreign key*. In this case, the association element is defined in a *source table* and refers to *a target table*. It is the counterpart of the foreign key field, which is defined in the target table and refers back the source table. You must define the foreign key that references the parent table of the association.

- Over a *link table*. In this case, the association element is defined in a *source table* and refers to a *target table* that is inferred from a *link table*. This link table defines two foreign keys: one referring to the source table and another one referring to the target table. The primary key of the link table must also refer to auto-incremented fields and/or the foreign key to the source or target table of the association. You must define the link table and these two foreign keys.
- Using an *XPath predicate*. In this case, the association element is defined in a *source table* and refers to a *target table* that is specified using a *path*. An *XPath expression* is also defined to specify the criteria used to associate a record of the current table to records of the target table. You must define the target table and an XPath expression.

In all types of association, we call *associated records* the records in the target table that are semantically linked to records in the source table.

Once you have created an association, you can specify additional properties. For an association, it is then possible to:

- Filter associated records by specifying an additional XPath filter. It is only possible to use fields from the source and the target table when defining an XPath filter. That is, if it is an association other a link table it is not possible to use fields of the link table in the XPath filter. You can use the available wizard to select the fields that you want to use in your XPath filter.
- Configure a tabular view to define the fields that must be displayed in the associated table. It is not possible to configure or modify an existing tabular view if the target table of the association does not exist. If a tabular view is not defined, all columns that a user is allowed to view according to the granted access rights are displayed.
- Define how associated records are to be rendered in forms. You can specify that associated records are to be rendered either directly in the form or in a specific tab. By default, associated records are rendered in the form at the same position of the association in the parent table.
- Hide/show associated records in data service 'select' operation. By default associated records are hidden in data service 'select' operation.
- Specify the minimum and maximum numbers of associated records that are required. In associated datasets, a validation message of the specified severity is added if an association does not comply with the required minimum or the maximum numbers of associated records. By default, the required minimum and the maximum numbers of associated records are not restricted.
- Add validation constraints using XPath predicates to restrict associated records. It is only possible to use fields from the source and the target table when defining an XPath predicate. That is, if it is an association over a link table it is not possible to use fields of the link table in the XPath predicate. You can use the available wizard to select the fields that you want to use in your XPath predicate. In associated datasets, a validation message of the specified severity is added when an associated record does not comply with the specified constraint.

8.4 Modifying existing elements

Removing a field from the primary key

Any field that belongs to the primary key can be removed from the primary key on the 'Primary key' tab of the table's 'Advanced properties'.

See primary key [p 35] in the glossary.

CHAPTER 9

Properties of data model elements

After the initial creation of an element, you can set additional properties in order to complete its definition.

See also *Data validation controls on elements* *[p 75]*

This chapter contains the following topics:

1. Basic element properties
2. Advanced element properties

9.1 Basic element properties

Common basic properties

The following basic properties are shared by several types of elements:

Information	Additional non-internationalized information associated with the element.
Minimum number of values	Minimum number of values for an element. As primary keys cannot be multi-valued, they must have this property set to '1' or 'undefined'. The minimum number of values is automatically set to '0' when defining the field as a selection node.
Maximum number of values	Maximum number of values for an element. When set to a value greater than '1', the element becomes multi-valued. As primary keys cannot be multi-valued, they must have this property set to '1' or 'undefined'. For tables, the maximum number of values is automatically set to 'unbounded' upon creation. The maximum number of values is automatically set to '0' when defining the field as a selection node.
Validation rules	This property is available for tables and fields in tables except `Password` fields, reusable types, fields in complex reusable types, and selection nodes. Used to define powerful and complex validation rules with the help of the provided XPath 1.0 criteria editor. See Criteria editor [p 509] for more information. This can be useful if the validation of the value depends on complex criteria or on the value of other fields. It is also possible to indicate that a rule defines a verification for a value that is mandatory under certain circumstances. In this case a value is mandatory if the rule is not satisfied. See Constraint on 'null' values [p 943] for more information. Using the associated wizard, you can define localized labels for the validation rule, as well as define a localized message with severity to be displayed if the criteria is not met. When defining the severity of the validation message it is possible to indicate whether an input that would violate a validation rule will be rejected or not when submitting a form. The error management policy is only available on validation rules defined on a field and when the severity is set to 'error'. If the validation rule must remain valid, then

any input that would violate the rule will be rejected and the values will remain unchanged. If errors are allowed, then any input that would violate the rule will be accepted and the values will change. If not specified, the validation rule always blocks errors upon the form submission by default.

If a validation rule is defined on a table, it will be considered as a 'constraint on table' and each record of the table will be evaluated against it at runtime. See Constraints on table [p 944] for more information.

Basic properties for fields

The following basic properties are specific to fields:

Default value	Default value assigned to this field. In new data creation forms, the default value appears automatically in the user input field. The default value must comply with the defined type of the field. See Default value [p 961] for more information.
Conversion error message	Internationalized messages to display to users when they enter a value that is invalid for the data type of this field.
Computation rule	This property is available for fields in tables, except in reusable types. Defines a rule for computing the value of the field using the provided XPath 1.0 editor. See criteria editor [p 509] This can be useful if the value depends on other values in the same record, but does not require a programmatic computation. The following limitations exist for computation rules: • Computation rules can only be defined on simple fields inside a table. • Computation rules cannot be defined on fields of type `OResource` or `Password`. • Computation rules cannot be defined on selection nodes and primary key fields. • Computation rules cannot be defined when accessing an element from the validation report.

9.2 Advanced element properties

Common advanced properties

The following advanced properties are shared by several types of elements:

Default view and tools > Visibility

Specifies whether or not this element is shown in the default view of a dataset, in the text search of a dataset or in the data service "select" operation.

- Model-driven view

  Specifies whether or not the current element is shown in the default tabular view of a table, the default record form of a table, and in the default view of a dataset if the current element is a table. Default dataset view, tabular view and default record form generated from the structure of the data model. If the current element is inside a table, then setting the property to 'Hidden' will hide the element from the default tabular view and default record form of the table without having to define specific access permissions. Current element will still be displayed in the view configuration wizard to be able to create a custom view that displays this element. If the current element is a table, then setting the property to 'Hidden' will hide the table from the default view of a dataset without having to define specific access permissions. This property is ignored if it is set on an element that is neither a table nor in a table.

- All views

  Specifies whether or not the current element is shown in all views of a table in a dataset. Setting the property to 'Hidden in all views' will hide the element in all views of the table, whether tabular (default tabular view included) or hierarchical, without having to define specific access permissions. The current element will also be hidden in the view configuration wizard. That is, it won't be possible to create a custom view that will display this element. This property is ignored if it is set on an element that is not in a table. This property is not applied on forms. That is, setting the property to 'Hidden in all views' will not hide the element in a record form but only in views.

- Structured search tool

  Specifies whether or not the current element is shown in a dataset structured search tool. Setting the property to 'Hidden in structured search' will hide the element in the structured search tool of a dataset. The element

will remain searchable in the quick search tool. This property is ignored if it is set on an element that is not in a table.

- Quick search

  Specifies whether or not the current element is included in the quick search tool. Setting the property to 'Excluded from Quick search' will exclude the element in the quick search tool of a dataset. The element will remain searchable in the structured search tool. This property is ignored if it is set on an element that is not in a table.

- Data services

  Specifies whether or not the current element is shown in the data service select operation. Setting the property to 'Excluded from Data Services' will hide the element in the data service "select" operation. This property is ignored if it is set on an element that is not in a table.

See Default view [p 963] in the Developer Guide.

Default view and tools > Widget

Defines the widget to be used. A widget is an input component that is displayed in forms in associated datasets. If undefined, a default widget is displayed in associated datasets according to the type and properties of the current element. It is possible to use a built-in widget or a custom widget. A custom widget is defined using a Java API to allow the development of rich user interface components for fields or groups. Built-in and custom widgets cannot be defined on a table or an association. It is forbidden to define both a custom widget and a UI bean. It is forbidden to define on a foreign key field both a custom widget and a combo-box selector.

See `UIWidgetFactory`[API] for more information.

Default view and tools > Combo-box selector

Specifies the name of the published view that will be used in the combo-box selection of the foreign key. A selection button will be displayed at the bottom right corner of the drop-down list. When defining a foreign key, this feature allows accessing an advanced selection view through the 'Selector' button that opens the advanced selection view, from where sorting and searching options can be used. If no published view is defined, the advanced selection view will be disabled. If the path of the referenced table is absolute then only the published views corresponding to this table will be displayed. If the path of the referenced table is relative then all the published views associated with the data model containing the target table will be displayed. This property can only be set if no custom widget is defined.

See Defining a view for the combo box selector of a foreign key [p 965] in the Developer Guide.

UI bean

> **Attention**
> From version TIBCO EBX® 5.8.0, it is recommended to use widgets instead of UI Beans. Widgets provide more features than UI Beans, and no further evolution will be made on UI beans. See _widget [p 65] for more information.

This property is available for all elements except tables and associations. Specifies a Java class to customize the user interface associated with this element in a dataset. A UI bean can display the element differently and/or modify its value by extending the `UIBeanEditor`[API] class in the Java API.

Transformation on export

This property is available for fields and for groups that are terminal nodes. Specifies a Java class that defines transformation operations to be performed when exporting an associated dataset as an archive. The input of the transformation is the value of this element.

See `NodeDataTransformer`[API] for more information.

Access properties

Defines the access mode for the current element, that is, if its data can be read and/or written.

- 'Read & Write' corresponds to the mode `RW` in the data model XSD.
- 'Read only' corresponds to the mode `R-` in the data model XSD.
- 'Not a dataset node' corresponds to the mode `CC` in the data model XSD.
- 'Non-terminal node' corresponds to the mode `--` in the data model XSD.

See Access properties [p 961] in the Developer Guide.

Comparison mode

Defines the comparison mode associated with the element, which controls how its differences are detected in a dataset.

- 'Default' means the element is visible when comparing associated data.
- 'Ignored' implies that no changes will be detected when comparing two versions of modified content (records or datasets).

  During a merge, the data values of ignored elements are not merged even if the content has been modified.

	However, values of ignored data sets or records being created during the operation are merged. During an archive import, values of ignored elements are not imported when the content has been modified. However, values of ignored datasets or records being created during the operation are imported. See Comparison mode [p 966] in the Developer Guide.
Apply last modifications policy	Defines if this element must be excluded from the service allowing to apply the last modifications that have been performed on a record to the other records of the same table. • 'Default' means that the last modification on this element can be applied to other records. • 'Ignored' implies that the last modification on this element cannot be applied to other records. This element will not be visible in the apply last modifications service. See Apply last modifications policy [p 966] in the Developer Guide.
Node category	Defines a category for this element. Categories allow controlling the visibility of data in a dataset to users. A node with the category 'Hidden' is hidden by default. Restriction: category specifications other than 'Hidden' do not apply to table record nodes. See Categories [p 967] in the Developer Guide.

Advanced properties for fields

The following advanced properties are specific to fields.

Check null input

Implements the property `osd:checkNullInput`. This property is used to activate and check a constraint on `null` at user input time.

By default, in order to allow for temporarily incomplete input, the check for mandatory elements is not performed upon user input, but only upon dataset validation. If a mandatory element must be checked immediately upon user input, set this property to 'true'.

> **Note**
>
> A value is considered mandatory if the 'Minimum number of values' property is set to '1' or greater. For terminal elements, mandatory values are only checked in activated datasets. For non-terminal elements, the values are checked regardless of whether the dataset is activated.

See Constraints, triggers and functions [p 948] in the Developer Guide.

Trim whitespaces

Trim white spaces

Implements the property osd:trim. This property is used to indicate whether leading and trailing white spaces must be trimmed upon user input. If this property is not set, leading and trailing white spaces are removed upon user input.

See Whitespace handling upon user input [p 949] in the Developer Guide.

UI bean

See Common advanced properties [p 66].

Function (computed value)

This property is available for non-primary key fields. Specifies a Java class that computes the value of this field programmatically. This can be useful if the value of the field depends on other values in the repository, or if the computation of the value needs to retrieve data from a third-party system.

A function can be created by implementing the `ValueFunction`[API] interface.

Disable validation

Specifies if the constraints defined on the field must be disabled. This property can only be defined on function fields. If true, cardinalities, simple and advanced constraints defined on the field won't be checked when validating associated datasets.

Transformation on export

See Common advanced properties [p 66].

Access properties

See Common advanced properties [p 66].

Auto-increment

This property is only available for fields of type 'Integer' that are contained in a table. When set, the value of the field is automatically calculated when a new record is created. This can be useful for primary keys, as it generates a unique identifier for each record. Two attributes can be specified:

Start value	Value with which to begin the auto-increment. If this attribute is not specified, the default value is '1'.
Increment step	Amount the value is incremented based on the previous value of the auto-increment. If this attribute is not specified, the default is value is '1'.
Disable auto-increment checks	Specifies whether to disable the check of the auto-incremented field value in associated datasets against the maximum value in the table being updated.

Auto-incremented values have the following behavior:

- The computation and allocation of the field value are performed whenever a new record is inserted and the field value is yet undefined.
- No allocation is performed if a programmatic insertion already specifies a non-`null` value. Consequently, the allocation is not performed for a record insertion in occulting or overwriting modes.
- If an archive import specifies the value, the imported value takes precedence.
- Whenever possible, the newly allocated value is unique in the scope of the repository.

  That is, the uniqueness of the allocation spans over all datasets based upon this data model, in any dataspace in the repository. The uniqueness across different dataspaces facilitates the merging of child dataspaces parent dataspaces while reasonably avoiding conflicts when a record's primary key includes the auto-incremented value.

  Despite this policy, a specific limitation exists when a mass update transaction assigning specific values is performed concurrently with a transaction that allocates an auto-incremented value on the same field. It is possible that the latter transaction will allocate a value that has already been set in the former transaction, as there is no locking between different dataspaces.

See Auto-incremented values [p 953] in the Developer Guide.

Default view

See Common advanced properties [p 64].

Node category

See Common advanced properties [p 67].

Inherited field

Defines a relationship from the current field to a field in another table in order to automatically fetch its field value.

Source record	A foreign key or white space-separated sequence of foreign keys that leads from the current element to the record from which to inherit this field's value. If this property is not specified, the current record is used as the source record for the inheritance.
Source element	XPath of the element in the source record from which to inherit this field's value. The source element must be terminal, belong to the record described by 'Source record', and its type must match the type of this field. This property is mandatory when using field inheritance.

See inheritance [p 37] in the glossary.

For more information, see also Inherited fields [p 482].

Advanced properties for tables

The following advanced properties are specific to tables.

Table

Primary key	A list of fields in the table that compose the table's primary key. You can add or remove primary key fields here, as in the 'Data structure' view. Each primary key field is denoted by its absolute XPath notation that starts under the table's root element. If there are several elements in the primary key, the list is white-space delimited. For example, "/name /startDate".
Presentation	Specifies how records are displayed in the user interface of this table in a dataset.
Presentation **> Record labeling**	Defines the fields to provide the default and localized labels for records in the table. Can also specify a Java class to set the label programmatically, or set the label in a hierarchy. This Java class must implement either the `UILabelRenderer`[API] interface or the `UILabelRendererForHierarchy`[API] interface. **Attention:** Access rights defined on associated datasets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
Presentation **> Default rendering for groups in forms**	Specifies the default display rendering mode of the groups contained in this table. If nothing is defined here, the default policy set in the Administration area will be used to display groups. See Record form: rendering mode for nodes [p 679] in the Administration Guide. **Enabled rendering for groups** Specifies a display rendering mode to be enabled for groups in the table in addition to the modes 'expanded' and 'collapsed', which are always available. Tabs must be enabled on the table to have the option to display groups as tabs. Similarly, links must be enabled to have the option to display groups as links. **Default rendering for groups** Specifies the default display rendering mode to use for the groups contained in this table. If a group does not specify a default mode then the default mode defined for this table will be used. Links must be enabled to define the default rendering mode as 'Link'. Select a rendering mode according to network and browser performance. Link mode is lighter

	as its content is not displayed on the same page, whereas the expanded and collapsed modes are heavier. **Note:** When the tabs view is enabled on a table, any groups that would otherwise have used links are automatically converted to collapsed mode. This is done to avoid the inherent display layout complexities that would arise from links and tabs coexisting in the same user interface.
***Presentation* > Specific rendering of forms**	Defines a specific rendering for customizing the record form in a dataset. See `UIForm`[API] and `UserServiceRecordFormFactory`[API] for more information.
Toolbars	Defines the toolbars to use in this table. Toolbars can be edited in the *Configuration* > *Toolbars* section. **Tabular view top:** Defines the toolbar to use on top of the default table view. **Tabular view row:** Defines the toolbar to use on each row of the default table view. **Tile view item:** Defines the toolbar to use on each item of the tile view. **Record top:** Defines the toolbar to use in the record form. **Hierarchy top:** Defines the toolbar to use in the default hierarchy view of the table. See Toolbars [p 86] for more information.
History	Specifies when historization is to be performed, and the level of guarantee requested. The available history profiles can be edited in **Administration > History and logs**. See History configuration in the repository [p 461] for more information.
Specific filters	Defines record display filters on the table.
Actions	Specifies the actions that are allowed on the table in associated datasets. By default, all actions are allowed unless specific access rights are defined in a dataset.

Uniqueness constraints

Indicates which fields or set of fields must be unique across the table.

Triggers

Specifies Java classes that defines methods to be automatically executed when modifications are performed on the table, such as record creation, updates, deletion, etc.

A built-in trigger for starting data workflows is included by default.

A built-in trigger to define a script for a trigger is also included by default.

See Triggers [p 952] in the Developer Guide.

Access properties

See Common advanced properties [p 66].

Default view

See Common advanced properties [p 64].

Node category

See Common advanced properties [p 67].

Advanced properties for groups

The following advanced properties are specific to groups.

Value container class (JavaBean)

Specifies a Java class to hold the values of the children of this group. The Java class must conform to the JavaBean standard protocol. That is, each child of the group must correspond to a JavaBean property in the class, and all properties must have getter and setter accessors defined.

UI bean

See Common advanced properties [p 66].

Transformation on export

See Common advanced properties [p 66].

Access properties

See Common advanced properties [p 66].

Default view

Visibility	See Common advanced properties [p 64].
Rendering in forms	Defines the rendering mode of this group. If this property is not set, then the default view for groups specified by the container table will be used. 'Tab' and 'Link' are each only available when the container table enables it. ***Tab position*** This attribute specifies the position of the tab with respect to all the tabs defined in the model. This position is used for determining tab order. If a position is not specified, the tab will be displayed according to the position of the group in the data model.

Node category

See Common advanced properties [p 67].

Related concepts *Data validation controls on elements [p 75]*

CHAPTER 10

Data validation controls on elements

After the initial creation of an element, you can set additional controls in order to complete its definition.

See also *Properties of data model elements* *[p 61]*

This chapter contains the following topics:

1. Simple content validation
2. Advanced content validation

10.1 Simple content validation

Using the following static controls on a field, you can establish simple validation rules on its content. The controls available for a given field are dependent on its data type.

Fixed length	The exact number of characters required for this field.
Minimum length	The minimum number of characters allowed for this field.
Maximum length	The maximum number of characters allowed for this field.
Pattern	A regular expression pattern that the value of the field must match. It is not possible to simultaneously define a pattern for both a field and its data type.
Decimal places	The maximum number of decimal places allowed for this field.
Maximum number of digits	The maximum total number of digits allowed for this integer or decimal field.
Enumeration	Defines a list of predefined possible values for this field. If enumerations are defined in both a field and its type, then the enumeration of this field in associated datasets is replaced by the intersection of these two enumerations.
Greater than [constant]	Defines the minimum value allowed for this field.
Less than [constant]	Defines the maximum value allowed for this field.

See XML schema supported facets [p 935].

10.2 Advanced content validation

Using the following dynamic and contextual controls on an element, you can establish advanced validation rules of its content. The controls available for a given element are dependent on the type of element and its data type, if it has one.

See also *Dynamic constraints* *[p 939]*

Foreign key constraint	
Table	Defines the table referenced by the foreign key. A foreign key references a table in the same dataset by default. It can also reference a table in another dataset in the same dataspace, or a dataset in a different dataspace.
Mode	Location of the table referenced by the foreign key. 'Default': current data model. 'Other dataset': different dataset, in the same dataspace. 'Other dataspace': dataset in a different dataspace.
Referenced table	XPath expression describing the location of the table. For example, `/root/MyTable`.
Referenced dataset	Required if the table is located in another dataset. The unique name of the dataset containing the referenced table.
Referenced dataspace	Required if the table is located in another dataspace. The unique name of the dataspace containing the referenced table.
Label	Defines fields to provide the default and localized labels for records in the table. Allows as well to customize the display of the specified label in breadcrumb. Can also specify a Java class to set the label programmatically if 'XPath expression' is set to 'No'. This Java class must implement the `TableRefDisplay`[API] interface of the Java API. **Attention:** Access rights defined on associated datasets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
Filter	Defines a foreign key filter using an XPath expression. Can also specify a Java class that implements the `TableRefFilter`[API] interface of the Java API.
Greater than [dynamic]	Defines a field to provide the minimum value allowed for this field.

Less than [dynamic]	Defines a field to provide the maximum value allowed for this field.
Fixed length [dynamic]	Defines a field to provide the exact number of characters required for this field.
Minimum length [dynamic]	Defines a field to provide the minimum number of characters allowed for this field.
Maximum length [dynamic]	Defines a field to provide the maximum number of characters allowed for this field.
Excluded values	Defines a list of values that are not allowed for this field.
Excluded segment	Defines an inclusive range of values that are not allowed for this field. **Minimum excluded value:** Lowest value not allowed for this field. **Maximum excluded value:** Highest value not allowed for this field.
Specific constraint (component)	Specifies one or more Java classes that implement the `Constraint`[API] interface of the Java API. See Programmatic constraints [p 943] for more information.
Specific enumeration (component)	Specifies a Java class to define an enumeration. The class must define an ordered list of values by implementing the `ConstraintEnumeration`[API] interface of the Java API.
Enumeration filled by another node	Defines the possible values of this enumeration using a reference to another list or enumeration element.
Dataspace set configuration	Define the dataspaces that can be referenced by a field of the type Dataspace identifier (osd:dataspaceKey). If a configuration is not set, then only opened branches can be referenced by this field by default. • Includes Specifies the dataspaces that can be referenced by this field. **Pattern:** Specifies a pattern that filters dataspaces. The pattern is checked against the name of the dataspaces. **Type:** Specifies the type of dataspaces that can be referenced by this field. If not defined, this restriction is applied to branches.

Include descendants: Specifies if children or descendants of the dataspaces that match the specified pattern are included in the set. If not defined, this restriction is not applied to child dataspaces. If "None" then neither children nor descendants of the dataspaces that match the specified pattern are included. If "All descendants" then all descendants of the dataspaces that match the specified pattern are included. If "All descendant branches" then all descendant branches of the dataspaces that match the specified pattern are included. If "All descendant snapshots" then all descendant snapshots of the dataspaces that match the specified pattern are included. If "Child branches" then only direct branches of the dataspaces that match the specified pattern are included. If the current dataspace is a version, includes the branches that are the direct children of this version; if the current dataspace is a branch, includes the branches that are the direct children of the versions which are children of this branch. If "Child snapshots" then only direct snapshots of the dataspaces that match the specified pattern are included. If the current dataspace is a branch, includes the snapshots that are the direct children of this branch; if the current dataspace is a version, includes the versions that are the direct children of the branches which are children of this version.

- Excludes

  Specifies the dataspaces that cannot be referenced by this field. Excludes are ignored if no includes are defined.

  Pattern: Specifies a pattern that filters dataspaces. The pattern is checked against the name of the dataspaces.

  Type: Specifies the type of dataspaces that can be referenced by this field. If not defined, this restriction is applied to branches.

  Include descendants: Specifies if children or descendants of the dataspaces that match the specified pattern are included in the set. If not defined, this restriction is not applied to child dataspaces. If "None" then neither children nor descendants of the dataspaces that match the specified pattern are included. If "All descendants" then all descendants of the dataspaces that match the specified pattern are included. If "All descendant branches" then all descendant branches of the dataspaces that match the specified pattern are included. If "All descendant snapshots" then all descendant snapshots of the dataspaces that match the specified pattern are included. If "Child branches" then

only direct branches of the dataspaces that match the specified pattern are included. If the current dataspace is a version, includes the branches that are the direct children of this version; if the current dataspace is a branch, includes the branches that are the direct children of the versions which are children of this branch. If "Child snapshots" then only direct snapshots of the dataspaces that match the specified pattern are included. If the current dataspace is a branch, includes the snapshots that are the direct children of this branch; if the current dataspace is a version, includes the versions that are the direct children of the branches which are children of this version.

- Dataspace filter

  Specifies a filter to accept or reject dataspaces in the context of a dataset or a record. This filter is only used in the dedicated input component that is associated to this field. That is, this filter is not used when validating this field. A specific constraint can be used to perform specific controls on this field. A filter is defined by a Java class that implements the `DataspaceSetFilter`[API] interface of the Java API.

Dataset set configuration

Define the datasets that can be referenced by a field of the type Dataset identifier (osd:datasetName).

- Includes

  Specifies the datasets that can be referenced by this field.

  Pattern: Specifies a pattern that filters datasets. The pattern is checked against the name of the datasets.

  Include descendants: Specifies if children or descendants of the datasets that match the specified pattern are included in the set.

- Excludes

  Specifies the datasets that cannot be referenced by this field. Excludes are ignored if no includes are defined.

  Pattern: Specifies a pattern that filters datasets. The pattern is checked against the name of the datasets.

  Include descendants: Specifies if children or descendants of the datasets that match the specified pattern are included in the set.

- Filter

  Specifies a filter to accept or reject datasets in the context of a dataset or record. This filter is only used in the dedicated input component that is associated to this field. That is, this filter is not used when validating

	this field. A specific constraint can be used to perform specific controls on this field. A filter is defined by a Java class that implements the `DatasetSetFilter`[API] interface of the Java API.

Validation properties

Each constraint not using a specific Java class can define localized validation messages with a severity using the following properties:

Validation	Defines a localized validation message with a user-defined severity level.
Severity	Defines the severity level of the validation message. Possible values are 'Error', 'Warning', and 'Information'.
Error management policy	Specifies the behavior of the constraint when validation errors occur. It is possible to specify that the constraint must always remain valid after an operation (dataset update, record creation, update or deletion), or when a user submits a form. In this case, any input or operation that would violate the constraint will be rejected and the values will remain unchanged. If not specified, the constraint only blocks errors upon form submission by default, except for foreign key constraints in relational data models where errors are prevented for all operations by default. This option is only available upon static controls, exclude values, exclude segment and foreign key constraints. On foreign key constraints the error management policy does not concern filters. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case updates are not rejected and a validation error occurs. It is not possible to specify an error management policy on structural constraints that are defined in relational data models, when table history or replication is activated. That is, setting this property on fixed length, maximum length, maximum number of digits and decimal place constraints will raise an error at data model compilation because of the underlying RDBMS blocking constraints validation policy. This property is ineffective when importing archives. That is, all blocking constraints, excepted structural constraints, are always disabled when importing archives.
Message	Defines the message to display if the value of this field in a dataset does not comply with this constraint. If specifying a custom message, you may optionally provide localized variants.

Related concepts *Properties of data model elements* *[p 61]*

CHAPTER 11

Data model extensions

This chapter contains the following topics:

1. Extensions used by the data model
2. Indexing and search strategies
3. Business objects
4. Toolbars
5. Entity mappings
6. Replication of data to relational tables
7. Node confidentiality

11.1 Extensions used by the data model

On any data model, it is possible to specify some *extensions* to be used. These extensions will have the capacity to define additional features around the data model and to enrich the current data model after the publication by adding properties and constraints to the data model elements. Data model extensions are displayed in the navigation pane under the section 'Extensions'.

The following extensions are automatically enabled on new data models:

- Toolbars [p 86]
- Entity mappings [p 92]
- Replications [p 92]
- Indexing and search strategies [p 84]
- Business objects [p 84]
- Custom forms [p 973]
- Scripts [p 1318]
- Record permissions [p 1301]
- Node confidentiality [p 93]

Some extensions are optional and defined in dedicated datasets. These extensions can be disabled or enabled through the user interface.

Optional data model extension can be disabled by selecting the action 'Disable extension' from the menu to the left of extension to disable.

> **Note**
>
> When disabling an extension its dedicated dataset is also deleted and it is not possible to recover it after its deletion

From the menu to the left of 'Extensions' entry in the navigation pane, you can enable an extension by selecting the action 'Enable an extension'. This option is only displayed if some extensions are available. When selecting this action a form displays the available data model extensions.

Once a data model extension is activated then it is displayed under the entry 'Extensions' in the navigation pane and can be edited as a regular dataset. That is, the configuration of an extension is embedded in a dedicated dataset that benefits of all EBX® features available on datasets.

11.2 Indexing and search strategies

This feature is documented in the chapter Search [p 511].

11.3 Business objects

A business object is an abstract layer that allows manipulating records from several tables within the same data model. The definition of business objects, named business object model, can be performed in the data model via the 'Extensions' section. A business object model is defined on a main table and contains child entities that are defined using the relationships from this main table.

Create a business object model from the *Business objects > Definitions* section of the navigation pane, by clicking on the menu located to the left of *Business Objects*, then selecting the *Create Business*

Object option. Follow the creation wizard to create a business object model. A business object model defines the following information:

Business object name	Business object model's name. The name must be unique in the context of the data model since it is used to uniquely identify the business object model. That is, it is not allowed to create several business objects with the same name.
Label and description	Internationalized labels and descriptions to be displayed to end users.
Main table	Defines the main table of this business object model. Specified path must be absolute (must start with "/"). The main table of this business object model cannot be used if it is already a part of another business object model. That is, the main table cannot be used if it is already defined as the main table or as a child entity of another business object model.

Defining the structure of a business object model

A business object model is composed of a main table and child entities. Child entities are defined using the relationships contained in the main table.

Add a child entity under a business object model by clicking on the menu located to the left of the existing business object, and by selecting the *Add child entity* option. Then, follow the creation wizard to create a child entity. The following information must be set:

Relationship to table	This field defines the relationship to the table to add as a child entity. The path to the relationship must be absolute (must start with "/"). Only foreign key constraints and associations by inverse foreign key are supported. Also these relationships must comply with the following constraints: Foreign key constraints cannot be a list or under a list. Associations by inverse foreign key must refer to a foreign key field that cannot be a list or under a list. Also, the target table of the relationship cannot refer to a table that is already used by this business object model or another one. That is, the target table cannot be used if it is already defined as the main table or as a child entity of a business object model.

11.4 Toolbars

A toolbar allows to customize the buttons and menus that are displayed when viewing tables or records in a dataset. The customization of toolbars can be performed in the data model via the 'Extensions' section.

Add a toolbar from the *Toolbars* section of the navigation pane, by clicking on the menu located to the left of *[All elements]*, then selecting the *Create toolbar* option. Follow the creation wizard to create a toolbar. A toolbar defines the following information:

Name	Toolbar's name. The name of the toolbar must be unique in the context of the data model. That is, it is not allowed to create several toolbars with the same name.
Label and description	Internationalized labels and descriptions to be displayed to end users.
Default template	Allows to create a toolbar with the structure of a default toolbar.
Locations	Specifies the locations where the toolbar can be used in associated datasets. **Tabular view top:** Defines the toolbar to use on top of the default table view. **Tabular view row:** Defines the toolbar to use on each row of the default table view. **Tile view item:** Defines the toolbar to use on each item of the tile view. **Record top:** Defines the toolbar to use in the record form. **Hierarchy top:** Defines the toolbar to use in the default hierarchy view of the table. **Known limitations** A toolbar defined to be used as "Tile view item" cannot have more than six items and it can only define action items. The display of those action buttons are always in mode "Icon only".

Defining the structure of a toolbar

A toolbar can define the following elements:

- Action button [p 87]
- Menu button [p 88]
- Separator [p 88]

- Menu group [p 89]
- Action menu item [p 90]
- Sub menu item [p 90]

Add one of these elements under a toolbar or to an existing element by clicking on the menu located to the left of the existing element, and by selecting a creation option in the menu. Then, follow the creation wizard to create an element.

Action button

This type of element allows to associate an action to a button in a toolbar. The action will be triggered when the user clicks on the associated button in one of the toolbars. A *Action button* type element defines the following information:

Service	Defines the service that will be executed when the user clicks on the button. It is possible to select a built-in service, or a user service defined in a module or in the current data model. If the 'Web component' target is selected, the service will have to be declared as available as a web component for toolbars.
Label and description	Internationalized labels and descriptions to be displayed to end users.
Layout	Defines how this element will be displayed in datasets using the toolbar. It is possible to display: the icon only, the text only, text with the icon to the left or text with the icon to the right.
Icon	Icon to display. It is possible to use an icon to choose from a set of suggested icons, or to refer to an icon using a URL.
Relief	Defines how this button will display. The button can be displayed as embossed or flat.
Is highlighted	Indicates if the button should be highlighted by default.

> **Note**
>
> A *Action button* type element can only be created under a *toolbar* type element.

Menu button

This type of element allows to define a menu that will be displayed when the user clicks on the associated button in a toolbar. An element of the *Menu button* type defines the following information:

Label and description	Internationalized labels and descriptions to be displayed to end users.
Layout	Defines how this element will be displayed in datasets using the toolbar. It is possible to display: the icon only, the text only, text with the icon to the left or text with the icon to the right.
Icon	Icon to display. It is possible to use an icon to choose from a set of suggested icons, or to refer to an icon using a URL.
Relief	Defines how this button will display. The button can be displayed as embossed or flat.
Is highlighted	Indicates if the button should be highlighted by default.

> **Note**
>
> An element of the *Menu button* type can only be created under an element of the *toolbar* type.

Separator

This type of element allows to insert a separator in the form of spacing between two elements of a toolbar.

> **Note**
>
> An element of the *Separator* type can only be created under an element of the *toolbar* type.

Menu group

This type of element allows to define a group of elements in a menu. An element of the *Menu group* type defines the following information:

Label and description	Internationalized labels and descriptions to be displayed to end users.
Group type	Specifies the type of menu group to create: - 'Local' allows to create an empty fully customizable menu group. - 'Service group' allows to assign an existing service group to this menu group. - 'Menu builder' allows to assign a predefined menu content to this menu group. Once created, it is not possible to change the type of this menu group.
Service group name	Specifies an existing group of services to reuse. A group is declared in a module and can include other groups of services. All services contained in this group will be displayed to end users in associated datasets.
Menu builder name	Specifies the predefined menu content to assign to this menu group: - 'Default menu "Actions"' has the same content as the default toolbar 'Actions' menu. Standard and custom services are displayed without distinction. - 'Default menu "Actions" (with separator)' has the same menu content as above, but displays differently since standard and custom services are separated (standard services first, then custom services).
Excluded services	Indicates the services to exclude from the group of reused services. These services will not be displayed to end users in associated datasets.
Excluded service groups	Indicates the groups to exclude from the group of services to reuse. Services in excluded groups will not be displayed to end users in associated datasets.
Filtering policy	In case of "Smart filtering", services that are configured in direct access, i.e. via an action button or an action menu item, will be removed from the automatic generation of this group.

> **Note**
>
> An element of the *Menu group* type can only be created under the following elements:

- Menu button

- Sub menu item

Action menu item

This type of element allows to associate an action to a menu item in a toolbar. The action will be triggered when the user clicks on the corresponding item in a menu. An element of the *Action menu item* type defines the following information:

Label and description	Internationalized labels and descriptions to be displayed to end users.
Service	Defines the service that will be executed when the user clicks on the button. It is possible to select a built-in service, or a user service defined in a module or in the current data model. If the 'Web component' target is selected, the service will have to be declared as available as a web component for toolbars.

> **Note**
>
> An element of the *Action menu item* type can only be created under a *Menu group* type element.

Sub menu item

This type of element allows to add a sub menu to a toolbar menu. Un *Sub menu item* defines the following information:

Label and description	Internationalized labels and descriptions to be displayed to end users.

> **Note**
>
> An element of the *Sub menu item* type can only be created under an element of the Menu group type.

Deleting elements

All the elements of a toolbar can be deleted from it by using the menu located to the left of the element to be deleted.

If an element containing other elements is deleted, then the deletion is recursively performed on all elements located below the deleted element.

Duplicating existing elements

To duplicate an element, click on the menu located to the left of the element to duplicate. Specify the name and properties of the duplicated element. All the source element properties are duplicated.

The duplicated element is added on the same level than the original element, in the final position. When an element containing other elements is duplicated, all the sub-elements are duplicated with their properties.

Moving elements

In order to move an element, click on the menu and select the moving option to be used.

Associate with existing tables

To associate a toolbar with existing tables, click on the menu located to the left of the toolbar and select the option *Associate to tables*. This service allows to set the toolbar has the default toolbar of several tables in one shot. To do so, specify the target locations of the toolbar and select the tables or complex data types, that define table properties, to be associated with the toolbar.

Exporting the toolbars

It is possible to export the toolbars defined in the model into an XML document. To do so, select the *XML export* option available in the *Actions* menu of the 'Toolbars' section. Follow the wizard to export the toolbars.

> **Note**
>
> A selection of toolbars can be exported by selecting in the 'Toolbars' section the toolbars to be exported and then by selecting the *XML export* option available in the *Actions* menu. The toolbars can also be exported by using the data model export service. It can be found in the **Data model 'Actions'** [p 45] menu in the navigation pane.

See also *Data model import and export* *[p 95]*

Importing toolbars

It is possible to import existing toolbars from an XML document. To do so, select the *XML import* option available in the *Actions* menu of the 'Toolbars' section. Then follow the wizard to import the toolbars.

> **Note**
>
> The toolbars can also be imported by using the data model import service accessible via the **Data model 'Actions'** [p 45] menu in the navigation pane.

See also *Data model import and export* *[p 95]*

See also *Use of toolbars* *[p 71]*

11.5 Entity mappings

You can refer to tables in SQL requests and data service operations using unique names instead of their paths. To accomplish this, you must define entity mappings to create an association between a table path and a name. When referencing a table you then use this name as a WSDL suffix in data service operations and an alias in SQL requests.

> **Note**
>
> EBX® automatically creates an entity name for the tables defined in a data model. The last step of the table's path is used as the default entity name. Since entity names must be unique in a data model, an error is now raised during compilation if the model contains tables with the same name. To prevent an error in this case, you must explicitly set the entity names.

To define an entity mapping through the user interface, in the navigation pane when viewing a data model, navigate to *Extensions* > *Entity mappings* and create a new record. You can define the following properties:

Table path	Specifies the path of the table in the current data model that is to be referred by this name in SQL requests or WSDL operations.
Entity name	This name is used to suffix all WSDL operation names of the concerned table and as an alias in SQL requests. If undefined for a given table, the last element of the table path is used instead. This name must be unique in the context of this data model.

See also *Data services* *[p 969]*

11.6 Replication of data to relational tables

In any data model, it is possible to define *replication units* for data in the repository to be mirrored to dedicated tables in the relational database. These tables then enable direct access to the data by SQL requests and views.

To define a replication unit through the user interface, create a new record in the 'Replications' table under the extensions' section in the navigation pane. Each replication unit record is associated with a

particular dataset in a given dataspace. A single replication unit can cover multiple tables, as long as they are in the same dataset. A replication unit defines the following information:

Name	Name of the replication unit. This name identifies a replication unit in the current data model. It must be unique.
Dataspace	Specifies the dataspace relevant to this replication unit. It cannot be a snapshot.
Dataset	Specifies the dataset relevant to this replication unit.
Refresh policy	Specifies the data synchronization policy. The possible policies are: • **On commit**: The replicated table content in the database is always up to date with respect to its source table. Every transaction that updates the EBX® source table triggers the corresponding insert, update, and delete statements on the replicated table. • **On demand**: The replicated table in the database is only updated when an explicit refresh operation is performed.
Tables	Specifies the tables in the data model to be replicated in the database. **Table path**: Specifies the path of the table in the current data model that is to be replicated to the database. **Table name in database**: Specifies the name of the table in the database to which the data will be replicated. This name must be unique amongst all replications units.
Aggregated lists	Specifies the properties of the aggregated lists in the table that are replicated in the database. **Path**: Specifies the path of the aggregated list in the table that is to be replicated to the database. **Table name in database**: Specifies the name of the table in the database to which the data of the aggregated list will be replicated. This name must be unique amongst all replications units.

See also *Replication* *[p 469]*

11.7 Node confidentiality

The confidentiality of a node defines whether or not it can be used in a Query or in a Request when hidden for a specific user. By default, all nodes are confidential.

This can be configured in the DMA, by using the "Extensions" tab in the node form editor. It is possible to define the confidentiality at several levels:

- at a data model level (root node): defines the default confidentiality for the data model terminal nodes, unless a more specific definition is set (that is on the table or the node itself).
- at a table level (table node): defines the default confidentiality for the table terminal nodes, unless a more specific definition is set (that is on the node itself).
- at a terminal node level: defines the confidentiality of the node.

For more information, see Defining confidentiality [p 492]

CHAPTER 12

Working with an existing data model

Once your data model has been created, you can perform a number of actions that are available from the **data model 'Actions'** [p 45] menu in the workspace.

This chapter contains the following topics:

1. Validating a data model
2. Data model import and export
3. Duplicating a data model
4. Deleting a data model

12.1 Validating a data model

To validate a data model at any time, select **Actions > Validate** from the navigation pane. The generated report provides the results of the validation. From the validation report, you have the option to update the reported validation status by clicking the **Revalidate** button, or to click the **Reinitialize validation report** button to clear the current validation report associated with the data model in order to be able to rerun a full validation from scratch.

> **Note**
>
> The validation process checks basic data model integrity, but more complex checks are only performed at publication time. More messages may be reported when you try to publish your data model.

See Validation [p 617] for detailed information on incremental data validation.

12.2 Data model import and export

TIBCO EBX® includes built-in data model services to import from and export to XML Schema Document (XSD) files or to archive files (zip). Imports and exports can be performed from the **data model 'Actions'** [p 45] menu of the target data model in the navigation pane. An import or export is always performed in the context of a single data model. That is, during imports, the structure of the target data model is completely replaced with the content of the imported data model (XSD or archive). Similarly, during exports, the entire data model is included in the XSD or archive file.

When importing a data model, the corresponding XSD file must be well-formed and must comply with EBX® validation rules.

If this document declares resources that are located in a module, the module must also be declared in the configuration of the data model. If the module has not been declared, you will not be able to import the data model. See Data model properties [p 52] for more information on declaring modules.

To perform an import select 'Import data model' from the **data model 'Actions'** [p 45] menu of the data model into which you are importing.

You can import an XML Schema Document (XSD) or an archive (zip file) from the local file system. To do so, click the 'From a local document' button in the import wizard and follow the next step:

- **Document name:** path on the local file system of the XSD or archive file to import.

> **Note**
>
> When importing an archive it must contain only one XML Schema Document (XSD) at its root. XML documents related to data model extensions must also be located at the root of the archive. These XML documents are automatically imported if they comply with the naming of the extensions supported and registered in EBX®.

You can also import a data model in an XSD that is packaged in a module. The import of a data model in XSD format from a module uses the following properties:

Module	Module in which the data model is packaged.
Module path	Path to the module containing the data model.
Source path	Path to Java source used to configure business objects and rules. This property is required if the data model being imported defines programmatic elements.
Model	The data model in the module to import.

> **Note**
>
> Imported XSD files must be encoded in 'UTF-8'. Exported XSD files are always encoded in 'UTF-8'.

To perform an export select 'Export data model' from the **data model 'Actions'** [p 45] menu of the data model you want to export.

12.3 Duplicating a data model

To duplicate a data model, select 'Duplicate' from the **data model 'Actions'** [p 45] menu for that data model. You must give the new data model a name that is unique in the repository.

12.4 Deleting a data model

To delete a data model, select 'Delete' from the **data model 'Actions'** [p 45] menu for that data model. When you delete a data model, all of its existing publications will remain and continue to be accessible

to their associated datasets. If you recreate a new data model with the same name as one that was previously deleted, the new data model will be reassociated with all the existing publications in the repository. At publication time of the new data model, you will have the opportunity to confirm the replacement of an existing publication.

> **Note**
>
> Only an administrator can clean up the publications of deleted data models in the 'Administration' area.

See Publishing data models [p 99] for more information on the publication process.

CHAPTER 13

Publishing a data model

This chapter contains the following topics:

1. About publications
2. Publication modes
3. Embedded publication mode

13.1 About publications

Each dataset based on an **embedded data model** in the TIBCO EBX® repository is associated with a publication of a data model, rather than directly to the data model itself. The first time you publish a data model using the **Publish** button in the navigation pane, a publication is created. Once the publication exists, datasets can be created based upon it.

> **Note**
>
> The **Publish** button is only displayed to users who have permission to publish the data model. See Data model permissions [p 51] for more information.

As datasets are based on publications, any modifications you make to a data model will only take effect on existing datasets when you republish to the publication associated with those datasets. When you republish a data model to an existing publication, all existing datasets associated with that particular publication are updated.

13.2 Publication modes

You can publish a data model using either 'Embedded' mode or 'In module' mode. The 'Embedded' publication mode generates a publication that is managed and persisted within the EBX® repository and thus has associated versioning and rollback functionality. The 'In module' publication mode creates an XML Schema Document contained in a module that is not managed or versioned within the repository.

Depending on the configuration of the data model, EBX® automatically determines the publication process to use when you click the **Publish** button in the navigation pane. When a data model specifies the publication mode 'In module' and provides a target XSD to be generated, the publication process generates an XSD file contained in the module defined in the configuration.

13.3 Embedded publication mode

The first time you publish a given embedded data model, a new publication with the same name as your data model is automatically created in the repository. If more than one publication has already been created for this model, you will need to select a target publication for this process.

See Viewing and creating publications [p 100] for more information on the use of different publications.

During the publication process, you have the opportunity to review the structural differences being introduced by the current publication in a side-by-side comparison view, if the data model has previously been published.

The publication process also offers the option to create a read-only snapshot of the current state of the data model for reference purposes. This snapshot can be useful if the data model ever needs to be rolled back to the current state after other modifications have been made.

> **Note**
>
> Snapshots, which are static archives of the state of the data model, must not be confused with data model *versions*, which act instead as parallel evolving branches of the data model. See Versioning embedded data models [p 101] for more information on data model versions.

Viewing and creating publications

To access the publications that exist for the current data model, select 'Manage publications' from its **data model 'Actions'** [p 45] menu in the navigation pane. From there, you can view the details of the publications and create new publications.

In certain cases, it may be necessary to employ several publications of the same data model, in order to allow datasets to be based on different states of that data model. Multiple publications must be handled carefully, as users will be asked to select an available publications to target when publishing if more than one exists. The action to create a new publication is only available to users who belong to the 'Administrator' role.

To create a new publication, select 'Manage publications' from the **data model 'Actions'** [p 45] menu of the data model in the navigation pane, then click the **Create publication** button. The name you give to the publication must unique in the repository.

CHAPTER 14

Versioning an embedded data model

This chapter contains the following topics:

1. About versions
2. Accessing versions
3. Working with versions
4. Known limitations on data model versioning

14.1 About versions

You can create *versions* for data models that evolve in parallel. Versions are not to be confused with data model snapshots, which are taken at publication time and kept strictly for historical read-only reference.

14.2 Accessing versions

To see the existing versions of your data model, select 'Manage versions' from the **data model 'Actions'** [p 45] menu of the data model.

The existing versions are represented in a tree format according to their parent-child relationships. Every data model has a root version by default, with the same name as the data model.

14.3 Working with versions

In the workspace, using the down arrow menu next to each version, you can perform the following actions:

Access data model version	Go to the corresponding version of the data model.
Create version	Creates a new version based on the contents of the selected version. The new version is added as a child of the selected version, though its contents bear no relation to those of its parent version after creation.
Set as default version	Sets the selected version as the default version opened when users access the data model.
Export archive	Exports the selected data model version to an archive containing the version's content, including its permissions and information. The exported archive is located in the archives directory, which is accessible to repository administrators. Exporting to an existing archive name will overwrite the existing file. See Archives directory [p 664] for more information.
Import archive	Imports the content of an archive into the selected version. The archive to import must contain a data model with the same name as the data model associated with the version.

A version can be deleted by clicking the **X** button to the right of its entry. A version cannot be deleted if it is linked to a publication or if it has child versions. The root version of a data model also cannot be deleted.

Two versions of the same data model can be compared in the workspace by selecting their checkboxes, then selecting **Actions > Compare selected versions**. The side-by-side comparison shows structural differences between the version of the data model, with the older version on the left and the newer version on the right.

14.4 Known limitations on data model versioning

- It is not possible to merge two versions of a data model.
- The comparison interface does not display updates on fields, only additions and deletions.
- Versioning of data models packaged in modules is not supported.
- Resources packaged in a module that are used by an embedded data model are not versioned when a version is created. That is, only the reference of the resources are saved during the creation of a version, and it is the responsibility of developers to ensure that the content of the referenced resources are compatible with any versions that may be using them.

Dataspaces

CHAPTER 15

Introduction to dataspaces

This chapter contains the following topics:

1. Overview
2. Using the Dataspaces area user interface

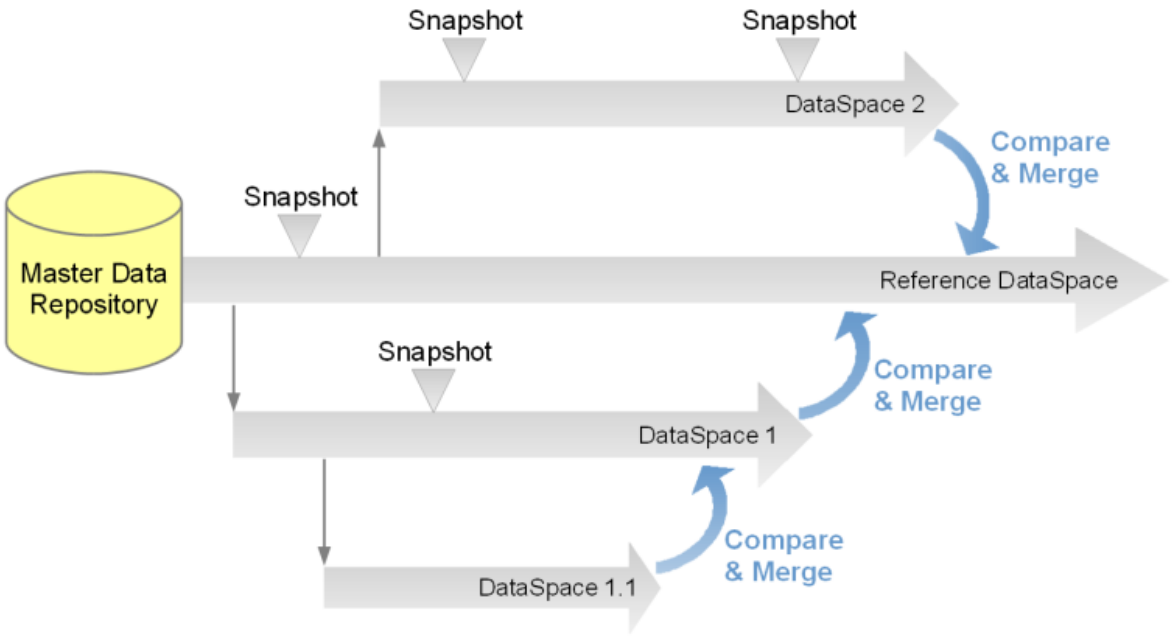
15.1 Overview

What is a dataspace?

The life cycle of data can be complex. It may be necessary to manage a current version of data while working on several concurrent updates that will be integrated in the future, including keeping a trace of various states along the way. In TIBCO EBX®, this is made possible through the use of dataspaces and snapshots.

A dataspace is a container that isolates different versions of datasets and organizes them. A dataspace can be branched by creating a child dataspace, which is automatically initialized with the state of its parent. Thus, modifications can be made in isolation in the child dataspace without impacting its parent or any other dataspaces. Once modifications in a child dataspace are complete, that dataspace can be compared with and merged back into the parent dataspace.

Snapshots, which are static, read-only captures of the state of a dataspace at a given point in time, can be taken for reference purposes. Snapshots can be used to revert the content of a dataspace later, if needed.



Basic concepts related to dataspaces

A basic understanding of the following terms is beneficial when working with dataspaces:

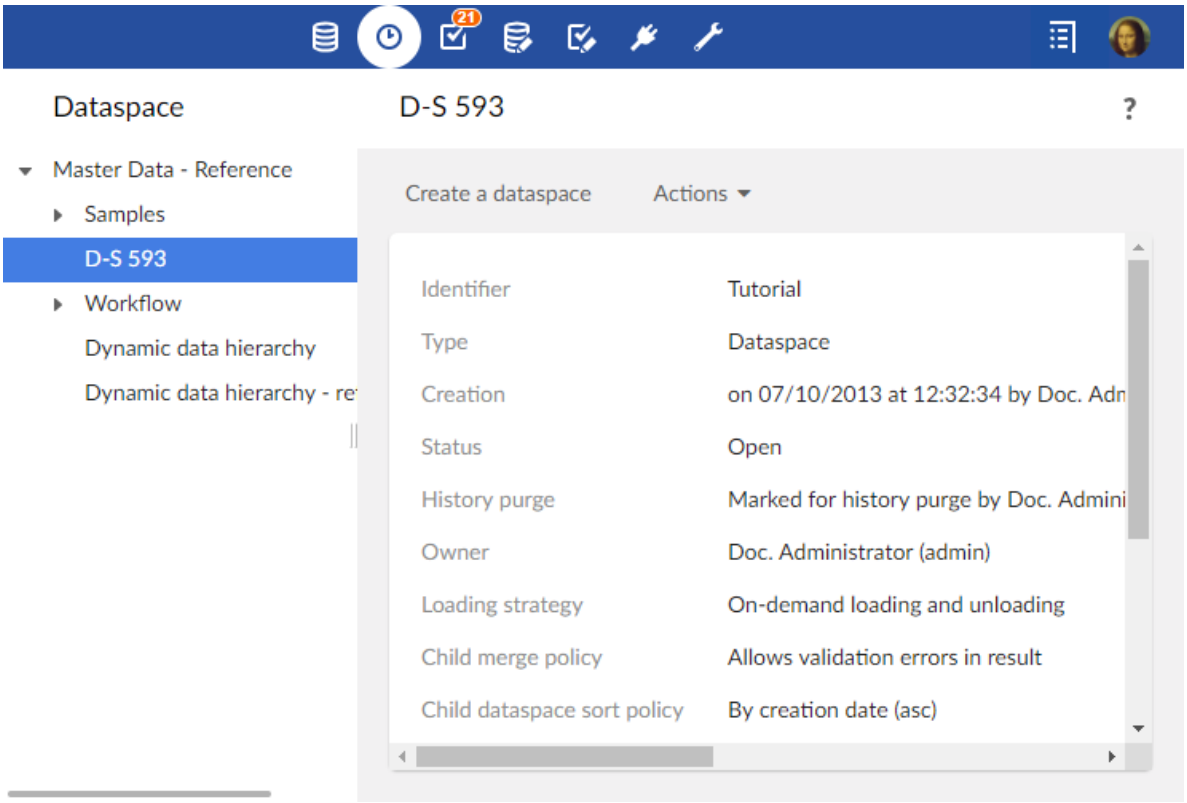
- dataspace [p 38]
- snapshot [p 38]
- dataset [p 36]
- dataspace merge [p 38]
- reference dataspace [p 38]

15.2 Using the Dataspaces area user interface

Dataspaces can be created, accessed and modified in the **Dataspaces** area.

> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective'.

The navigation pane displays all existing dataspaces, while the workspace displays information about the selected dataspace and lists its snapshots.



See also

Creating a dataspace *[p 107]*

Snapshots *[p 117]*

Related concepts *Datasets* *[p 122]*

CHAPTER 16

Creating a dataspace

This chapter contains the following topics:

1. Overview
2. Properties

16.1 Overview

To create a new dataspace, select an existing dataspace on which to base it, then click the **Create a dataspace** button in the workspace.

> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective'.

The new dataspace will be a child dataspace of the one from which it was created. It will be initialized with all the content of the parent at the time of creation, and an initial snapshot will be taken of this state.

Aside from the reference dataspace, which is the root of all dataspaces in the repository, dataspaces are always a child of another dataspace.

16.2 Properties

The following information is required at the creation of a new dataspace:

Identifier	Unique identifier for the dataspace.
Owner	Owner of the dataspace, who is, by default, allowed to modify its information and permissions. The owner does not necessarily have to be the creator of the dataspace.
Label	Localized label and description associated with the dataspace.

CHAPTER 17

Working with existing dataspaces

This chapter contains the following topics:

1. Dataspace information
2. Dataspace permissions
3. Merging a dataspace
4. Comparing a dataspace
5. Validating a dataspace
6. Dataspace archives
7. Closing a dataspace

17.1 Dataspace information

Certain properties associated with a dataspace can be modified by selecting **Actions > Information** from the navigation panel in the Dataspaces area.

Documentation	Localized labels and descriptions associated with the dataspace.
Child merge policy	This merge policy only applies to user-initiated merge processes; it does not apply to programmatic merges, for example, those performed by workflow script tasks. The available merge policies are: • **Allows validation errors in result:** Child dataspaces can be merged regardless of the validation result. This is the default policy. • **Pre-validating merge:** A child dataspace can only be merged if the result would be valid.
Current Owner	Owner of the dataspace, who is, by default, allowed to modify its information and permissions. The owner does not necessarily have to be the creator of the dataspace.
Child dataspace sort policy	Defines the display order of child dataspaces in dataspace trees. If not defined, the policy of the parent dataspace is applied. Default is 'by label'.
Change owner	Whether the current owner of the dataspace is allowed to give ownership to another profile by modifying its 'Current owner' property. If the value is 'Forbidden', only an administrator can modify the owner.
Change permissions	Whether the current owner of the dataspace is allowed to modify its permissions. If the value is 'Forbidden', only an administrator can modify the permissions of the dataspace.

17.2 Dataspace permissions

General permissions

Dataspace id	The dataspace to which the permissions will apply.
Profile selection	The profile to which the rule applies.
Restriction policy	Whether these permissions restrict the permissions assigned to a given user through policies defined for other profiles. See Restriction policy [p 501].
Dataspace access	The global access permission on the dataspace. **Read-only** • Can see the dataspace and its snapshots, as well as child dataspaces, according to their permissions. • Can see the contents of the dataspace depending on their permissions; cannot make modifications. **Write** • Can see the dataspace and its snapshots, as well as child dataspaces, according to their permissions. • Can modify the contents of the dataspace depending on their permissions. **Hidden** • Cannot see the dataspace nor its snapshots directly. • From a child dataspace, the current dataspace can be seen but not selected. • Cannot access the contents of the dataspace. • Cannot perform any actions on the dataspace.

Allowable actions

Users can be allowed to perform the following actions:

Create a child dataspace	Whether the profile can create child dataspaces.
Create a snapshot	Whether the profile can create snapshots from the dataspace.
Initiate merge	Whether the profile can merge the dataspace with its parent.
Export archive	Whether the profile can perform exports.
Import archive	Whether the profile can perform imports.
Close dataspace	Whether the profile can close the dataspace.
Close snapshot	Whether the profile can close snapshots of the dataspace.
Rights on services	Specifies the access permissions for services.
Permissions of child dataspaces when created	Specifies the default access permissions for child dataspaces that are created from the current dataspace.

17.3 **Merging a dataspace**

When the work in a given dataspace is complete, you can perform a one-way merge of the dataspace back into the dataspace from which it was created. The merge process is as follows:

1. Both the parent and child dataspaces are locked to all users, except the user who initiated the merge and administrator users. These locks remain for the duration of the merge operation. When locked, the contents of a dataspace can be read, but they cannot be modified in any way.

   Note: This restriction on the parent dataspace means that, in addition to blocking direct modifications, other child dataspaces cannot be merged until the merge in progress is finished.
2. Changes that were made in the child dataspace since its creation are integrated into its parent dataspace.
3. The child dataspace is closed.
4. The parent dataspace is unlocked.

Initiating a merge

To merge a dataspace into its parent dataspace:

1. Select that dataspace in the navigation pane of the Dataspaces area.
2. In the workspace, select **Merge dataspace** from the **Actions** menu.

Reviewing and accepting changes

After initiating a dataspace merge, you must review the changes that have been made in the child (source) dataspace since its creation, to decide which of those changes to apply to the parent (target) dataspace.

> **Note**
>
> This change set review and acceptance stage is bypassed when performing merges using data services or programmatically. For automated merges, all changes in the child dataspace override the data in the parent dataspace.

The change acceptance process uses a side-by-side comparison interface that recapitulates the changes that require review. Two *change set* columns are obtained by taking the relevant changes from the following dataspace state comparisons:

- The current child dataspace compared to its initial snapshot.
- The parent dataspace compared to the initial snapshot of the child dataspace.

By default, all detected changes are selected to be merged. You may deselect any changes that you want to omit from the merge. You can view the changes relevant to different scopes in your data model by selecting elements in the navigation pane.

In order to detect conflicts, the merge involves the current dataspace, its initial snapshot and the parent dataspace, because data is likely to be modified both in the current dataspace and its parent.

The merge process also handles modifications to permissions on tables in the dataspace. As with other changes, access control changes must be reviewed for inclusion in the merge.

When you have decided which changes to merge for a given scope, you must click the button **Mark difference(s) as reviewed** to indicate that you have reviewed all the changes in that scope. All changes must be reviewed in order to proceed with the merge.

Types of modifications

The merge process considers the following changes as modifications to be reviewed:

- Record and dataset creations
- Any changes to existing data
- Record, dataset, or value deletions
- Any changes to table permissions

Types of conflicts

This review interface also shows conflicts that have been detected. Conflicts may arise when the same scope contains modifications in both the source and target dataspaces.

Conflicts are categorized as follows:

- A record or a dataset creation conflict
- An entity modification conflict
- A record or dataset deletion conflict
- All other conflicts

Finalizing a merge

Once you have reviewed all changes and decided which to include in the merge result, click on the **Merge >>** button in the navigation pane.

Depending on the child merge policy that is configured on the parent dataspace in your merge, the subsequent process may differ. By default, merges are finalized even if the result would contain validation errors. The administrator of the parent dataspace in your merge can set its child merge policy so that merges of its children are only finalized if the result would not contain any validation errors.

If, however, the administrator of the parent dataspace has set its child merge policy to 'Pre-validating merge', a dedicated dataspace is first created to hold the result of the merge. When the result is valid, this dedicated dataspace containing the merge result is automatically merged into the parent dataspace, and no further action is required.

In the case where validation errors are detected in the dedicated merge dataspace, you only have access to the original parent dataspace and the dataspace containing the merge result, named "[merge] < *name of child dataspace* >". The following options are available to you from the **Actions > Merge in progress** menu in the workspace:

- **Cancel**, which abandons the merge and recuperates the child dataspace in its pre-merge state.
- **Continue**, which you can use to reattempt the merge after you have made the necessary corrections to the dedicated merge dataspace.

Setting the child merge policy of a dataspace

As the administrator of a dataspace, you can block the finalization of merges of its child dataspaces through the user interface when the merges would result in a dataspace with validation errors. To do so, select **Actions > Information** from the workspace of the parent dataspace. On the dataspace's information page, set the **Child merge policy** to **Pre-validating merge**. This policy will then be applied to the merges of all child dataspaces into this parent dataspace.

> **Note**
>
> When the merge is performed through a Web Component, the behavior of the child merge policy is the same as described; the policy defined in the parent dataspace is automatically applied when merging a child dataspace. However, this setting is ignored during programmatic merge, which includes script tasks in data workflows.

See also *Child merge policy* *[p 114]*

Abandoning a merge

Merges are performed in the context of a user session, and must be completed in a single operation. If you decide not to proceed with a merge that you have initiated, you can click the **Cancel** button to abandon the operation.

If you navigate to another page after initiating a merge, the merge will be abandoned, but the locks on the parent and child dataspaces will remain until you unlock them in the Dataspaces area.

You may unlock a dataspace by selecting it in the navigation pane, and clicking the **Unlock** button in the workspace. Performing the unlock from the child dataspace unlocks both the child and parent dataspaces. Performing the unlock from the parent dataspace only unlocks the parent dataspace, thus you need to unlock the child dataspace separately.

17.4 Comparing a dataspace

You can compare the contents of a dataspace to those of another dataspace or snapshot in the repository. To perform a comparison, select the dataspace in the navigation pane, then select **Actions > Compare** from the workspace.

The comparison wizard prompts you to select the dataspace or snapshot with which to compare the current dataspace.

For a faster comparison that ignores fields with inherited and computed values, select the filter 'Persisted values only'.

See also *Compare contents* *[p 437]*

17.5 Validating a dataspace

To perform a global validation of the contents of a dataspace, select that dataspace in the navigation panel, then select **Actions > Validate** in the workspace.

> **Note**
>
> This service is only available in the user interface if you have permission to validate every dataset contained in the current dataspace.

17.6 Dataspace archives

The content of a dataspace can be exported to an archive or imported from an archive.

Exporting

To export a dataspace to an archive, select that dataspace in the navigation panel, then select **Actions > Export** in the workspace. Once exported, the archive file is saved to the file system of the server, where only an administrator can retrieve the file.

> **Note**
>
> See Archives directory [p 664] in the Administration Guide for more information.

In order to export an archive, the following information must be specified:

Name of the archive to create	The name of the exported archive.
Export policy	Required. The default export policy is 'The whole content of the dataspace', which exports all selected data to the archive. It may be useful to export only the differences between the dataspace and its initial snapshot using a change set. There are two different export options that include a change set: 'The updates with their whole content' and 'The updates only'. The first option exports all current data and a change set containing differences between the current state and the initial snapshot. The second option only exports the change set. Both options lead to a comparison page, where you can select the differences to include in this change set. Differences are detected at the table level.
Datasets to export	The datasets to export from this dataspace. For each dataset, you can export its data values, permissions, and/or information.

Importing

To import content into a dataspace from an archive, select that dataspace in the navigation panel, then select **Actions > Import** in the workspace.

If the selected archive does not include a change set, the current state of the dataspace will be replaced with the content of the archive.

If the selected archive includes the whole content as well as a change set, you can choose to apply the change set in order to merge the change set differences with the current state. Applying the change set leads to a comparison screen, where you can then select the change set differences to merge.

If the selected archive only includes a change set, you can select the change set differences to merge on a comparison screen.

17.7 Closing a dataspace

If a dataspace is no longer needed, it can be closed. Once it is closed, a dataspace no longer appears in the **Dataspaces** area of the user interface, nor can it be accessed.

An administrator can reopen a closed dataspace as long as it has not been cleaned from the repository.

To close a dataspace, select **Actions > Close this dataspace** .

See also *Closing unused dataspaces and snapshots [p 667]*

CHAPTER 18

Snapshots

This chapter contains the following topics:

1. Overview of snapshots
2. Creating a snapshot
3. Viewing snapshot contents
4. Snapshot information
5. Comparing a snapshot
6. Validating a snapshot
7. Export
8. Closing a snapshot

18.1 Overview of snapshots

A snapshot is a read-only copy of a dataspace. Snapshots exist as a record of the state and contents of a dataspace at a given point in time.

See also *Snapshot [p 38]*

18.2 Creating a snapshot

A snapshot can be created from a dataspace by selecting that dataspace in the navigation pane of the Dataspaces area, then selecting **Actions > Create a Snapshot** in the workspace.

The following information is required:

Identifier	Unique identifier for the snapshot.
Label	Localized labels and descriptions associated with the snapshot.

18.3 Viewing snapshot contents

To view the contents of a snapshot, select the snapshot, then select **Actions > View datasets** from the workspace.

18.4 Snapshot information

You can modify the information associated with a snapshot by selecting **Actions > Information**.

Documentation	Localized labels and descriptions associated with the snapshot.
Current Owner	Owner of the snapshot, who is, by default, allowed to modify its information and permissions. The owner does not necessarily have to be the creator of the snapshot.
Change owner	Whether the current owner of the snapshot is allowed to give ownership to another profile by modifying its 'Current owner' property. If the value is 'Forbidden', only an administrator can modify the owner.

18.5 Comparing a snapshot

You can compare the contents of a snapshot to those of another snapshot or dataspace in the repository. To perform a comparison, select the snapshot, then select **Actions > Compare** from the workspace.

The comparison wizard prompts you to select the dataspace or snapshot with which to compare the current snapshot.

For a faster comparison that ignores fields with inherited and computed values, select the filter 'Persisted values only'.

See also *Compare contents* *[p 437]*

18.6 Validating a snapshot

To perform a global validation of the contents of a snapshot, select **Actions > Validate** in the workspace.

> **Note**
>
> In order to use this service, you must have permission to validate every dataset contained in the snapshot.

18.7 Export

To export a snapshot to an archive, open that snapshot, then select **Actions > Export** in the workspace. Once exported, only an administrator can retrieve the archive.

> **Note**
>
> See Archives directory [p 664] in the Administration Guide for more information.

In order to export an archive, the following information must be specified:

Name of the archive to create	The name of the exported archive.
Datasets to export	The datasets to export from this snapshot. For each dataset, you can choose whether to export its data values, permissions, and information.

18.8 Closing a snapshot

If a snapshot is no longer needed, it can be closed. Once it is closed, a snapshot no longer appears under its associated dataspace in the Dataspaces area, nor can it be accessed.

An administrator can reopen a closed dataspace as long as it has not been cleaned from the repository.

To close a snapshot, select **Actions > Close this snapshot**.

See also *Closing unused dataspaces and snapshots [p 667]*

Datasets

CHAPTER 19

Introduction to datasets

This chapter contains the following topics:

1. Overview
2. Using the Data user interface

19.1 Overview

What is a dataset?

A dataset is a container for data that is based on the structural definition provided by its underlying data model. When a data model has been published, it is possible to create datasets based on its definition. If that data model is later modified and republished, all its associated datasets are automatically updated to match.

In a dataset, you can consult actual data values and work with them. The views applied to tables allow representing data in a way that is most suitable to the nature of the data and how it needs to be accessed. Searches and filters can also be used to narrow down and find data.

Different permissions can also be accorded to different roles to control access at the dataset level. Thus, using customized permissions, it would be possible to allow certain users to view and modify a piece of data, while hiding it from others.

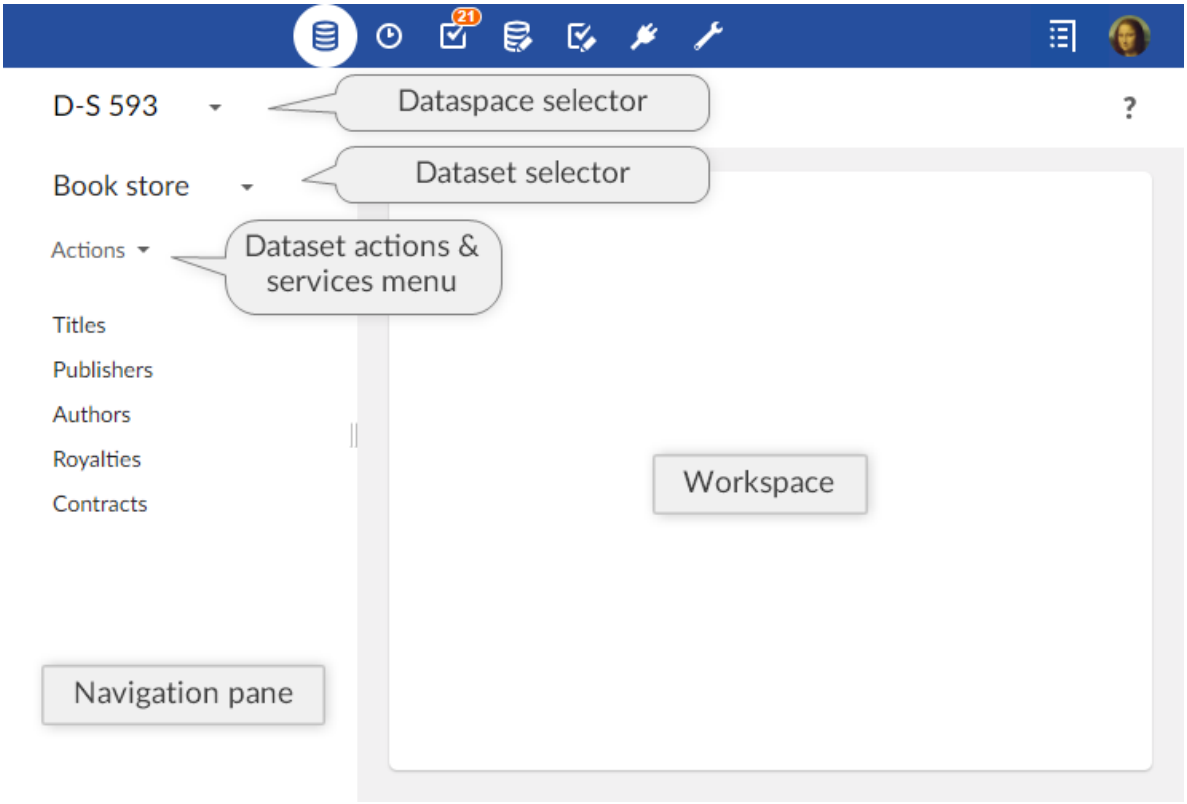
Basic concepts related to datasets

A basic understanding of the following terms is beneficial when working with datasets:

- dataspace [p 38]
- dataset [p 36]
- record [p 36]
- field [p 35]
- primary key [p 35]
- foreign key [p 35]
- table (in dataset) [p 36]
- group [p 35]

19.2 Using the Data user interface

Datasets can be created, accessed and modified in the **Data** area using the Advanced perspective [p 21] or from a specifically configured perspective. Only authorized users can access these interfaces.



Select or create a dataset using the 'Select dataset' menu in the navigation pane. The data structure of the dataset is then displayed in the navigation pane, while record forms and table views are displayed in the workspace.

When viewing a table of the dataset in the workspace, the button displays searches and filters that can be applied to narrow down the records that are displayed.

Operations at the dataset level are located in the **Actions** menu in the navigation pane (services are available at the bottom of the list).

See also

Creating a dataset *[p 125]*

Quick Search *[p 128]*

Working with records in the user interface *[p 149]*

Inheritance *[p 37]*

Related concepts

Data model *[p 44]*

Dataspace *[p 104]*

CHAPTER 20

Creating a dataset

This chapter contains the following topics:

1. Creating a root dataset
2. Creating an inheriting child dataset

20.1 Creating a root dataset

To create a new root dataset, that is, one that does not inherit from a parent dataset, select the '**Select dataset** [p 123]' menu in the navigation pane, click the '**Create a dataset**' button in the pop-up, and follow through the wizard.

> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective' or from a specifically configured perspective.

The wizard allows you to select one of three data model packaging modes on which to base the new dataset: packaged, embedded, or external.

- A *packaged data model* is a data model that is located within a module, which is a web application.
- An *embedded data model* is a data model that is managed entirely within the TIBCO EBX® repository.

After locating the data model on which to base your dataset, you must provide a unique name, without spaces or special characters. Optionally, you may provide localized labels for the dataset, which will be displayed to users in the user interface depending on their language preferences.

Attention

Table contents are not copied when duplicating a dataset.

20.2 Creating an inheriting child dataset

The inheritance mechanism allows datasets to have parent-child relationships, through which default values are inherited from ancestors by descendants. In order to be able to create child datasets, dataset inheritance must be enabled in the underlying data model.

To create a child dataset, select the '**Select dataset** [p 123]' ▼ menu in the navigation pane, then click the + button next to the desired parent dataset.

As the dataset will automatically be based on the same data model as the parent dataset, the only information that you need to provide is a unique name, and optionally, localized labels.

See also *Dataset inheritance* *[p 169]*

CHAPTER 21

Viewing data

TIBCO EBX® offers different ways to list records. This chapter presents how to sort, search, and display records in varying ways, and according to different user profiles, through to the concept of 'Views'.

This chapter contains the following topics:

1. 'View' menu
2. Sorting data
3. Quick Search
4. Searching and filtering data
5. Views
6. Views management
7. Grid edit
8. History

21.1 'View' menu

The 'View' drop-down menu allows accessing all available views and management features.

Views are managed directly in the 'View' menu toolbar, available on each listed view: 'View' menu toolbar [p 146].

Views can also be grouped. An administrator has to beforehand define groups in 'Views configuration' under the 'Groups of views' table. The end-user can then set a view as belonging to a group, through the field 'View group' upon creation or modification of the view. See 'View description' [p 134] for more information.

21.2 Sorting data

Sort criteria control the order in which records are presented.

Use the 'Select and Sort' button at the top left of the table to define specific sorting criteria.

There are two types of sorting:

- sorting by relevance, used in the quick search [p 128],
- sorting by column.

Sorting by column

The 'Sorting criteria' dialog box offers:

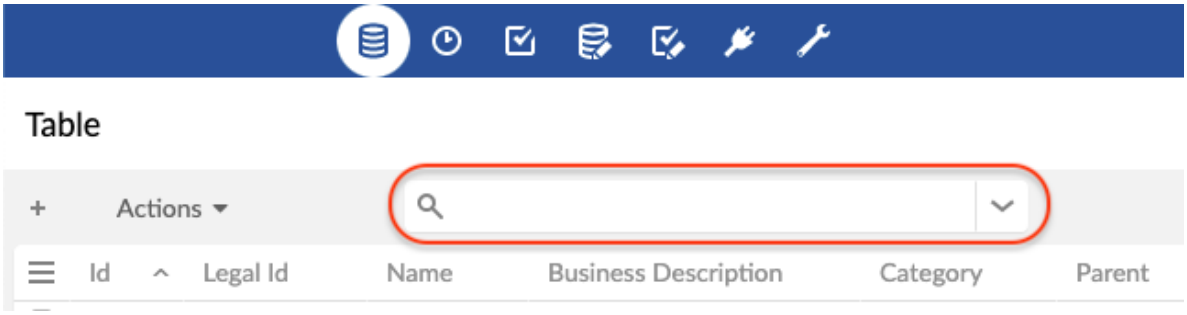
- on the left, the list of sorted columns,
- on the right, the list of unsorted columns.

Use the ← → arrow buttons to toggle columns from one list to another.

Use the ↑ ↓ arrow buttons to change the priority order of the column.

To change the sort order of a column, use the 'ASC' (ascending) or 'DESC' (descending) button that appears on the mouse-over.

21.3 Quick Search



The quick search is used to easily find a result in a tabular, a hierarchical, or a tile view.

It does not differentiate between upper and lower case. It allows you to search for several terms at once (separated by spaces). By default, the records found are sorted by relevance. This search includes all searchable columns, even those that are not visible. When the quick search is not possible for a column, it is indicated by an icon when hovering over the column header.

See also *Search* *[p 511]*

Special characters

The quick search offers special characters to refine your search on strings. Please note that columns that are not string-based do not support this syntax, and certain search strategies such as 'Code' does not support it either.

+... required

Makes it required that the word be present in the result. Prevents the search engine from excluding this word from the search.

> **Note**
>
> Works with **+"**phrase**"** or **+(**group**)**.

Example:

```
+flute +bach
```

↳ Finds results with required *flute* and required *bach*. Results with only *flute* or only *bach* will be ignored.

-... not

Excludes the word from the result.

> **Note**
>
> Works with **-"**phrase**"** or **-(**group**)**.

Example:

```
bach -flute
```

↳ Find results with *bach*, but without *flute*.

...~fuzzy

Specifies that the search can change 2 characters of the word to find it.

To allow for a single character to change, use **~1**.

> **Note**
>
> Works with **"**phrase**"~**, to change *2 words* in the phrase.

Example:

```
handel~
```

↳ Find results with a word that is *handel*, with 1 or 2 letters differing.

?single joker

Replaces an unknown character.

Example:

```
?uttle
```

	↳ Finds results with a word starting with any character, and ending with *uttle*.
***extended joker**	Replaces several unknown characters. *Example:* `rachmanino*` ↳ Find results with a word beginning with *rachmanino*.
"..."*phrase*	Find the exact match of the phrase. **Note** Can be surrounded by **+-~**. *Example:* `"Johann Sebastian Bach"` ↳ Find results containing exactly *Johann Sebastian Bach*.
(...)*group*	Allows grouping words to apply a special character **+** or **-** to them. **Note** You can make groups of groups. *Example:* `bach +(flute piano)` ↳ Find the results with possibly *bach*, and necessarily *flute* or *piano*.

Note

These special characters can also be used in the documentation search engine.

Quick search result highlighting

Quick search results are highlighted to improve the user experience by visually identifying the parts of the data that match the search query, making it easier to locate relevant records in the result. Highlighting is enabled only in the tabular view and does not apply to hidden columns.

Known limitations

When the search matches a foreign key displayed using computed label, the highlight may not appear as expected. Instead, the displayed label may be replaced by the raw field value that matched the query, since the highlighting mechanism operates on raw values rather on labels.

Known limitations

The highlight does not appear for fields using custom widgets.

21.4 Searching and filtering data

The search pane is hidden by default and accessible via the ⌄ icon located to the right of the quick search in the toolbar of the tabular, the hierarchical or the tile view.

The quick search and the criteria lines combine to narrow the search (restricting the result to fewer and fewer records).

It is possible to deactivate a criteria line by unchecking it. The deactivated criteria are not kept during a save.

The trashcan button 🗑 at the end of the line of each criterion permanently deletes the criterion.

To save the filter applied to a search, use the 'Save' button. Saving takes into account the quick search and all active criteria.

To recall a saved filter, use the 'Load' button. Loading replaces the quick search and the whole criteria panel. Click on the 'Apply' button to start the new search.

When a view is applied, it ensures that it is displayed according to its configuration. All existing criteria in the search panel are therefore removed. The view can contain a set of search criteria, which are applied together with the at the same time as the view.

Some operators (such as 'text-search') allow to use Lucene regular expressions. See technical specifications of Lucene's regex pattern for more information.

Search on a field

All searchable fields are available, except those of type `osd:locale` which are not defined as enumerations, and those of type `osd:resource`.

Validation filter

In field selection, the validation criteria display the records as of the last validation performed.

> **Note**
>
> This filtering only applies to records of the table that have been validated at least once by selecting *Actions* > *Validate* at the table level from the workspace, or at the dataset level from the navigation pane.

To filter on the validation severity level (independent from validation Message), use the 'Severity' validation criterion. Available levels are: 'Errors', 'Warnings' and 'Information'.

To filter on the validation message (independent from validation Severity level), use the 'Message' validation criterion.

Create complex filters

Use multiple criteria grouped by logical operator to create complex filters. The groups display in a tree view that utilizes a variety of features to streamline the construction of even the most complex filters.

For example, take a record where Bob and Smith are values passed to the filter in the Name and Surname fields, respectively. If the **contains** condition is used for both fields, the logical operators impact record filtering as follows:

- AND: restricts the search, as a record must meet *all* of the criteria in this group to display. Only a record that has Bob Smith displays in the results.
- OR: broadens the search, as a record only needs to meet *one* of the criteria in this group to display. Any record displays where the Name field includes Bob, or the Surname field includes Smith. For example, results might include Jon Smith and Bob Jones.
- NOT: inverts a group or criterion so that the opposite conditions from those specified must be met to display the record.
  - NOT AND: All records that are *not* Bob Smith are returned.
  - NOT OR: Only records that do not have Bob *or* Smith are returned. For example, Peter Jones and Mary Stewart.
  - NOT applied to a single criterion: Reverses the condition for that criterion *only*. For example, if the group's condition is AND, and NOT was applied to the Surname field, the Bob Smith record would be excluded from results as only records that *do not* contain Smith in the Surname field are returned.

The following table describes actions available when working with groups:

Select an operator	Toggle between the AND and OR operators.
Mouse hover:	On operators allows you to select the following: • NOT to invert the group. •  to duplicate the group. •  to ungroup the criteria. On individual criterion hover to select NOT and invert the condition applied to that criterion.
Drag and drop	Drag lines connected to criteria: • Up/down to reorder within the containing group. • To another criterion in the same group to create a new subgroup. • To a different operator to move the criterion to that group.
Group criteria button	Use this button to enable group edit. Select the checkboxes next to criteria you want to group. Note that action moves the selected criteria to the new group.

Custom table searches

For backward compatibility, the feature for custom searching and filtering records is still operational and accessible via the icon  in the workspace. The icon and feature are only available when at least one custom filter exists.

Additional custom filters can be specified for each table in the data model.

See also *Customizing table filter [p 1026]*

Technical limitation

Optimized text search filters cannot be combined using the OR operator with non-optimized filters, or filters that involve a different table.

Typical examples of this situation could be a text search with an OR filter on:

- An inherited field.
- A computed value without a local dependency.
- A field from a different table (e.g. a linked field).

This technical limitation displays as "The request cannot be resolved since a "text search" operator is inside an "OR" group" (technical limitation).

21.5 Views

It is possible to customize the display of tables in EBX® according to the target user. There are three types of views: tabular [p 135], hierarchical [p 135] and tiles [p 142].

A view can be created by selecting *View > Create a new view* in the workspace. To apply a view, select it in *View > name of the view*.

Three types of views can be created:

- 'Simple tabular view': A table view to sort and filter the displayed records.
- 'Hierarchical view': A tree view that links data in different tables based on their relationships.
- 'Tile view': A view which displays the records in the form of cards that are positioned in rows and columns. This view allows to show information in an eye catching form, sort, and filter the records displayed.

View description

When creating or updating a view, the first page allows specifying general information related to the view.

Documentation	Localized label and description associated with the view.
Owner	Name of the owner of the view. This user can manage and modify it. (Only available for administrators and dataset owners)
Share with	Other profiles allowed to use this view from the 'View' menu. **Note** Requires a permission, see Views permissions [p 686].
View mode	Simple tabular view, hierarchical view or tile view.
View group	Group to which this view belongs (if any).

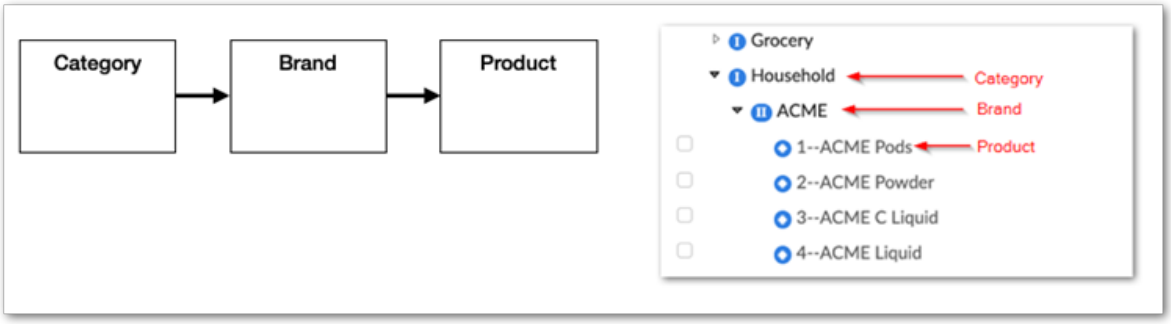
Simple tabular views

Simple tabular views offer the possibility to define criteria to filter records and also to select the columns that will be displayed in the table.

Displayed columns	Specifies the columns that will be displayed in the table.
Sorted columns	Specifies the sort order of records in the table. See Sorting data [p 127].
Filter	Defines filters for the records to be displayed in this table view. See Criteria editor [p 509].
Initial request	When set to manual, users of this view will have to apply a filter or a sort criteria before they can view data.
Pagination limit	Forces a limit to the number of visible records.
Grid edit	If enabled, users of this view can switch to grid edit, so that they can edit records directly from the tabular view.
Disable create and duplicate	If yes, users of this view cannot create nor duplicate records from the grid edit.
Toolbar on top of table	Defines the toolbar to use on top of this table view.
Toolbar on table row	Defines the toolbar to use on each row of this table view.

Hierarchical views

A hierarchy view displays your data in a tree-like representation. Use hierarchy views to visualize different relationships between data, or to group common values. To create a hierarchy view, you define a dimension that determines what data will be included and how it displays. Consider the following example and image where the dimension is Category --> Brand --> Product and displays Products by Brand by Category.

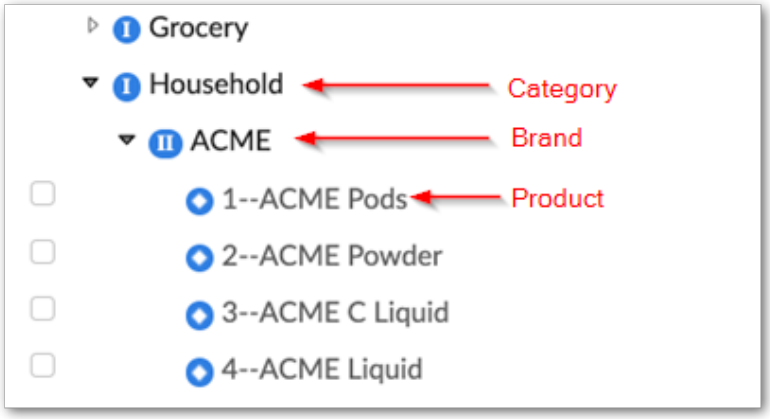


Hierarchy dimensions and members

Each hierarchy is defined by its dimension. The dimension is subdivided into dimension levels. Members are hierarchy nodes and represent a record, or a field value. A member can be a parent of other members, or a leaf node with no children. The table from which you create the hierarchy view is called the target table. Members of the target table are lowest-level leaves, except in the case of a recursive hierarchy.

In the following image:

- Grocery and Household are members of the Category dimension level. This is the top dimension level.
- ACME is a member of the Brand dimension level and is at an intermediate level. A dimension can have many intermediate levels.
- The individual products listed at the Product dimension level are leaf members from the target table. This is the bottom dimension level.



Creating a hierarchy view

This topic provides high-level instructions for creating a hierarchy view. See the following topics for additional details about creating views of specific hierarchy types:

- Ragged hierarchies [p 137]
- Group by hierarchies [p 138]
- Time-based hierarchy dimensions [p 139]

To create a hierarchy view:

1. From the target table's **Actions** menu, select **Create a new view**.
2. After specifying the following information, click **Next**:
   - Add any view-related documentation such as a label and description.
   - Set access permissions.
   - For the **View mode** property, select **Hierarchical view**.
3. Use the **Dimension** box to define the hierarchy by working backwards through the dimension levels:
   1. Bottom level: The members of this level are pre-determined based on the target table.

2. Intermediate levels: Start with the attribute that corresponds to the next highest dimension level and continue to expand until you navigate to the highest level.
3. Top level: Select the attribute corresponding to the top dimension level.

4. Review the required options and after making desired changes, click **Next**.
5. Specify labels, filters, and sort options for the members of each dimension level:
   - Navigate between dimension levels by selecting them in the breadcrumb at the top of the screen.
   - Make adjustments to labels, filters, and sort strategies as desired.
6. Click **Save and quit** to display the hierarchy view.

Ragged hierarchies

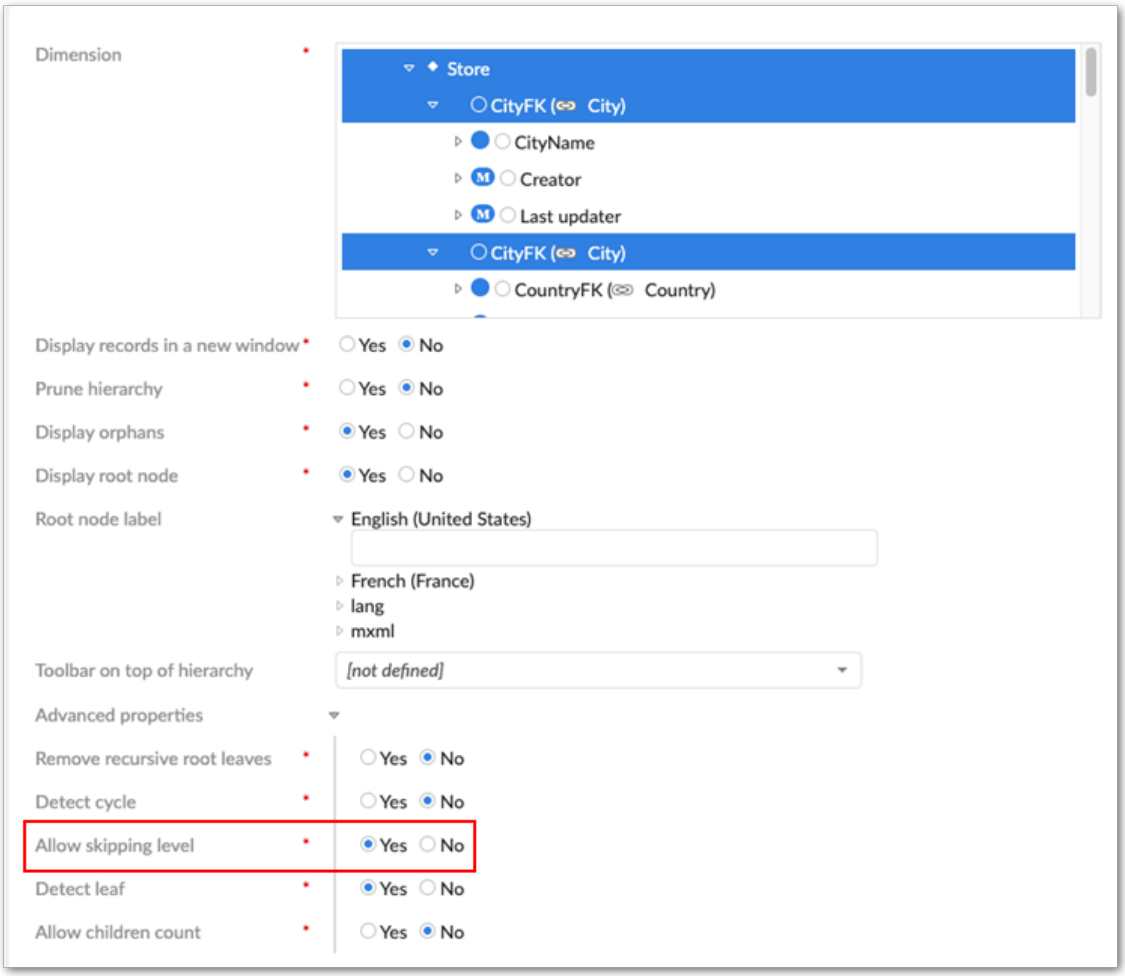
A ragged hierarchy includes members where their parent skips one level. An example of this type of hierarchy is shown by using the Country, State, and City dimension. Some countries do not have states, but they all have cities. In the case of the latter, the city members directly roll up to the Country dimension level.

> **Attention**
> A ragged hierarchy only supports skipping one level.

To display ragged hierarchies in EBX®:

- In the data model, you must define all foreign key relationships from the target table. Using the example above, the City table must hold a foreign key to the State and Country tables. Of course, for cities with no state, this field is left undefined.

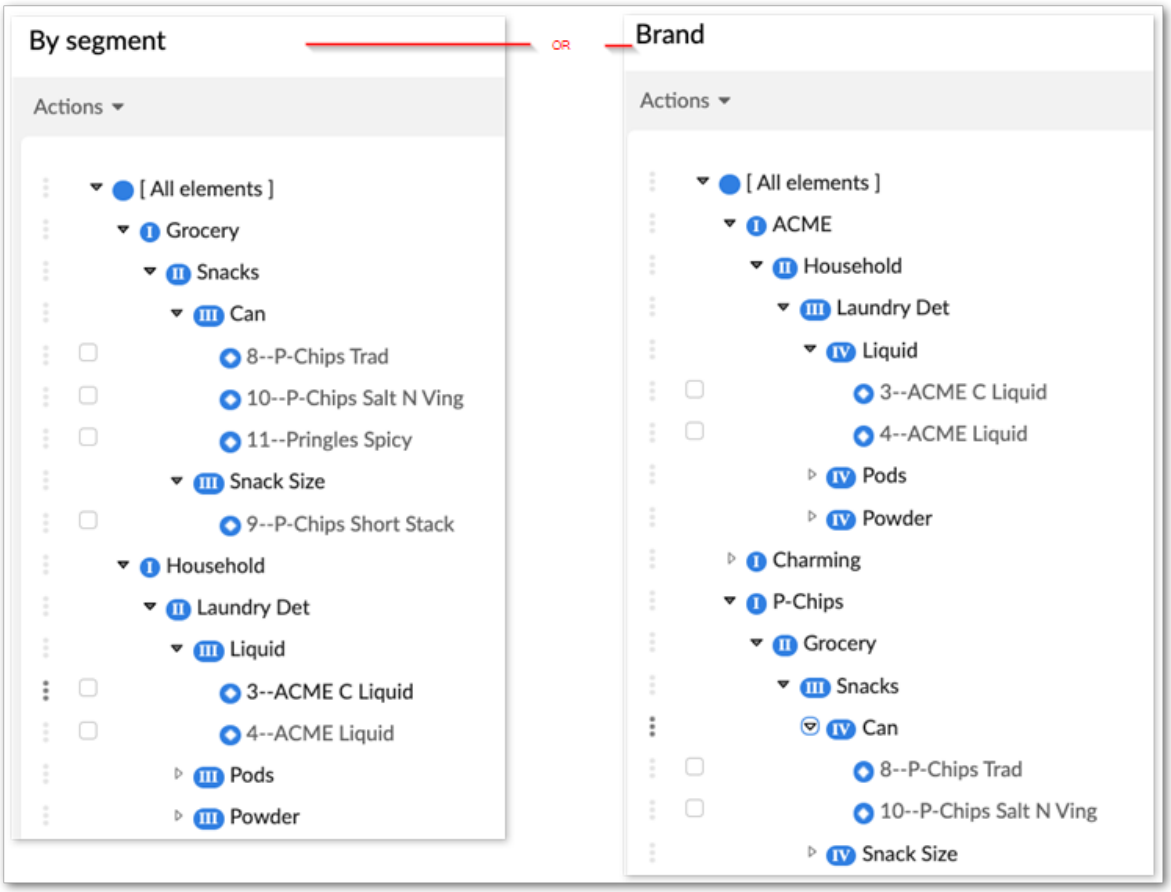
- When configuring your hierarchy view, enable the **Allow skipping level** option under **Advanced properties**. This property is highlighted in the image below:



Group by hierarchies

The dimension in a Group By hierarchy is defined using columns located in the same table. Allowed column data types are foreign keys or the following enumerable types: String, Integer, Enumeration, and Boolean. This provides a powerful tool for defining hierarchy views that group common values together. A group by hierarchy also provides flexibility when setting the order for dimension levels. For example, a Product table might include the following columns: Brand, Segment, Type, and

Packaging. Use these columns in any order to show products by: Segment --> Type --> Packaging, or Brand --> Segment --> Type --> Packaging.



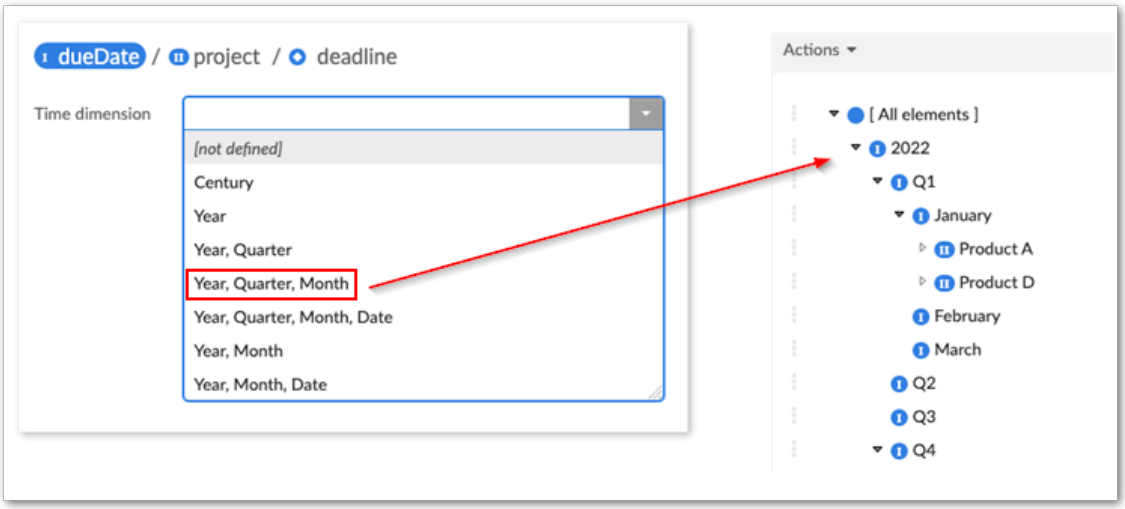
Time-based hierarchy dimensions

Use fields with a Date data type to define a time-based hierarchy dimension. When configuring the view, choose from several predefined display options. Some options subdivide the time into smaller units. For example, you can specify that a project's Due Date field is used to display all project deadlines by month in Q1 of 2022.

To use a time-based dimension in a hierarchy view:

1. Create a new hierarchy view configuration:
    1. When viewing a table, open the **View** menu and select **Create a new view**.
    2. Provide a name and optionally specify share options.
    3. For the **View mode**, select **Hierarchical view** and click **Next**.
2. Ensure the field used for the hierarchy's root dimension is of the Date data type and click **Next**.

3. Click on the Date field at the top of the screen in the hierarchy structure breadcrumb and use the **Time dimension** menu to specify how you want the dimension to display.



4. Save and view the hierarchy.

Hierarchical view configuration options

This form allows configuring the hierarchical view options.

Display records in a new window	If 'Yes', a new window will be opened with the record. Otherwise, it will be displayed in a new page of the same window.
Prune hierarchy	If 'Yes', hierarchy nodes that have no children and do not belong to the target table will not be displayed.
Display orphans	If 'Yes', hierarchy nodes without a parent will be displayed.
Display root node	If 'No', the root node of the hierarchy will not be displayed in the view.
Root node label	Localized label of the hierarchy root node.
Toolbar on top of hierarchy	Allows setting the toolbar on top of the hierarchy.
Detect cycle	Allow cycle detection and display in a recursive case, the oldest node record will be chosen as the cycle root. Limitation: does not work in search or pruned mode.
Detect leaf	Allows detecting whether the member is a leaf or not. The leaf detection is very costly for large volumes of data. Thus, it is recommended to disable this option when the query response is delayed to display the hierarchy view. This property is always disabled for orphans' parent members.

Labels

For each dimension level that references another table, it is possible to define localized labels for the corresponding nodes in the hierarchy. The fields from which to derive labels can be selected using the built-in wizard.

Filter

The criteria editor allows creating a record filter for the view.

See also *Criteria editor* *[p 509]*

Sort strategy

For each dimension level, it is possible to choose one of the following sort strategies:

Default	Nodes are sorted by label in alphabetical order
Sort by columns	Nodes are sorted by selected column(s). The direction (ascending/descending) can be chosen for each column.
Sort by ordering field	Nodes are sorted by a hidden numeric field, which allows users to dynamically change the order of sibling nodes in the hierarchy view. This strategy is available only if there is at least one 'Hidden' numeric field in the table. In order to enable this option, you must designate an eligible ordering field defined in the table on which the hierarchical view is applied. An ordering field must have the 'Integer' data type and have a 'Hidden' default view mode in its advanced properties in the data model definition. Except when the ordering field is in 'read-only' mode or when the hierarchy is filtered, any field can be repositioned. **Attention** Do not designate a field that is meant to contain data as an ordering node, as the data will be overwritten by the hierarchical view.

Actions on hierarchy nodes

Each node in a hierarchical view has a menu ▾ containing contextual actions.

Leaf nodes can be dissociated from their parent record, using 'Detach from parent'. The record then becomes an orphan node in the tree, organized under a container "unset" node.

Leaf nodes can also change parent nodes, using 'Attach to another parent'. If, according to the data model, a node can have relationships to multiple parents, the node will be both under the current parent and added under the other parent node. Otherwise, the leaf node will be moved under the other parent node.

Quick search on hierarchy nodes

Quick search on hierarchy nodes is similar to the tabular view, except that it only applies to the records of the target table's nodes and on the fields defined for the node's label.

Below the Quick search tool there is a **Quick search on all columns** checkbox, which is unchecked by default. If you want it to default to checked, go to **Advanced properties** and enable the corresponding option. Please note that this option is disabled by default for performance reasons.

Tile views

This view displays records as tiles arranged in a grid. Each tile and can include the following information: Title (required), Subtitle, Image, and description. For the titles and description you can:

select fields to automatically populate the values, manually specify the text that displays, or use a combination of both methods. Additionally, you can define sort and filter criteria that applies to the

view. Toolbars can also be added. The following table lists and defines the properties available when creating a tile view:

Title	Defines the tile title displayed on first line.
Subtitle	Defines the subtitle displayed on second line.
Image	Defines the tile image.
Image > Default image URL	Defines the fallback image URL.
Image > URL	Defines the image URL.
Image > Height (px)	Sets the height of the image slot in pixels (between 10 and 250). The slot width will always be 180 pixels. If the dimensions of the image exceed the size of the intended slot, it will be adjusted according to the cropping rules defined by the view.
Image > Cropping	Defines the cropping of the image. If "Cover", the image will be scaled up or down so that occupy all the available space. In this case, the image might be truncated. If "Scale down", the image will be scaled down only if it overflows the available space. Whatever the value, the image will always be centered.
Tile description	Defines the tile text.
Tile description > Number of displayed lines	Defines the number of lines (between 3 and 6).
Tile description > Line wrap	If "Yes", each row is displayed entirely (with line wrap) up to the defined limit of the number of lines. The excess is then truncated. If "No", each line is truncated.
Tile description > Content	Defines the tile text content.
Sort criteria	Specifies the sort order of records in the view. See Sorting data [p 127].
Filter	Defines filters for the records to be displayed in this tile view. See Criteria editor [p 509].
Toolbar on top of table	Defines the toolbar to use on top of this table view.

Tile toolbar	Defines the toolbar to use on each item of this tile view.

View sharing

Users having the 'Share views' permission on a view are able to define which users can display this view from their 'View' menu.

To do so, simply add profiles to the 'Share with' field of the view's configuration screen.

View publication

Users having the 'Publish views' permission can publish views present in their 'View' menu.

A published view is then available to all users via Web components, workflow user tasks, data services and perspectives. To publish a view, go to the 'View menu', click on the Edit button displayed on the mouseover of a listed view and add a 'Publication name' to the view.

21.6 **Views management**

Manage recommended views

When a user logs in with no view specified, their recommended view (if any) is applied. Otherwise, the default view is applied. The 'Manage recommended views' action allows defining assignment rules of recommended views depending on users and roles.

Available actions on recommended views are: change order of assignment rules, add a rule, edit existing rule, delete existing rule.

Thus, for a given user, the recommended views are evaluated according to the user's profile: the applied rule will be the first that matches the user's profile.

> **Note**
>
> The 'Manage recommended view' feature is only available to dataset owners.

'View' menu toolbar

The 'View' menu toolbar offers the following actions:

Edit	Click on the 'Edit' button of the targeted view's toolbar to access the editable form.
Duplicate	Click on the 'Duplicate' button of the targeted view's toolbar to duplicate the view. The new view creation form pre-populates the field values from the view being duplicated.
Delete	Click on the 'Delete' button of the targeted view's toolbar to delete the view.
Define this view as my favorite	Click on the 'Define this view as my favorite' button of the targeted view's toolbar. The favorite view will automatically be applied when accessing the table. Click a second time on the button to remove the view as the user's favorite view.

21.7 Grid edit

The grid edit feature allows modifying data in a table view. This feature can be accessed by clicking on the  button.

Accessing the grid edit from a table view requires that the feature be previously activated in the view configuration.

See also *Grid edit [p 135]*

Copy/paste

The copy/paste of one or more cells into another one in the same table can be done through the *Edit* menu. It is also possible to use the associated keyboard shortcuts *Ctrl+C* and *Ctrl+V*.

This system does not use the operating system clipboard, but an internal mechanism. As a consequence, copying and pasting a cell in an external file will not work. Conversely, pasting a value into a table cell won't work either.

All simple type fields using built-in widgets are supported.

21.8 History

The history feature allows tracking changes on master data.

The history feature must have been previously enabled at the data model level. See Advanced properties for tables [p 69] for more information.

To view the history of a dataset, select *Actions* > *History* in the navigation pane.

To view the history of a table or of a selection of records, select *Actions* > *View history* in the workspace.

Several history modes exist, which allow viewing the history according to different perspectives:

History in current dataspace	The table history view displays operations on the current branch. This is the default mode.
History in current dataspace and ancestors	The table history view displays operations on the current branch and on all its ancestors.
History in current dataspace and merged children	The table history view displays operations on the current branch and on all its merged children.
History in all dataspaces	The table history view displays operations on the whole branch hierarchy.

In the history view, use the **VIEW** menu in order to switch to another history mode.

See also *History* *[p 461]*

CHAPTER 22

Editing data

This chapter contains the following topics:

1. Working with records in the user interface
2. Importing and exporting data
3. Restore from history

22.1 Working with records in the user interface

Record editing takes place in the workspace portion of the user interface.

> **Note**
>
> This action is available only to authorized users in the 'Advanced perspective' or from a specifically configured perspective.

Creating a record

In a tabular or a tile view, click the **+** button located above the table.

In a hierarchical view, select 'Create a record' from the menu of the parent node under which to create the new record.

Next, enter the field values in the new record creation form. Mandatory fields are indicated by asterisks.

Updating an existing record

Double-click the record to update, then edit the field values in the record form.

To discard any changes in progress and restore the fields to their values before editing, click the **Revert** button.

Duplicating a record

To duplicate a selected record, select **Actions** > **Duplicate**.

A new record creation form pre-populates the field values from the record being duplicated. The primary key must be given a unique value, unless it is automatically generated (as is the case for auto-incremented fields).

Deleting records

To delete one or more selected records, select **Actions > Delete**.

Comparing two records

To compare two selected records, select **Actions > Compare**.

> **Note**
>
> The content of complex terminal nodes, such as aggregated lists and user defined attributes, are excluded from the comparison process. That is, the compare service ignores any differences between the values of the complex terminal nodes in the records.

Using the AI Assistant

The AI Assistant leverages Generative AI to help completing tasks by generating suggestions based on record data and user input.

To run the AI Assistant:

- At least one function must be defined for the current table in the data model and it must be published.
- You must have permission to edit the record and all of the input and output fields contained in at least one function.

To use the AI Assistant:

1. Select a record in the table.
2. From the **Actions** menu, select **AI Assistant**.
3. Run an assistance function:
   - Select the tab for the required assistance function.
   - Edit the input fields as needed.
   - Select **Execute** to generate a suggestion.
4. Review and apply results:
   - Review the generated values in the output fields.
   - Optionally, modify the values.
   - To keep the generated values, save the record. Note that any modification done to input fields will also be saved.
   - To discard the generated values, discard the output field changes before saving.

> **Note**
>
> When the same field is configured as both an input field and an output field:
>
> - Editing it as input does not change its output value, and editing it as output does not change its input value.
> - When you save the record, only the output value is retained and applied to the record.

See also *AI Assistant [p 741]*

22.2 Importing and exporting data

In a table, records can be exported to or imported from CSV or XML format.

You can either manually select certain records of the table to export, or you can export the whole table.

See also

CSV Services *[p 159]*

XML Services *[p 153]*

22.3 Restore from history

When history is enabled on a table, it is possible to restore a record to a previous state, based on its registered history. If the record (identified by its primary key) still exists in the table, it will be updated with the historized values to be restored. Otherwise, it will be created.

In order to restore a record to a previous state, select a record in the history table view, and the select **Actions > Restore from history** in the workspace. A summary screen is displayed with the details of the update or creation to be performed.

The restore feature is available only on one record at a time.

If a table trigger must have a specific behavior on restore, different from the one on regular create and update, the developer can use the method `TableTriggerExecutionContext.isHistoryRestore`[API].

> **Note**
>
> This feature has limitations linked to the limitations of the history feature:
>
> - the 'restore from history' feature is not available on tables containing lists that are not supported by history. See Data model limitations [p 466].
> - computed values, encrypted values and fields on which history has been disabled are ignored when restoring a record from history, since these fields are not historized.

See also *History* *[p 461]*

CHAPTER 23

XML import and export

This chapter contains the following topics:

1. Introduction
2. Imports
3. Exports
4. Handling of field values
5. Known limitations

23.1 Introduction

XML imports and exports can be performed on tables through the user interface using the **Actions** menu in the workspace.

Both imports and exports are performed in the context of a dataset.

Imports and exports can also be done programmatically.

Default import and export option values can be set in the Administration area, under *User interface > Graphical interface configuration > Default option values > Import/Export*.

23.2 Imports

Attention

Imported XML documents must be encoded in UTF-8 and its structure must conform to the underlying data model of the target dataset.

Import mode

When importing an XML file, you must specify one of the following import modes, which will dictate how the import procedure handles the source records.

Insert mode	Only record creations are allowed. If a record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled.
Update mode	Only modifications of existing records are allowed. If no record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled.
Update or insert mode	If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created.
Replace (synchronization) mode	If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created. If a record exists in the target table but is not present in the source XML file, that record is deleted from the table.

Insert and update operations

The mode '*by delta*' allows ignoring data model elements that are missing from the source XML document. This mode can be enabled through data services or the Java API. The following table summarizes the behavior of insert and update operations when elements are not present in the source document.

See the data services operations update [p 1087] and insert [p 1089], as well as `ImportSpec.setByDelta`[API] in the Java API for more information.

State in source XML document	Behavior
Element does not exist in the source document	**If 'by delta' mode is disabled (default):** Target field value is set to one of the following: • If the element defines a default value, the target field value is set to that default value. • If the element is of a type other than a string or list, the target field value is set to `null`. • If the element is an aggregated list, the target field value is set to an empty list. • If the element is a string that distinguishes `null` from an empty string, the target field value is set to `null`. If it is a string that does not distinguish between the two, an empty string. • If the element (simple or complex) is hidden in data services, the target value is not changed. **See also** *Hiding a field in Data Services* [p 964] **Note:** The user performing the import must have the permissions necessary to create or change the target field value. Otherwise, the value will remain unchanged. **If 'by delta' mode has been enabled through data services or the Java API:** • For the `update` operation, the field value remains unchanged. • For the `insert` operation, the behavior is the same as when `byDelta` mode is disabled.
Element exists but is empty (for example, `<fieldA/>`)	• For nodes of type `xs:string` (or one of its sub-types), the target field's value is set to `null` if it distinguishes `null` from an empty string. Otherwise, the value is set to empty string. • For non-`xs:string` type nodes, an exception is thrown in conformance with XML Schema. **See also** *TIBCO EBX® whitespace management for data types* [p 948]
Element is present and `null` (for example, `<fieldA xsi:nil="true"/>`)	The target field is always set to `null` except for lists, for which it is not supported. In order to use the `xsi:nil="true"` attribute, you must import the namespace declaration `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`.

Set missing values as null

When updating existing records, if a node is missing or empty in the XML file: if this option is "yes", it will be considered as null. If this option is "no", it will not be modified.

Ignore extra columns

It may happen that the XML document contains elements that do not exist in the target data model. By default, in this case, the import procedure will fail. It is possible, however, to allow users to launch import procedures that will ignore the extra columns defined in the XML files. This can be done in the configuration parameters of the import wizard for XML. The default value of this parameter can be configured in the 'User interface' configuration under the 'Administration' area.

Optimistic locking

If the technical attribute `ebxd:lastTime` exists in the source XML file, the import mechanism performs a verification to prevent an update operation on a record that may have changed since the last read. In order to use the `ebxd:lastTime` attribute, you must import the namespace declaration `xmlns:ebxd="urn:ebx-schemas:deployment_1.0`. The timestamp associated with the current record will be compared to this timestamp. If they are different, the update is rejected.

23.3 **Exports**

> **Note**
>
> Exported XML documents are always encoded in UTF-8.

When exporting to XML, if the table has filters applied, only the records that correspond to the filter are included in the exported file.

The XML export options are as follows:

Download file name	Specifies the name of the XML file to be exported. This field is pre-populated with the name of the table from which the records are being exported.
User-friendly mode	Specifies whether exported values will be represented in a user-friendly way, or in the standard XML raw format. For example, in user-friendly mode, dates and numbers are formatted according to the user's locale, and foreign keys and enumerated values display their associated labels. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Include technical data	Specifies whether internal technical data will be included in the export. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Is indented	Specifies whether the file should be indented to improve its readability by a human.
Omit XML comment	Specifies whether the generated XML comment that describes the location of data and the date of the export should be omitted from the file.

23.4 Handling of field values

Date, time & dateTime format

The following date and time formats are supported:

Type	Format	Example
xs:date	yyyy-MM-dd	2007-12-31
xs:time	HH:mm:ss or HH:mm:ss.SSS	11:55:00
xs:dateTime	yyyy-MM-ddTHH:mm:ss or yyyy-MM-ddTHH:mm:ss.SSS	2007-12-31T11:55:00

23.5 Known limitations

Association fields

The XML import and export services do not support association values.

Exporting such fields will not cause any error, however, no value will be exported.

Importing such fields will cause an error, and the import procedure will be aborted.

Selection nodes

The XML import and export services do not support selection values.

Exporting such fields will not cause any error, however, no value will be exported.

Importing such fields will cause an error, and the import procedure will be aborted.

CHAPTER 24

CSV import and export

This chapter contains the following topics:

1. Introduction
2. Exports
3. Imports
4. Handling of field values
5. Known limitations

24.1 Introduction

CSV imports and exports can be performed on tables through the user interface using the **Actions** menu in the workspace.

Both imports and exports are performed in the context of a dataset.

Imports and exports can also be done programmatically.

Default import and export option values can be set in the Administration area, under *User interface > Graphical interface configuration > Default option values > Import/Export*.

See also *Default option values [p 679]*

24.2 Exports

When exporting to CSV, if the table has filters applied, only the records that correspond to the filter are included in the exported file.

The CSV export options are as follows:

Download file name	Specifies the name of the CSV file to be exported. This field is pre-populated with the name of the table from which the records are being exported.
File encoding	Specifies the character encoding to use for the exported file. The default is UTF-8.
Enable inheritance	In order to consider the inheritance [p 37] during a CSV export, the option has to be defined in the model. For more information on inheritance, see Inheritance and value resolution [p 480]. Specifies if inheritance will be taken into account during a CSV export. If inheritance is enabled, resolved values of fields are exported with the technical data that define the possible inheritance mode of the record or the field. If inheritance is disabled, resolved values of fields are exported and occulted records are ignored. By default, this option is disabled. **Note:** Inheritance is always ignored, if the table dataset has no parent or if the table has no inherited field.
User-friendly mode	Specifies whether exported values will be represented in a user-friendly way, or in a raw format. For example, in user-friendly mode, dates and numbers are formatted according to the user's locale, and foreign keys and enumerated values display their associated labels. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Include technical data	Specifies whether internal technical data will be included in the export. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Column header	Specifies whether or not to include column headers in the CSV file. • **No header** • **Label:** For each column in the spreadsheet, the CSV displays its label. Each label is localized according to the locale preference of the current session. If no user-

	friendly label is defined for a node, the technical name of the node is used. • **XPath:** For each column in the spreadsheet, the CSV displays the path to the node in the table.
Field separator	Specifies the field separator to use for exports. The default separator is comma, it can be modified under *Administration > User interface*.
List separator	Specifies the separator to use for values lists. The default separator is line return, it can be modified under *Administration > User interface*.

Programmatic CSV exports are performed using the classes `ExportSpec`[API] and `ExportImportCSVSpec`[API] in the Java API.

24.3 Imports

Download file name	Specifies the name of the CSV file to be imported.
Import mode	When importing a CSV file, you must specify one of the following import modes, which will control the integrity of operations between the source and the target table. • **Insert mode:** Only record creation is allowed. If a record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled. • **Update mode:** Only modifications of existing records are allowed. If no record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled. • **Update or insert mode:** If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created. • **Replace (synchronization) mode:** If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created. If a record exists in the target table but is not present in the source XML file, that record is deleted from the table.
File encoding	Specifies the character encoding to use for the exported file. The default is UTF-8.
Enable inheritance	In order to consider the inheritance [p 37] during a CSV import, the option has to be defined in the model. For more information on inheritance, see Inheritance and value resolution [p 480] and `ExportImportCSVSpec.setInheritanceEnabled`[API]. Specifies whether the inheritance will be taken into account during a CSV import. If technical data in the CSV file define an inherit mode, corresponding fields or records are forced to be inherited. If technical data define an occult mode, corresponding records are forced to be occulted. Otherwise, fields are overwritten with values read from the CSV file. By default, this option is disabled. **Note:** Inheritance is always ignored if the dataset of the table has no parent or if the table has no inherited field.

Column header	Specifies whether or not to include column headers in the CSV file. • **No header** • **Label:** For each column in the spreadsheet, the CSV displays its label. Each label is localized according to the locale preference of the current session. If no user-friendly label is defined for a node, the technical name of the node is used. • **XPath:** For each column in the spreadsheet, the CSV displays the path to the node in the table.
Field separator	Specifies the field separator to use for exports. The default separator is comma, it can be modified under *Administration > User interface*.
List separator	Specifies the separator to use for values lists. The default separator is line return, it can be modified under *Administration > User interface*.

Programmatic CSV imports are performed using the classes `ImportSpec`[API] and `ExportImportCSVSpec`[API] in the Java API.

24.4 Handling of field values

Aggregated lists

The CSV import and export services support multi-valued fields, namely aggregated lists. This is only supported for simple typed lists, such as lists of `string`, `date`, or `int`, and for foreign keys. If a table reference is linked to a composite primary key, each item in the list is a formatted string, for example, "true|99". Aggregated lists of groups are not exported.

At export, the items in the list are separated using line separators. In cases where the exported field already contains a line separator, for example in an `osd:html` or an `osd:text`, the code `_crnl_` is inserted in place of the field value's line separators. The same formatting is expected at import, with the whole field value surrounded by quotes.

Hidden fields

Hidden fields are exported as `ebx-csv:hidden` strings. An imported hidden string will not modify a field's content.

'Null' value for strings

Using CSV import and export services, a string with a value set to `null` is exported as an empty string. Therefore, a round trip export-import procedure will end up replacing `null` string values with empty strings.

Using programmatic services, the specific value `ebx-csv:nil` can be assigned to strings with values set to `null`. If this is done, the `null` string values will not be replaced by empty strings during round

trip export-import procedures. See `ExportImportCSVSpec.setNullStringEncoded`[API] in the Java API for more information.

Date, time & dateTime format

The following date and time formats are supported:

Type	Format	Example
xs:date	yyyy-MM-dd	2007-12-31
xs:time	HH:mm:ss or HH:mm:ss.SSS	11:55:00
xs:dateTime	yyyy-MM-ddTHH:mm:ss or yyyy-MM-ddTHH:mm:ss.SSS	2007-12-31T11:55:00

24.5 Known limitations

Aggregated lists of groups

The CSV import and export services do not support multi-valued groups, that is, aggregated lists of complex type elements. Exporting such nodes will not cause any error, however, no value will be exported.

Terminal groups

In a CSV file, it is impossible to differentiate a created terminal group that contains only empty fields from a non-created one.

As a consequence, some differences may appear during comparison after performing an export followed by an import. To ensure the symmetry of import and export, use XML import and export instead. See XML import and export [p 153].

Column label headers

If two columns share the same label header, an export of the table can be performed successfully, but exported data cannot later be re-imported.

Association fields

The CSV import and export services do not support association values, i.e. the associated records.

Exporting such fields will not cause any error, however, no value will be exported.

Importing such fields will cause an error and the import procedure will be aborted.

Selection nodes

The CSV import and export services do not support selection values, i.e. the selected records.

Exporting such fields will not cause any error, however, no value will be exported.

Importing such fields will cause an error and the import procedure will be aborted.

CHAPTER 25

Working with existing datasets

This chapter contains the following topics:

1. Validating a dataset
2. Duplicating a dataset
3. Deactivating a dataset
4. Managing dataset permissions

25.1 Validating a dataset

To validate a dataset at any time, select **Actions > Validate** from the navigation pane. A generated report provides the results of the validation. From the validation report, you have the option to update the reported validation status by clicking the **Revalidate** button, or to click the **Reinitialize validation report** button to clear the current validation report associated with the dataset in order to be able to rerun a full validation from scratch.

Validations of data can also be run at the table level by navigating to the desired table from the navigation pane, then selecting **Actions > Validate** from the workspace.

See Validation [p 617] for detailed information about incremental data validation.

25.2 Duplicating a dataset

To duplicate an existing dataset, select it from the '**Select dataset** [p 123]' menu in the navigation pane, then select **Actions > Duplicate**.

25.3 Deactivating a dataset

When a dataset is activated, it will be subject to validation. That is, all mandatory elements must be defined in order for the dataset to be valid. If a dataset is active and validated, it can be safely exported to external systems or to be used by other Java applications.

If a dataset is missing mandatory elements, it can be deactivated by setting the property 'Activated' to 'No' from **Actions > Information**.

25.4 Managing dataset permissions

Dataset permissions can be accessed by selecting **Actions > Permissions** in the navigation pane.

Permissions are defined using *profile* records. To define a new permissions profile, create a new record in the 'Access rights by profile' table.

See also *Profile* *[p 33]*

Profile	Defines the profile to which these permissions apply.
Restriction policy	If 'Yes', indicates that when the permissions defined here are more strict than otherwise defined, these permissions are respected. This is contrary to the default where the most permissive rights defined take precedence. See Resolving user-defined rules [p 501].
Dataset actions	Specifies the permissions for actions on the dataset.
Create a child dataset	Indicates whether the profile can create a child dataset. Inheritance also must be activated in the data model.
Duplicate the dataset	Indicates whether the profile can duplicate the dataset.
Delete the dataset	Indicates whether the profile can delete the dataset.
Activate/deactivate the dataset	Indicates whether the profile can modify the *Activated* property in the dataset information. See Deactivating a dataset [p 165].
Create a view	Indicates whether the profile can create views and hierarchies in the dataset.
Tables policy	Specifies the default permissions for all tables. Specific permissions can also be defined for a table by clicking the '+' button.
Create a new record	Indicates whether the profile can create records in the table.
Overwrite inherited record	Indicates whether the profile can override inherited records in the table. This permission is useful when using dataset inheritance.
Occult inherited record	Indicates whether the profile can occult inherited records in the table. This permission is useful when using dataset inheritance.
Delete a record	Indicates whether the profile can delete records in the table.
Values access policy	Specifies the default access permissions for all the nodes of the dataset and allows the definition of permissions for

	specific nodes. The default access permissions are used if no custom permissions have been defined for a node. The specific policy selector allows granting specific access permissions for a node. The links "ReadOnly", "ReadWrite", and "Hidden" set the corresponding access levels for the selected nodes. It is possible to remove custom access permissions using the "(default)" link.
Rights on services	This section specifies the access permissions for services. A service is not accessible to a profile if it is crossed-out.

CHAPTER 26

Dataset inheritance

Using the concept of dataset inheritance, it is possible to create child datasets that branch from a parent dataset. Child datasets inherit values and properties by default from the parent dataset, which they can then override if necessary. Multiple levels of inheritance can exist.

An example of using dataset inheritance is to define global default values in a parent dataset, and create child datasets for specific geographical areas, where those default values can be overridden.

> **Note**
>
> By default, dataset inheritance is disabled. It must be explicitly activated in the underlying data model.

See also *Data model configuration* *[p 52]*

This chapter contains the following topics:

1. Dataset inheritance structure
2. Value inheritance

26.1 Dataset inheritance structure

Once the root dataset has been created, create a child dataset from it using the + button in the dataset selector in the navigation pane.

> **Note**
>
> - A dataset cannot be deleted if it has child datasets. The child datasets must be deleted first.
> - If a child dataset is duplicated, the newly created dataset will be inserted into the existing dataset tree as a sibling of the duplicated dataset.

26.2 Value inheritance

When a child dataset is created, it inherits all its field values from the parent dataset. A record can either keep the default inherited value or override it.

In tabular views, inherited values are marked in the top left corner of the cell.

The button can be used to override a value.

Record inheritance

A table in a child dataset inherits the records from the tables of its ancestor datasets. The table in the child dataset can add, modify, or delete records. Several states are defined to differentiate between types of records.

Root	A root record is a record that was created in the current dataset and does not exist in the parent dataset. A root record is inherited by the child datasets of the current dataset.
Inherited	An inherited record is one that is defined in an ancestor dataset of the current dataset.
Overwritten	An overwritten record is an inherited record whose values have been modified in the current dataset. The overwritten values are inherited by the child datasets of the current dataset.
Occulted	An occulted record is an inherited record which has been deleted in the current dataset. It will still appear in the current dataset as a record that is crossed out, but it will not be inherited in the child datasets of the current dataset.

When the inheritance button is toggled on, it indicates that the record or value is inherited from the parent dataset. This button can be toggled off to override the record or value. For an occulted record, toggle the button on to revert it to inheriting.

The following table summarizes the behavior of records when creating, modifying or deleting a record, depending on its initial state.

State	Create	Modify value	Delete
Root	Standard new record creation. The newly created record will be inherited in child datasets of the current dataset.	Standard modification of an existing record. The modified values will be inherited in the child datasets of the current dataset.	Standard record deletion. The record will no longer appear in the current dataset and the child datasets of the current dataset.
Inherited	If a record is created using the same primary key as an existing inherited record, that record will be overwritten and its value will be the one submitted at creation.	An inherited record must first be marked as overwritten in order to modify its values.	Deleting an inherited record changes it state to occulted.
Overwritten	Not applicable. Cannot create a new record if the primary key is already used in the current dataset.	An overridden record can be returned to the inherited state, but its modified value will be lost. Individual values in an overridden record can be set to inheriting or can be modified.	Deleting an overwritten record changes its state to occulted.
Occulted	If a record is created using the primary key of an existing occulted record, the record state will be changed to overwritten and its value modified according to the one submitted at creation.	Not applicable. An occulted record cannot be modified.	Not applicable. An occulted record is already considered to be deleted.

See also *Record lookup mechanism* *[p 482]*

Collaborative features (Team Up)

CHAPTER 27

Introduction to collaborative features (Team Up)

This chapter contains the following topics:

1. Overview
2. Accessing the Team Up features

27.1 Overview

Use Team Up to collaborate and engage with others. Leave comments on records, rate records, and assign tags to records. These interactions can promote a better overall understanding of data.

The following provides an overview of Team Up features:

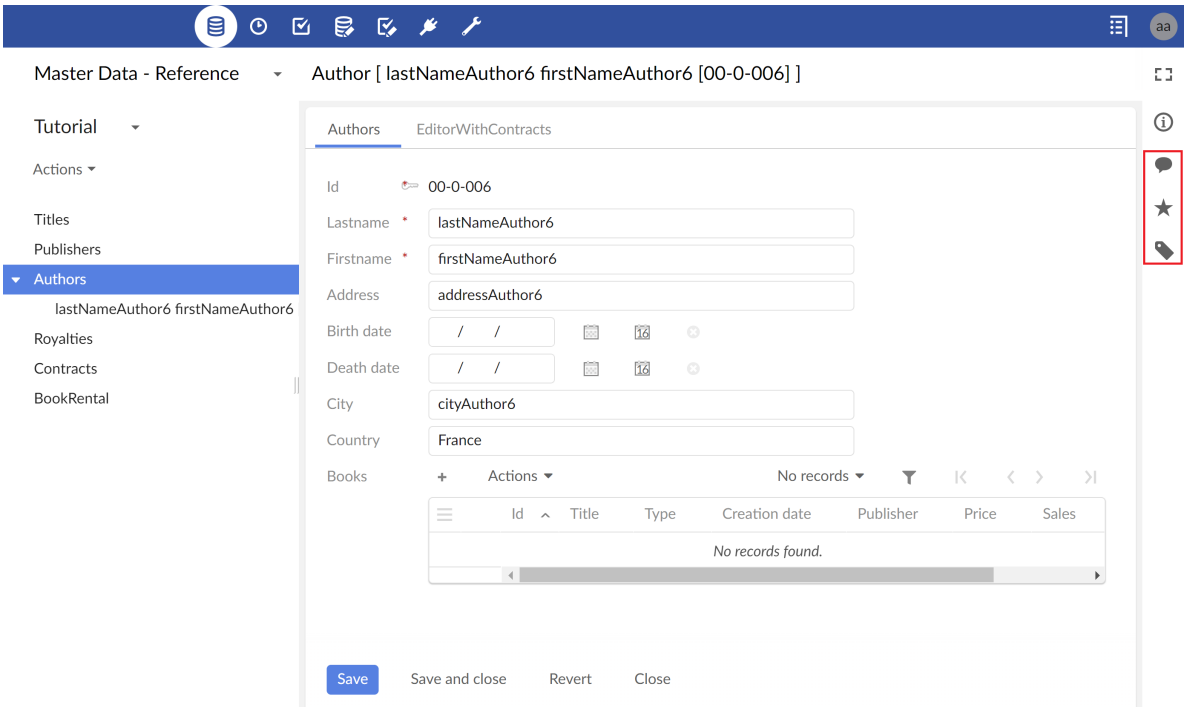
- Comments: Leave comments on records, reply, and react to other's comments. Assign moderators to tables that can manage comments from other users.
- Tags: Apply tags to records. Tags are classified in tag catalogs. Assign an administrator to manage tag catalogs.
- Ratings: Leave a rating from 1 to 5 stars on each record. When leaving a rating, optionally add a note to explain it.

27.2 Accessing the Team Up features

To access the Team Up features available for a record:

1. Open a record details page.
2. In the right sidebar, click the icon corresponding to the Team Up feature you want to use.

Known limitation: The Team Up features are not accessible when a custom user service is opened on the main workspace.



See also

Comments [p 177]

Ratings [p 183]

Tags [p 187]

CHAPTER 28

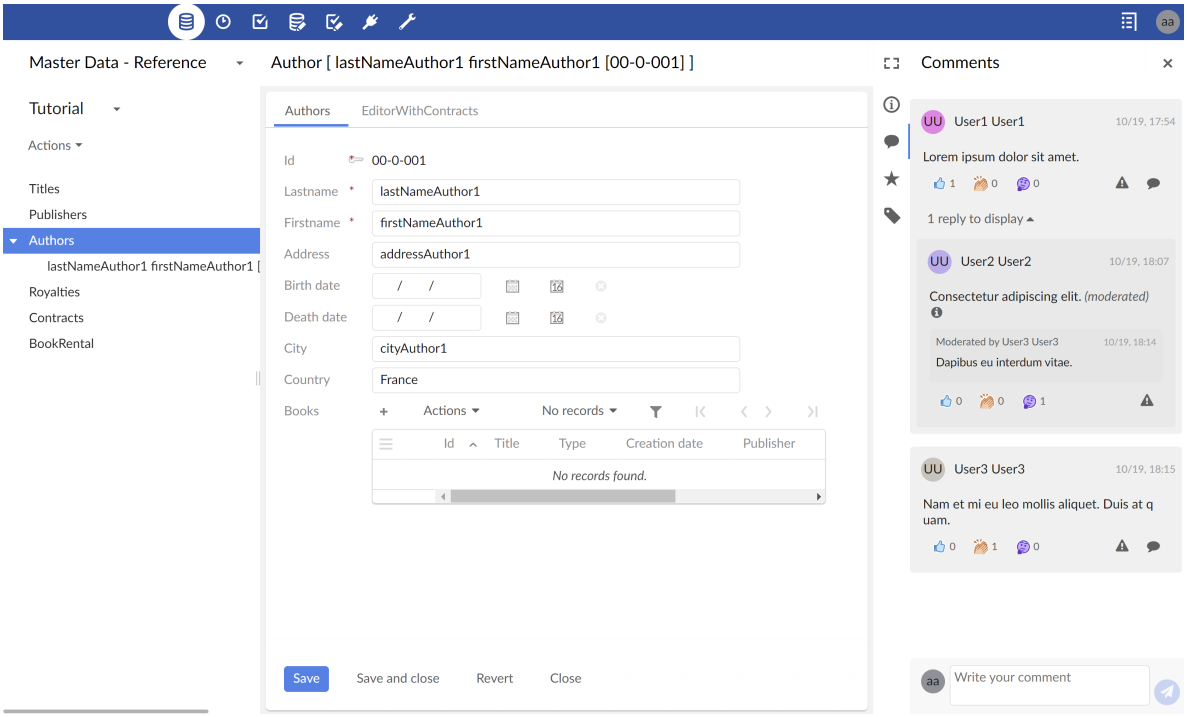
Comments

This chapter contains the following topics:

1. Overview
2. Adding a comment
3. Editing a comment
4. Deleting a comment
5. Replying to a comment
6. Reacting to a comment
7. Moderating a comment

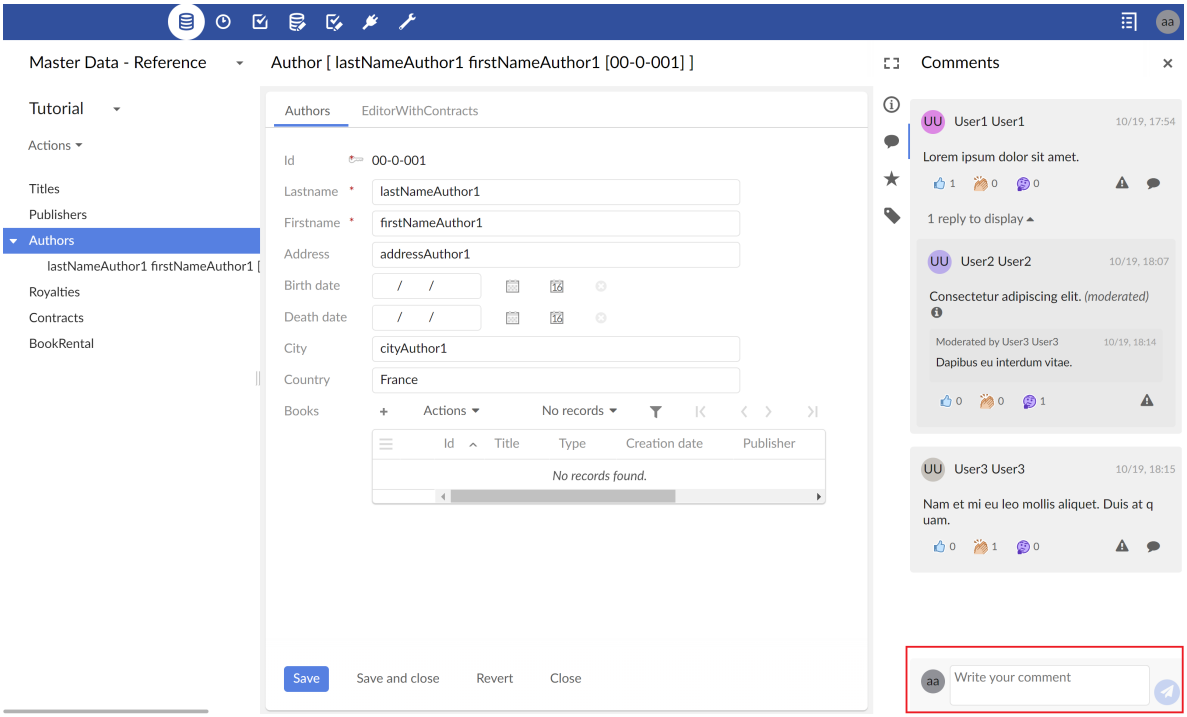
28.1 Overview

Use the Team Up comments feature to discuss the details of a record. Add your own comments, or reply and react to comments from other users.



28.2 **Adding a comment**

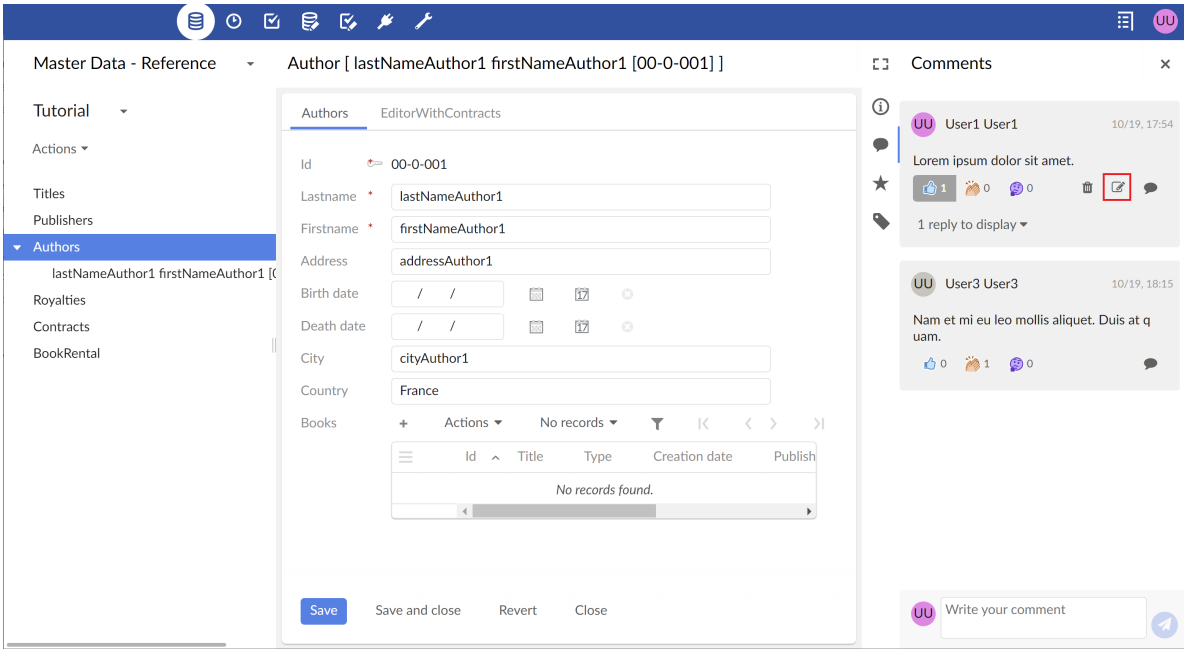
Add a comment using the text input at the bottom of the **Comments** sidebar.



28.3 **Editing a comment**

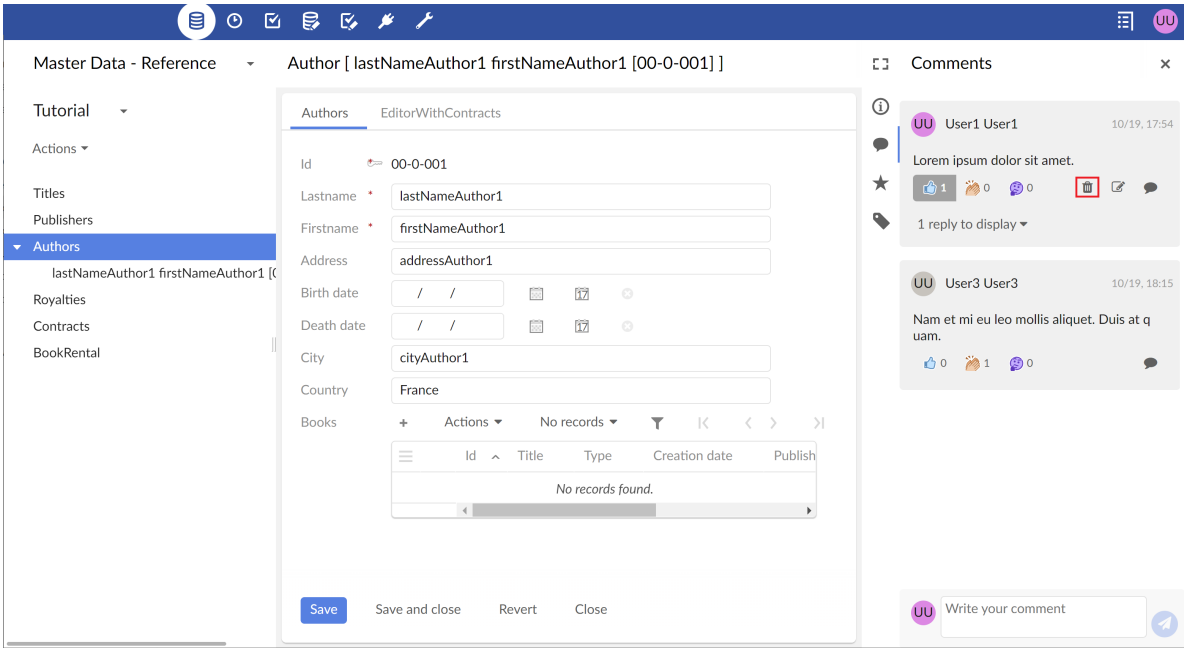
Edit your comments by clicking the editing button in the toolbar (highlighted below). Note that comments edited by a moderator cannot be edited.

For details about comment moderation, see Moderating a comment [p 180].



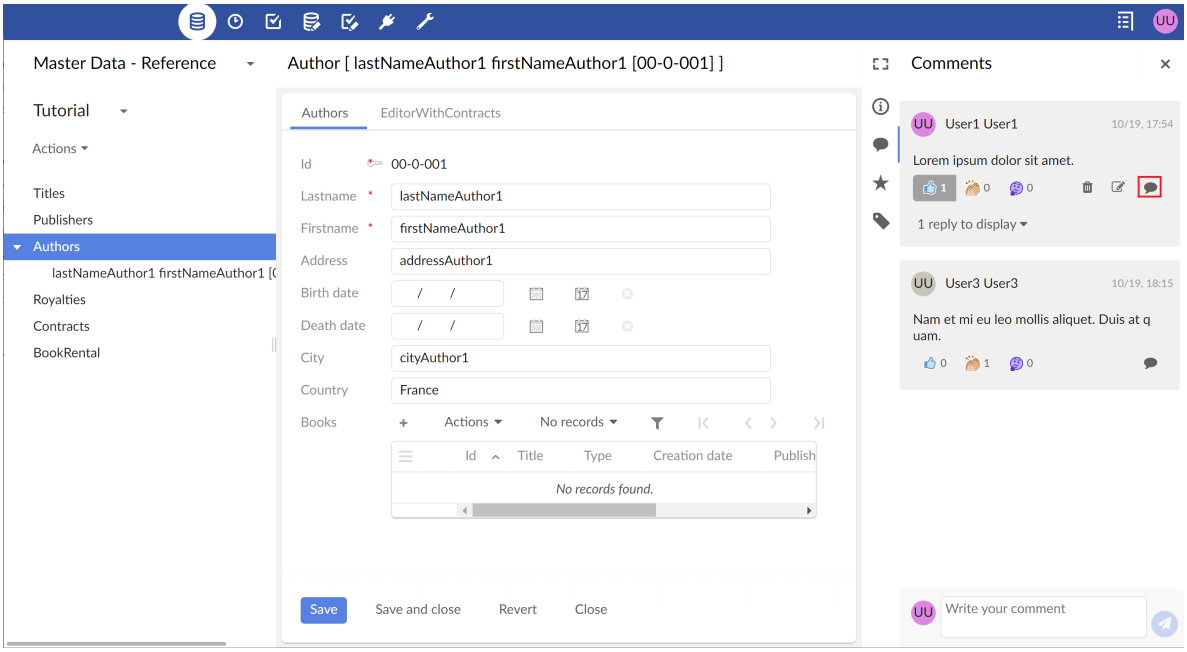
28.4 **Deleting a comment**

Delete one of your comments by clicking the delete button in the toolbar located under the comment.



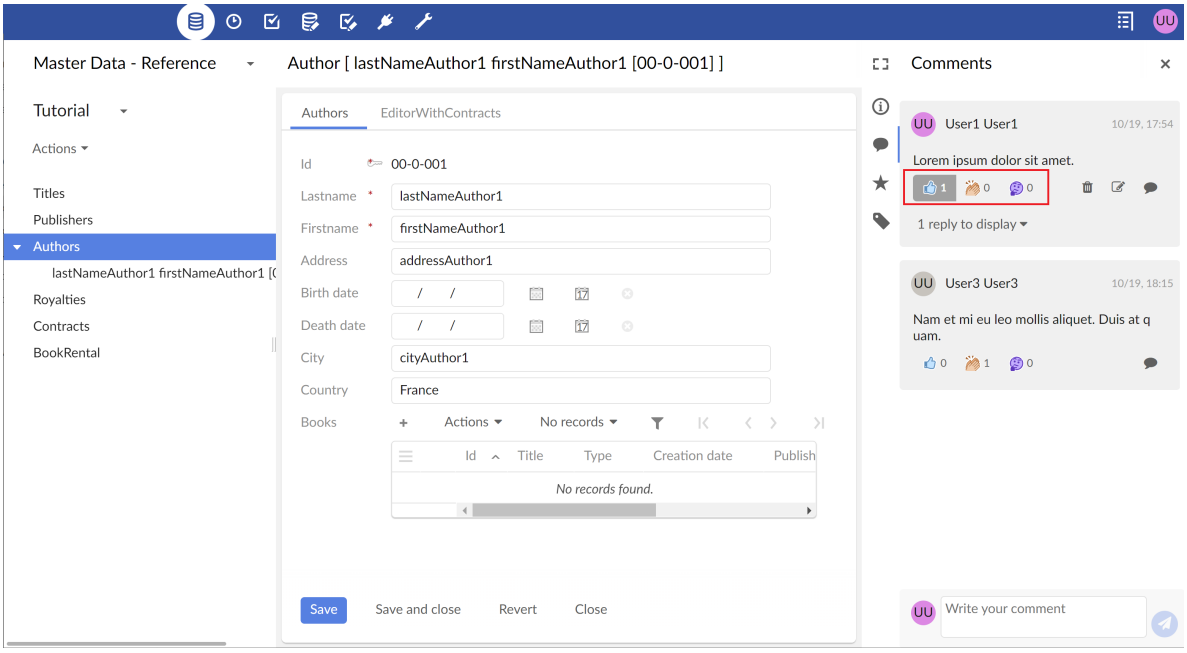
28.5 **Replying to a comment**

Reply to a comment using the reply button in the toolbar located under the comment.



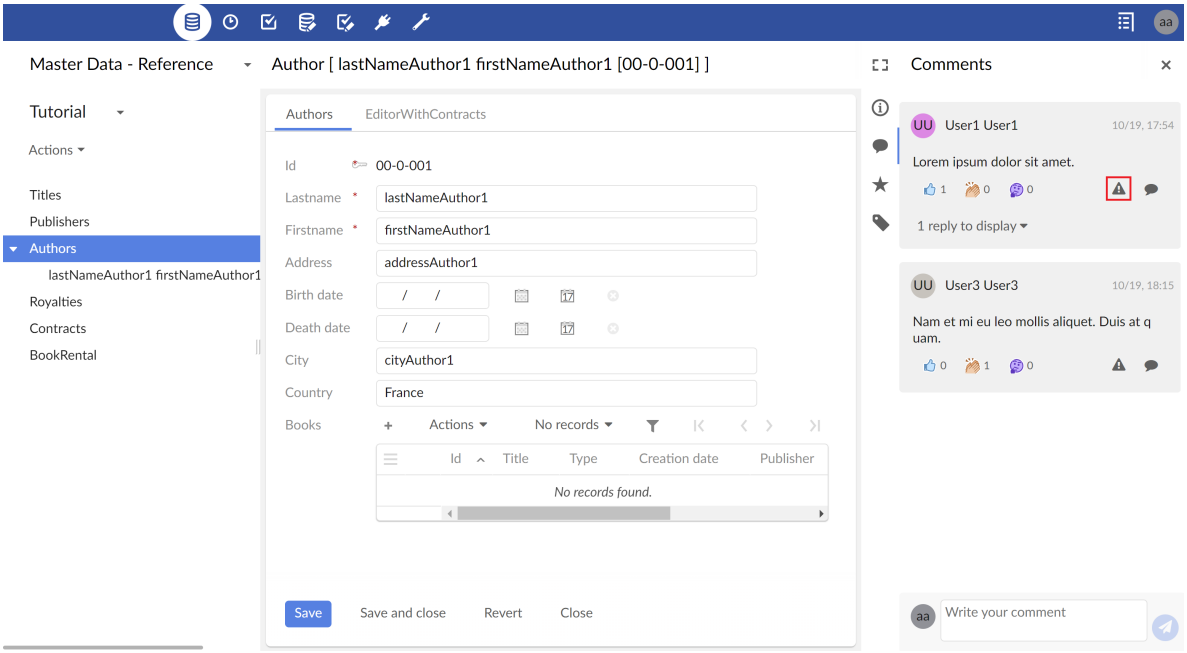
28.6 **Reacting to a comment**

React to comments using the available reaction buttons in the toolbar located under the comment. Change your reaction by clicking a different reaction; remove your reaction by clicking the same reaction again.



28.7 **Moderating a comment**

Moderate and edit other's comments by clicking the moderate icon highlighted below. When editing other's comments you can leave a note that indicates why the original content was moderated.



For details about how to manage moderators, see Managing user access and Team Up features [p 738].

See also

Ratings [p 183]

Tags [p 187]

CHAPTER 29

Ratings

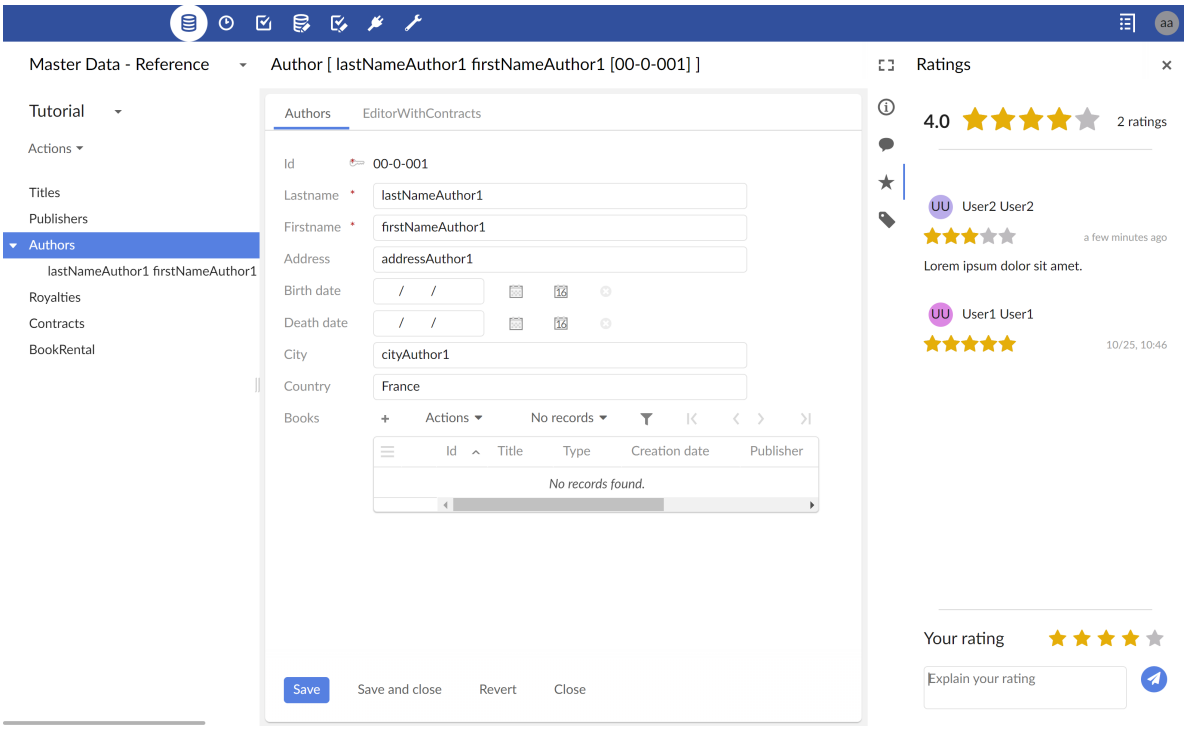
This chapter contains the following topics:

1. Overview
2. Rating a record and editing a rating
3. Deleting a rating

29.1 Overview

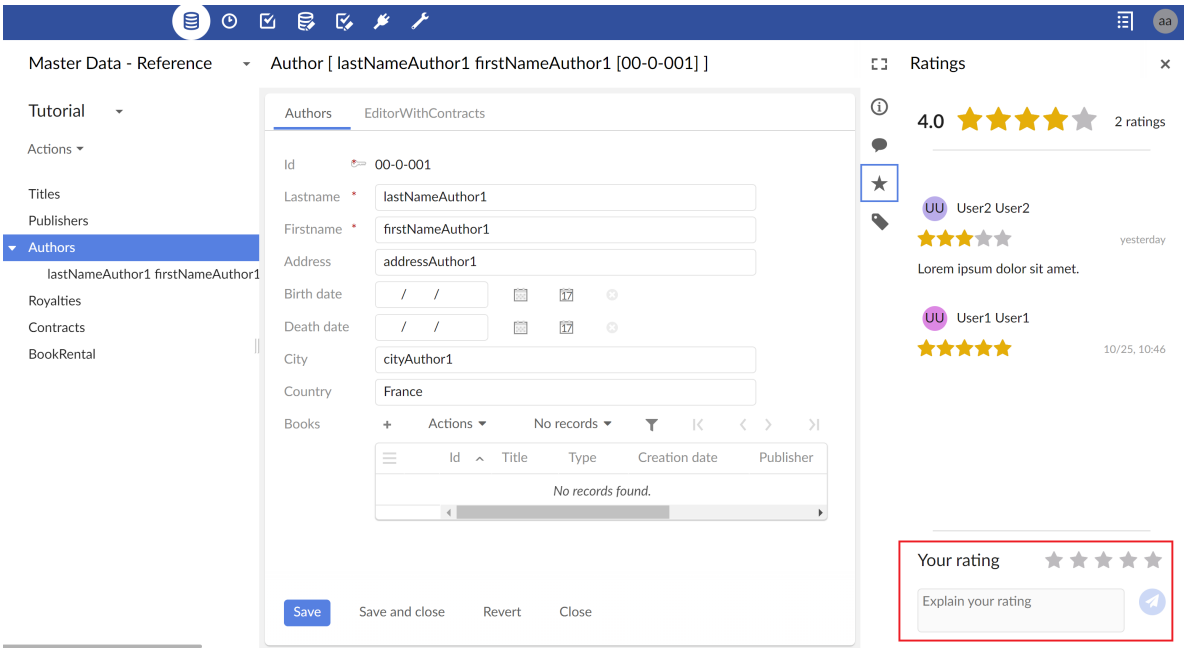
Give records ratings by leaving up to 5 stars and optionally leave an explanation for the rating. The **Ratings** sidebar displays:

- A record's overall rating.
- The individual ratings supplied by other users.
- The comments related to those ratings.



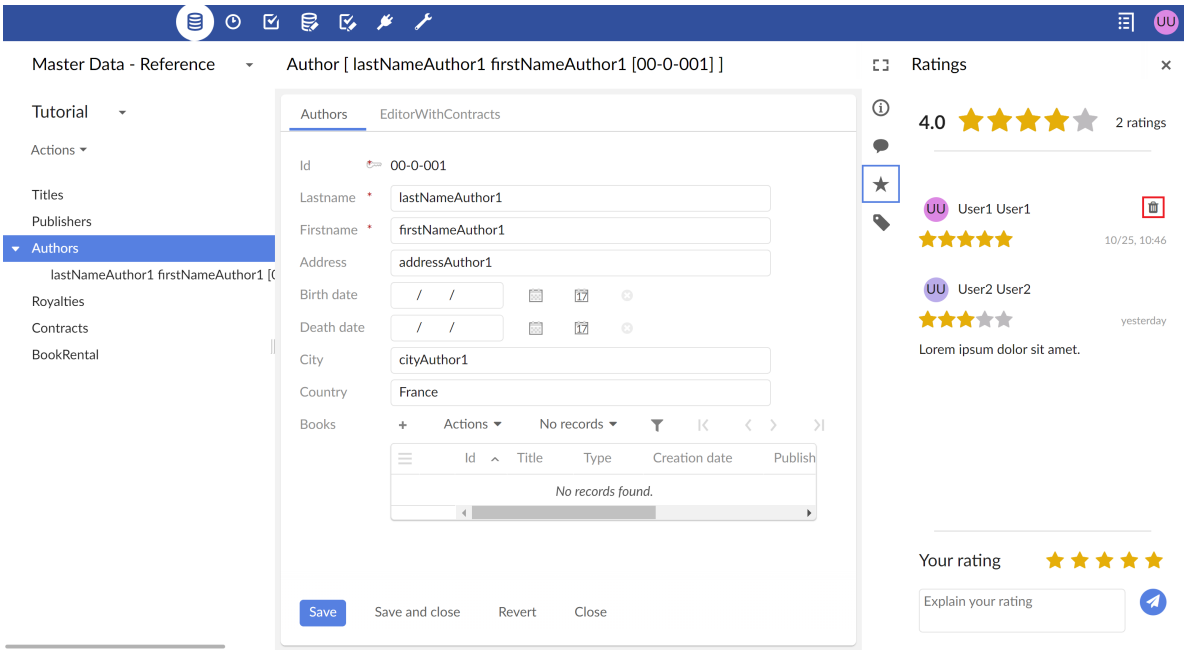
29.2 **Rating a record and editing a rating**

Use the inputs at the bottom of the **Rating** sidebar to add your rating and optionally leave a comment. Perform the same actions to edit your rating.



29.3 **Deleting a rating**

Delete your rating using the delete icon located to the right of your rating. Note that your rating always displays at the top of the individual ratings list.



See also

Comments *[p 177]*

Tags *[p 187]*

CHAPTER 30

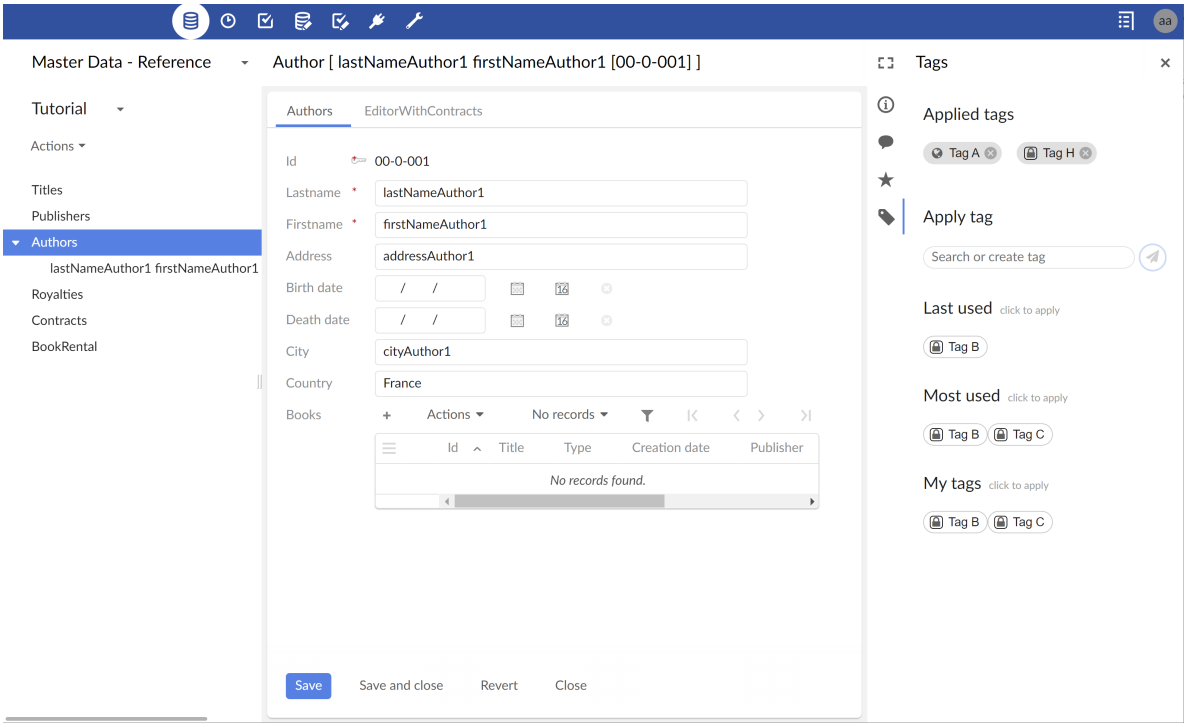
Tags

This chapter contains the following topics:

1. Overview
2. Applying tags
3. Removing a tag

30.1 Overview

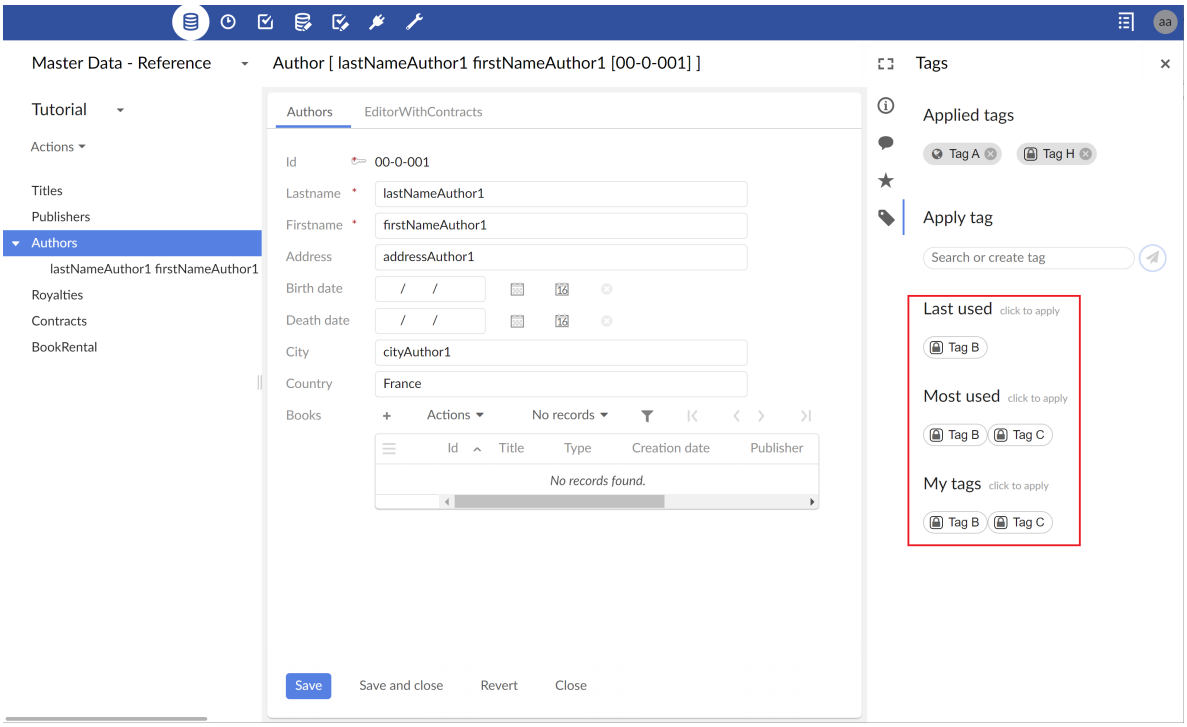
Use the **Tags** sidebar to assign one or more tags to a record.



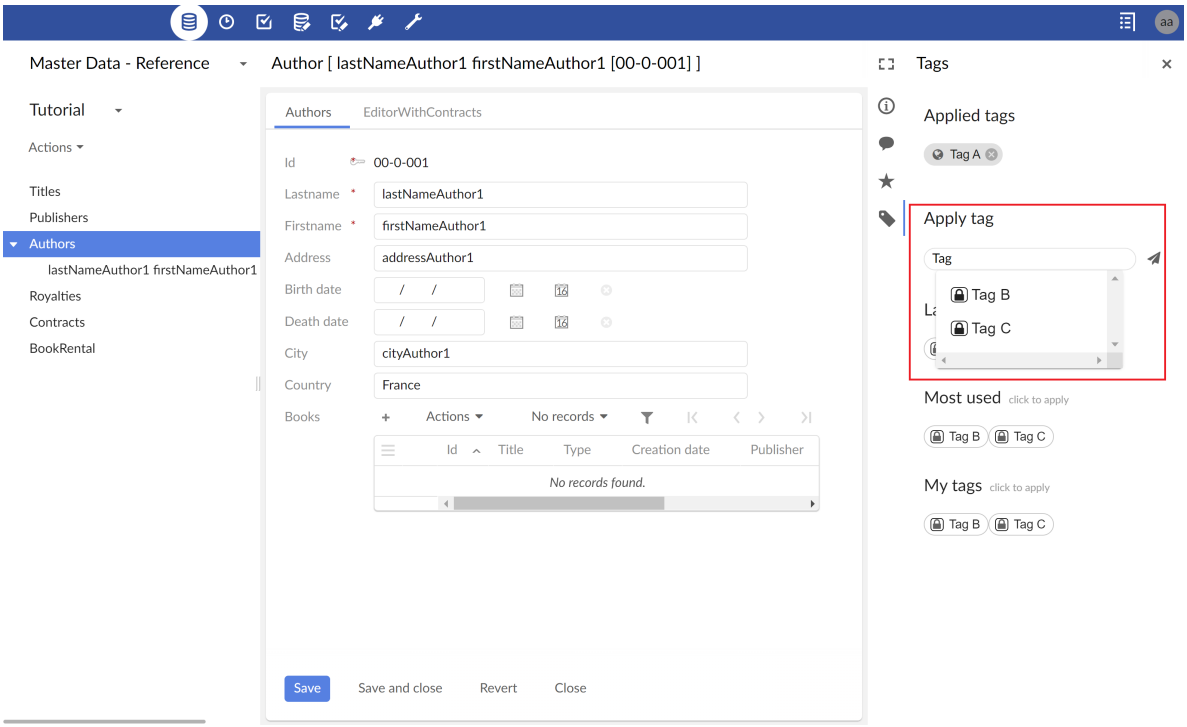
30.2 **Applying tags**

Applying existing tags

Apply a tag from the sections **Last used**, **Most used** and **My tags** by clicking on it.



If the required tag is not displayed under the sections **Last used**, **Most used** and **My tags** then search for it in the table's tags catalog using the input field **Search or create**.

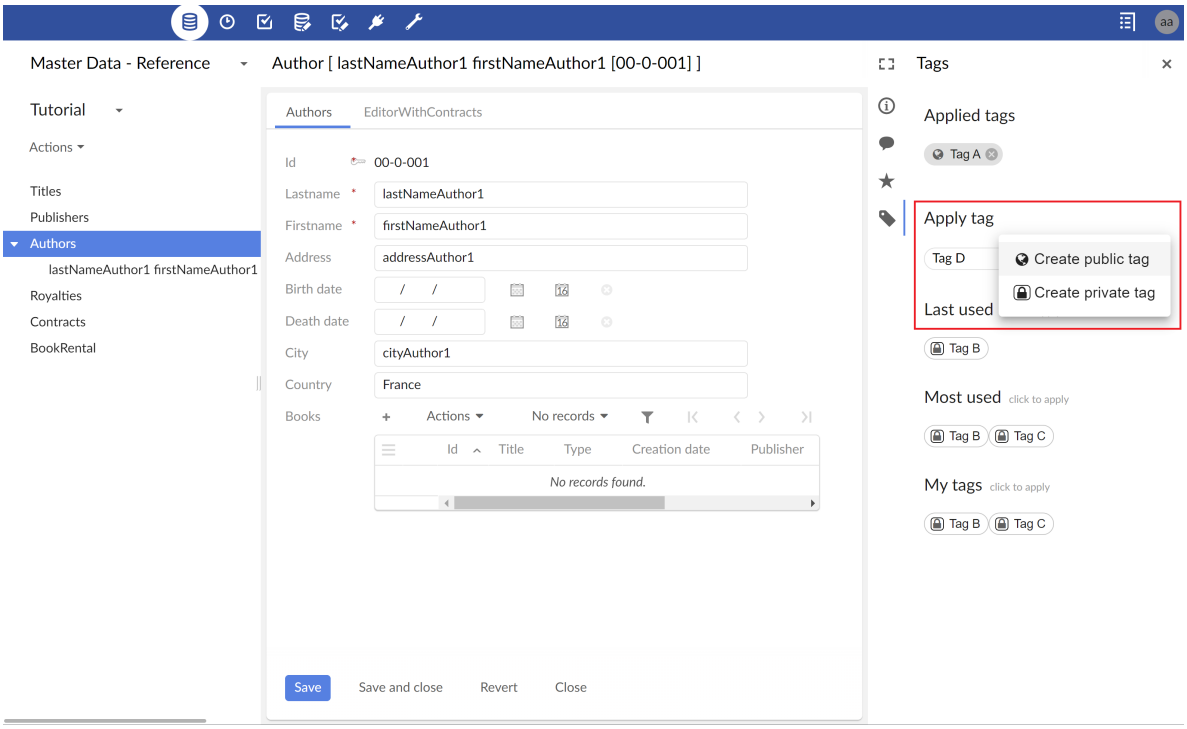


For details about managing tag catalogs, see Managing tags and tag catalogs [p 738].

Applying a new tag

If the required tag doesn't exist in the table's tags catalog, then create it by typing its name into the input field **Search or create** and pressing Enter or by clicking the **Send** button.

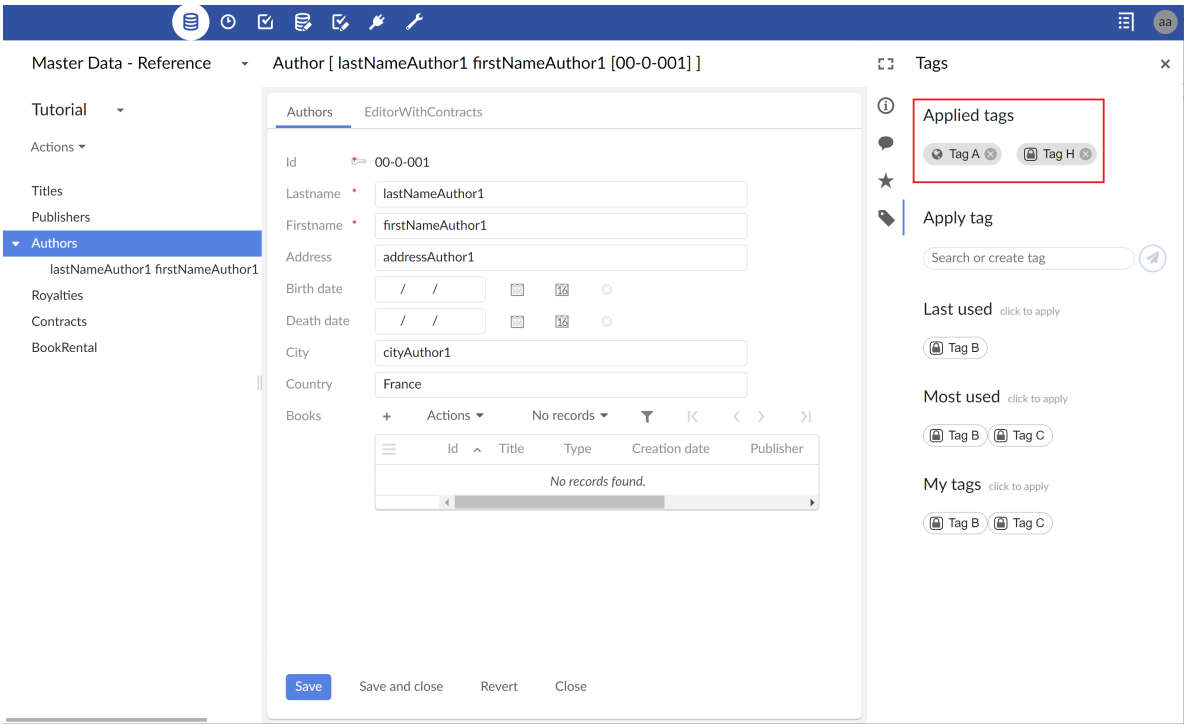
If you are an administrator for the table's tag catalog, you can choose between making the tag public or private when creating a new tag. All profiles can see public tags, and only creators can see private tags.



For details about managing tag catalog administrators, see Managing tags and tag catalogs [p 738].

30.3 **Removing a tag**

Remove a tag assigned to the record using the 'x' button that appears on the tag under the section **Applied tags**.



See also

Comments *[p 177]*

Ratings *[p 183]*

Workflow models

CHAPTER 31

Introduction to workflow models

This chapter contains the following topics:

1. Overview
2. Using the Workflow Models area user interface
3. Generic message templates
4. Limitations of workflows

31.1 Overview

What is a workflow model?

Workflows in TIBCO EBX® facilitate the collaborative management of data in the repository. A workflow can include human actions on data and automated tasks alike, while supporting notifications on certain events.

The first step of realizing a workflow is to create a *workflow model* that defines the progression of steps, responsibilities of users, as well as other behavior related to the workflow.

Once a workflow model has been defined, it can be validated and published as a *workflow publication*. Data workflows can then be launched from the workflow publication to execute the steps defined in the workflow model.

See also

Workflow model (glossary) *[p 39]*

Data workflow (glossary) *[p 41]*

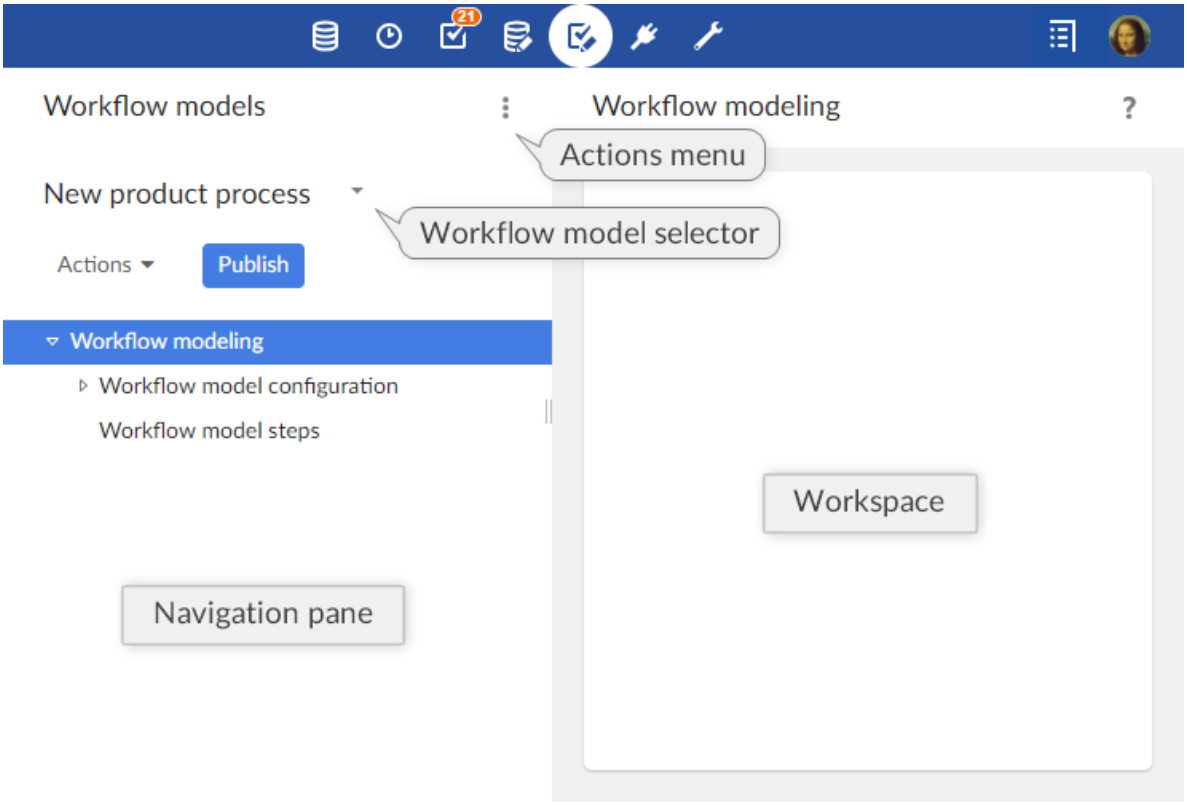
Basic concepts related to workflow models

A basic understanding of the following terms is necessary to proceed with the creation of workflow models:

- script task [p 40]
- user task [p 40]
- work item [p 41]
- workflow condition [p 40]
- sub-workflow invocation [p 40]

- wait task [p 40]
- data context [p 40]

31.2 Using the Workflow Models area user interface



> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective'. Only authorized users can access these interfaces.

31.3 Generic message templates

Notification emails can be sent to inform users of specific events during the execution of data workflows.

Generic templates can be defined and reused by any workflow model in the repository. To work with generic templates, select 'Message templates' from the Workflow Models area **Actions** menu.

These templates, which are shared by all workflow models, are included statically at workflow model publication. Thus, in order to take template changes into account, you must update your existing publication by re-publishing the affected models.

Please note that, if you want to export those templates in an archive, you will have to select the dataset "configuration" as it is the one containing the message templates.

When creating a new template, two fields are required:

- **Label & Description**: Specifies the localized labels and descriptions associated with the template.
- **Message**: Specifies the localized subjects and bodies of the message.

The message template can include data context variables, such as `${variable.name}`, which are replaced when notifications are sent. System variables that can be used include:

system.time	System time of the repository.
system.date	System date of the repository.
workflow.lastComment	Last comment on the previous user task. (Note: this variable refers to the last user task, not the current one. Also the current task is the one on which the workflow is positioned, and it also includes the completion notification of a user task).
workflow.lastDecision	Last decisions made on the previous user task. (Note: this variable refers to the last user task, not the current one. Also the current task is the one on which the workflow is positioned, and it also includes the completion notification of a user task).
user.fullName	Full name of the notified user.
user.login	Login of the notified user.
workflow.process.label	Label of the current workflow.
workflow.process.description	Description of the current workflow.
workflow.workItem.key	Key of the current work item.
workflow.workItem.label	Label of the current work item.
workflow.workItem.description	Description of the current work item.
workflow.workItem.offeredTo	Role to which the current work item has been offered.
workflow.workItem.allocatedTo	User to whom the current work item has been allocated.
workflow.workItem.link	Link to access the current work item in the work item inbox, using the Web Component API. This link can only be computed if a current work item is defined and if the URL is configured in Workflow-executions, in the email configuration.

workflow.workItem.link.allocateAndStart	Link to access the current work item in the work item inbox, using the Web Component API. If the target work item has not yet been started, it will be automatically allocated to and started by the user clicking the link. This link can only be computed if a current work item is defined and if the URL is configured in Workflow-executions, in the email configuration.
workflow.currentStep.label	Label of the current step.
workflow.currentStep.description	Description of the current step.

Example

Generic template message:

```
Today at ${system.time}, a new work item was offered to you
```

Resulting email:

```
Today at 15:19, a new work item was offered to you
```

31.4 Limitations of workflows

The following functionality is currently unsupported in EBX®:

- **Scheduled tasks**, task executed as soon as its turn comes, and whose execution cannot be delayed.
- **Event tasks**, allowing the workflow to move forward upon receiving an event, such as a web service call.
- **Time limitation** on a task duration.
- **User profile names can be up to 64 characters**, otherwise compatibility is not guaranteed.

Related concepts *Data workflows [p 224]*

CHAPTER 32

Creating and implementing a workflow model

This chapter contains the following topics:

1. Creating a workflow model
2. Implementing the steps
3. User tasks
4. Script tasks
5. Conditions
6. Sub-workflow invocations
7. Wait tasks
8. Editing the workflow diagram

32.1 Creating a workflow model

A new workflow model can be created in the **Workflow Models** area. The only required information at creation is a name that is unique in the repository.

The steps of the workflow model are initialized with a single transition. In order to fully implement the workflow model, you must define the sequence of steps beyond this initial transition.

32.2 Implementing the steps

A workflow model defines steps that correspond to different operations that must be performed on data, and associated conditions. The following types of steps exist:

- User task
- Script task
- Condition
- Sub-workflow invocation
- Wait task

A data context is linked to each data workflow. This data context can be used to define variables that can be used as input and output variables in the steps of the workflow.

Progress strategy of the next step

For each step type (excluding user task with multiple work items), a property is available to define which progress strategy has to be applied for the next step. Upon step completion, this strategy is evaluated in order to define the navigation when the workflow is executed. By default, the progress strategy is set to 'Display the work items table'. In that case, after the step has been executed, the work items table (work items inbox or monitoring > work items) is automatically displayed, in order to select the following work item.

Another strategy can be selected: 'Automatically open the next step'. This strategy allows the user to keep working on this workflow and to directly execute the next step. If, following to this execution, a work item is reached and the connected user can start it, then the work item is automatically opened (if several work items are reached, the first created is opened). Otherwise, the next step progress strategy is evaluated. If no work item has been reached, the work items table will be displayed.

This strategy is used to execute several steps in a row without going back to the work items inbox.

There are some limitations that will lead to disregard this strategy. In that case, the work items table is automatically displayed. This property will be disregarded when: the current step is a user task with more than one work item.

In the case of conditions, two other strategies are available: 'If true, automatically open the next step' and 'If false, automatically open the next step'. These strategies allow choosing which strategy will be applied according to the condition result.

In the case of sub-workflows invocation, a dedicated property « Foreground sub-workflow » is available to precise the progress strategy. For the previous progress strategy « Open directly the next step », a foreground sub-workflow must be selected. Only the steps and associated progress strategy included in this foreground sub-workflow will be evaluated. Please note the following specific rules about the foreground sub-workflow property:

- If only one sub-workflow is launched, it is automatically considered in the foreground,
- The foreground sub-workflow property will be ignored if the previous step has not selected the progress strategy « Automatically open the next step »,
- When all the sub-workflow are completed, and if the last completed sub-workflow is the foreground one, then the progress strategy defined on the sub-workflow invocation step is evaluated: if the progress strategy is « automatically open the next step », the next step can be opened without displaying the inbox. In all cases, the progress strategy of the last step of the sub-workflow is always ignored.

Hidden in progress view

For each step type, a property is available to define which steps must be hidden in the workflow progress view by default.

If this property is enabled, the step will be automatically hidden in the workflow progress view for non-administrator users (neither built-in administrator nor workflow administrator). Hidden steps can be displayed on demand.

32.3 **User tasks**

User tasks are steps that involve actions to be performed by human users. Their labels and descriptions can be localized.

Mode

For backward compatibility reasons, two user task modes are available: the default mode and the legacy mode.

By default, a user task generates a single work item. This mode offers more features, such as offering a work item to a list of profiles or directly displaying the avatars in the workflow progress view.

In the legacy mode, a user task can generate several work items.

By default, the user task creation service is hidden in legacy mode. To display it, a property should be enabled in the `ebx.properties` file. For more information, see Disabling user task legacy mode [p 595].

List of profiles

The definition of the profiles of a user task may vary depending on the user task mode.

[Default] Offered to the following profiles

The defined profiles are the roles or the users to whom the user task is being offered. When executing the user task, a single work item is generated. If a single user is defined, the work item is automatically assigned to this user. If a role is defined, the work item is offered to the members of the role. If several users and roles are defined, the work item is offered simultaneously to these users and to the members of these roles.

[Legacy mode] Participants

The participants are the roles or the users to whom the user task is intended. By default, when executing the user task, a work item is generated by profile. If a profile refers to a user instead of a role, the work item is directly allocated to that user. If a profile is a role, the work item is offered to the members of the role.

For more information

See also *Extension [p 203]*

Setting a user allocation preference

You can set a preference for allocation of the current task to the user who most recently completed an earlier task in the workflow. Use the menu below **Prefer allocation to the user who completes the following task** to select the earlier task. If the user who completed that task belongs to any profile the current task is offered to, the system allocates (assigns) the task to that user. If that user does not belong to one of those profiles, the system offers the task to users in the eligible profiles as usual.

> **Note**
>
> This feature is not available for legacy user tasks.

Service

TIBCO EBX® includes the following built-in services:

- Access a dataspace
- Access data (default service)

- Access the dataspace merge view
- Compare contents
- Create a new record
- Duplicate a record.
- Export data to a CSV file
- Export data to an XML file
- Import data from a CSV file
- Import data from an XML file
- Merge a dataspace
- Validate a dataspace, a snapshot or a dataset

See also *EBX® built-in services [p 427]*

Configuration

Main options > Enable reject

By default, only the *accept* action is offered to the user when saving a decision.

It is possible to also allow users to reject the work item by setting this field to 'Yes'.

Main options > Enable confirmation request

By default, a confirmation request is displayed after user task execution when the user saves the decision by clicking the 'Accept' or 'Reject' button.

To disable this confirmation prompt, set this field to 'Yes'.

Main options > Enable comments

By default, comments are enabled. When a work item is open, a 'Comments' button is displayed and allows the user to enter a comment.

It is possible to hide this 'Comments' button by setting this property to *No*.

Main options > Comments required

By default, it is optional to submit a comment associated with a work item.

It is possible to require the user to enter a comment before saving the decision by setting this field to the desired validation criteria. Comments can be set to be always required, required only if the work item has been accepted, or required only if the work item has been rejected.

Main options > Customized labels

When the user task is run, the user can accept or reject the work item by clicking the corresponding button. In the workflow model, it is possible for a user task to define a customized label and confirmation message for these two buttons. This can be used to adapt the buttons to a more specific context.

[Legacy mode] Termination > Task termination criteria

A single user task could be assigned to multiple *participants* and thus generate multiple work items during workflow execution. When defining a user task in the workflow model, you can select one of the predefined methods for determining whether the user task is finished, based on the statuses of its component work items. When the user task's exit requirement has been satisfied, the data workflow will move on to the next step defined in its model.

For example, for the case of a user task where a product record needs to be approved, you could designate three potential participants. The task termination criteria can specify whether the product record needs to be approved by all three users, or only the first user to respond.

The default user task termination criteria is 'When all work items have been accepted.'

> **Note**
>
> If you specify a service extension overriding the method `UserTask.handleWorkItemCompletion` to handle the determination of the user task's completion, it is up to the developer of the extension to verify from within their code that the task termination criteria defined through the user interface has been met. See `UserTask.handleWorkItemCompletion`[API] in the JavaDoc for more information

[Legacy mode] Termination > Reject tolerance

By default, if a user rejects a work item during workflow execution, the user task is placed into an error state and the workflow progress is halted. When the user task is in an error state, a workflow administrator must intervene by replaying the step where the error occurred in order to continue the workflow execution.

In order to change this default behavior, it is possible to define a certain number of work item rejections to tolerate. While within the limit of tolerated rejections, no error will occur and it is the task termination criteria that determines when to end the user task.

The following task termination criteria automatically tolerate all rejections:

- 'When all work items have been either accepted or rejected'
- 'Either when all work items have been accepted, or as soon as one work item has been rejected'

Extension

A custom class can be specified in order for the task to behave dynamically in the context of a given data workflow. For example, this can be used to create work items or complete user tasks differently than the default behavior.

The specified rule is a JavaBean that must extend the `UserTask`[API] class.

> **Attention**
>
> If a rule is specified and the `handleWorkItemCompletion` method is overridden, the completion strategy is no longer automatically checked. The developer must check for completion within this method.

Notification

A notification email can be sent to users when specific events occur. For each event, you can specify a content template.

It is possible to define a monitor profile that will receive all emails that are sent in relation to the user task.

See also *Generic message templates [p 195]*

Reminder

Reminder emails for outstanding offered or allocated work items can be periodically sent to the concerned users. The recipients of the reminder are the users to whom the work item is offered or allocated, as well as the recipients on copy.

The content of the reminder emails is determined by the current state of the work item. That is, if the work item is offered, the notification will use the "Offered work items" template; if the work item is allocated, the notification will use the "Allocated work items" template.

Deadline

Each user task can have a completion deadline. If this date passes and associated works items are not completed, a notification email is sent to the concerned users. This same notification email will then be sent daily until the task is completed.

There are two deadline types:

- *Absolute deadline*: A calendar date.
- *Relative deadline*: A duration in hours, days or months. The duration is evaluated based on the reference date, which is the beginning of the user task or the beginning of the workflow.

32.4 Script tasks

Script tasks are automatic tasks that are performed without human user involvement.

Three types of script tasks exist, which, once defined, can be used in workflow model steps:

Library	EBX® includes a number of built-in library script tasks, which can be used as-is. Any additional library script tasks must be declared in a `module.xml` file. A library script task must define its label, description and parameters. When a user selects a library script task for a step in a workflow model, its associated parameters are displayed dynamically.
Specific	Specifies a Java class that performs custom actions. The associated class must belong to the same module as the workflow model. Its labels and descriptions are not displayed dynamically to users in workflow models.
Script	Specifies a script based on DSL (Domain Specific Language) to define custom operations. The EBX® IDE is used to facilitate scripts writing. A script is defined by its name, label, description and parameters. When a user selects the name of the script for a step in a workflow model, its associated parameters are displayed dynamically.

Packaging TIBCO EBX® modules [p 879]

Library

EBX® includes the following built-in library script tasks:

- Create a dataspace
- Create a snapshot
- Merge a dataspace
- Import an archive
- Close a dataspace
- Set a data context variable
- Send an email
- Delete records (Note: this script can remove several records)

Library script tasks are classes that extend the class `ScriptTaskBean`[API]. Besides the built-in library script tasks, additional library script tasks can be defined for use in workflow models. Their labels and descriptions can be localized.

The method `ScriptTaskBean.executeScript`[API] is called when the data workflow reaches the corresponding step.

Attention

The method `ScriptTaskBean.executeScript`[API] must not create any threads because the data workflow moves on as soon as the method is executed. Each operation in this method must therefore be synchronous.

See the example [p 1014].

It is possible to dynamically set variables of the library script task if its implementation follows the Java Bean specification. Variables must be declared as parameters of the bean of the library script task in `module.xml`. The workflow data context is not accessible from a Java bean.

> **Note**
>
> Some built-in library script tasks are marked as "deprecated" because they are not compatible with internationalization. It is recommended to use the new script tasks that are compatible with internationalization.

Specific

Specific script tasks are classes that extend the class Sample of ScriptTask [p 1014].

The method `ScriptTask.executeScript`[API] is called when the data workflow reaches the corresponding step.

Attention

The method `ScriptTask.executeScript`[API] must not create any threads because the data workflow moves on as soon as the method is executed. Each operation in this method must therefore be synchronous.

See the example [p 1014].

It is not possible to dynamically set the variables of the bean for specific script tasks. However, the workflow data context is accessible from the Java bean.

Script

EBX® Script is procedural language based on DSL (Domain Specific Language). It proposes a large library which can be used to define custom tasks.

The method executeScriptTask [p 1346] of the script is called when the data workflow reaches the corresponding step. See the examples [p 1346].

32.5 Conditions

Conditions are decision steps in workflows.

Two types of conditions exist, which, once defined, can be used in workflow model steps:

Library condition	EBX® includes a number of built-in library conditions, which can be used as-is. Any additional library script tasks must be declared in a `module.xml` file. A library condition must define its label, description and parameters. When a user selects a library condition for a step in a workflow model, its associated parameters are displayed dynamically.
Specific condition	Specifies a Java class that implements a custom condition. The associated class must belong to the same module as the workflow model. Its labels and descriptions are not displayed dynamically to users in workflow models.

Packaging TIBCO EBX® modules [p 879]

Library conditions

EBX® includes the following built-in library conditions:

- Dataspace modified?
- Data valid?
- Last user task accepted?
- Value null or empty?
- Values equals?

Library conditions are classes that extend the class `ConditionBean`[API]. Besides the built-in library conditions, additional library conditions can be defined for use in workflow models. Their labels and descriptions can be localized.

See the example [p 1017].

It is possible to dynamically set variables of the library condition if its implementation follows the Java Bean specification. Variables must be declared as parameters of the bean of the library condition in `module.xml`. The workflow data context is not accessible from a Java bean.

Specific conditions

Specific conditions are classes that extend the class `Condition`[API].

See the example [p 1016].

It is not possible to dynamically set the variables of the bean for specific conditions. However, the workflow data context is accessible from the Java bean.

32.6 **Sub-workflow invocations**

Sub-workflow invocation steps in workflow models put the current workflow into a waiting state and invoke one or more workflows.

It is possible to include another workflow model definition in the current workflow by invoking it alone in a sub-workflow invocation step.

If multiple sub-workflows are invoked by a single step, they are run concurrently, in parallel. All sub-workflows must be terminated before the original workflow continues onto the next step. The label and description of each sub-workflow can be localized.

The property « Foreground sub-workflow » is useful to precise the progress strategy when several sub-workflows are launched. If the previous step progress strategy is « directly open the next step », this property defines which sub-workflow should be considered in the foreground (only one sub-workflow can hold this behavior). Only the work items which have been generated in this foreground sub-workflow will be able to be opened automatically without passing by the inbox. If only one sub-workflow is defined, this property is ignored (the single sub-workflow is automatically considered in the foreground). For further information, refer to the progress strategy: Progress strategy of the next step [p 200]

Two types of sub-workflow invocations exist:

Static	Defines one or more sub-workflows to be invoked each time the step is reached in a data workflow. For each sub-workflow, it is possible to set its localized labels and descriptions, as well as the input and output variable mappings in its data context. This mode is useful when the sub-workflows to be launched and the output mappings are predetermined.
Dynamic	Specifies a Java class that implements a custom sub-workflow invocation. All workflows that could be potentially invoked as sub-workflows by the code must be declared as dependencies. The workflow data context is directly accessible from the Java bean. Dynamic sub-workflow invocations must be declared in a `module.xml` file. This mode is useful when the launch of sub-workflows is conditional (for example, if it depends on a data context variable), or when the output mapping depends on the execution of the sub-workflows.

32.7 **Wait tasks**

Wait task steps in workflow models put the current workflow into a waiting state until a specific event is received.

When a wait task is reached, the workflow engine generates a unique resume identifier associated with the wait task. This identifier will be required to resume the wait task, and as a consequence the associated workflow.

A wait task specifies which profile is authorized to resume the wait task; and a Java class that implements a wait task bean: `WaitTaskBean`[API].

The workflow data context is directly accessible from the Java bean.

Wait task beans must be declared in a `module.xml` file.

First, the wait task bean is called when the workflow starts waiting. At this time, the generated resume identifier is available to call a web service for example. Then, the wait task bean is called when the wait task is resumed. In this way, the data context may be updated according to the received parameters.

> **Note**
>
> The built-in administrator always has the right to resume a workflow.

32.8 Editing the workflow diagram

About

A workflow model is displayed in a BPMN-like editable diagram.

This view provides full editing capabilities and can help modelers have a clear view of the workflow model they are designing.

Please also note that, although the diagram is derived from BPMN standards, it is not a strict representation of BPMN since EBX® workflow concepts are slightly different.

Saving the layout

It is possible to save the modified layout. Please note that this is not a user-based save: it will be shared by all the users.

Actions

Export as PNG	Creates a PNG image.
Export as SVG	Creates an SVG image.
Export as PDF	Creates a PDF document from the workflow model. Several properties are available to custom your export. You can configure the orientation of the pdf pages (Landscape or Portrait), the size (A3, Letter, etc ...). There is also two properties that might provide more details to your model: • Display identifiers (default value is true): Same behavior with "Show steps identifiers" in the workflow diagram toolbar but restricted to the pdf export. It will show the step identifiers for each step. • Add index (default value is true): At the end of the pdf, an index recording all the steps of the diagram will be added. It might be useful in the case where several steps title are too long to be fully displayed. *For this property, we recommend showing the steps identifiers as it will help in recognizing which index line corespond to which step.*

View

Layout > Default layout	Applies the default layout to the diagram.
Display > Show/Hide grid	Shows the grid if the grid is not visible, hides it otherwise.
Display > Zoom In	A zoom in is executed on the diagram. To go faster you can use "Ctrl, Up mouse wheel" with the mouse pointing the diagram boundary.
Display > Zoom Out	A zoom out is executed on the diagram. To go faster you can use "Ctrl, Down mouse wheel" with the mouse pointing the diagram boundary.
Display > Show/Hide steps identifiers	Show/hide a badge displaying the identifier of a workflow step.
Plan view > Hierarchy	Shows the hierarchical view of a given workflow model if enabled.

Buttons

Save layout	Saves the current layout.
Save layout and close	Saves the current layout and closes the service.
Revert	Reverts changes and reloads a previously saved layout.
Close	Closes the service.

Edit

Create	Hovering over a link or selecting it, makes '+' appear. It is used to create a new step.
Delete	DEL Pressing 'DEL' key will remove the selected node. Please note that this action cannot be undone.
Node toolbar	Hovering on a node or directly selecting it makes a toolbar appear. It is used to either: • Edit a step. • Remove a step. • Duplicate a step. • Relink to an existing step. • Show or hide in progress view.
Edit a node	Double click on a node in order to quick edit a step.

Features

The diagram view offers useful additional features

Undo last action	CTRL + Z Please note that it cannot be done after the following actions: • Create/Remove/Edit/Duplicate a step • Relink • Show/Hide in progress view • Revert
Zoom in/Zoom out.	Mouse middle button then mouse wheel / CTRL then mouse wheel. Also see View [p 210]
Multiple selection	Click on the nodes or links selected holding down the CTRL button / Draw a selection rectangle (you will need to hold down the left click for 1 second before drawing the area).
Customizing links drawing	When clicking on a link, you can either move the segments by dragging the squares which appear on the corners, or separate a specific segment by moving the circle in the middle.
Edit a step	Double clicking on a step will display an edition form.
Overview	A panel is now available with a miniature workflow diagram view which can be used to navigate within it. This panel can be collapsed, expanded and dragged inside the area allocated for the workflow diagram view.

CHAPTER 33

Configuring the workflow model

This chapter contains the following topics:

1. Information associated with a workflow model
2. Workflow model properties
3. Data context
4. Custom workflow execution views
5. Permissions on associated data workflows
6. Workflow model snapshots
7. Deleting a workflow model

33.1 Information associated with a workflow model

To view and edit the owner and documentation of your workflow model, select 'Information' from the workflow model 'Actions' [p 195] menu for your workflow model in the navigation pane.

Owner	Specifies the workflow model owner, who will have the rights to edit the workflow model's information and define its permissions.
Localized documentation	Localized labels and descriptions for the workflow model.
Activated	*This property is deprecated.* Whether the workflow model is activated. A workflow model must be activated in order to be able to be published.

33.2 Workflow model properties

Configuration for a workflow model is accessible in the navigation pane under 'Workflow model configuration'.

Module name	Module containing specific Java resources (user task extensions, specific scripts and conditions).
Notification of start	The list of profiles to which to send notifications, based on a template, when a data workflow is launched. See Generic message templates [p 195].
Notification of completion	The list of profiles to which to send notifications, based on a template, when a data workflow is completed. The notification is only sent if the workflow has been completed under normal circumstances, that is, not due to an administration action. See Generic message templates [p 195].
Notification of error	The list of profiles that will receive notifications, based on a template, when a data workflow is in error state. See Generic message templates [p 195].
Priority	By default, each workflow associated with this model will be launched with this priority. Setting a priority is optional. If no priority is defined here, and a default priority is set for the repository, the repository default priority will be used for any associated workflow with no priority assigned. See Work item priorities [p 235] for more information. **Note:** Only users who are defined as workflow administrators will be able to manually modify the priority level of any associated data workflows.
Activate quick launch	By default, when a workflow is launched, the user is prompted to enter a documentation for the new workflow in an intermediate form. This documentation is optional. Setting the 'Activate quick launch' property to 'Yes' allows skipping this documentation step and proceeding directly to the workflow launch.
Automatically open the first step	Allows determining the navigation after a workflow is launched. By default, once a workflow is launched, the current table (workflow launchers or monitoring > publications) is automatically displayed.

	Enabling this property will allow the workflow user to keep working on the launched workflow. If, after the first workflow step is executed, a work item is reached, and this work item can be started by the workflow creator, then the work item is automatically opened (if several work items are reached, the first created is opened). This will save the user from selecting the corresponding work item from the work items inbox. If no work item has been reached, the next step progress strategy is evaluated. If no work item has been opened, the table from which the workflow has been launched is displayed. **Limitation:** This property will be ignored if the first step is a sub-workflow invocation.
Workflow trigger	Component that intercepts the main events of a workflow. This bean must be declared in a `module.xml` file. See the example [p 1020].
Permissions	Permissions on actions related to the data workflows associated with the workflow model. This bean must be declared in a `module.xml` file. See the example [p 1019].
Programmatic action permissions	Defines a custom component that handles the permissions of the workflow. If set, this overrides all permissions defined in the property 'Permissions'.

33.3 Data context

The data context configuration can be accessed from the navigation pane.

Each workflow has its own data context, thus allowing to have its own local dataspace during its execution. This gives the possibility to store and to vary values that will direct the workflow execution.

The data context is defined by a list of variables. Each variable has the following properties:

Name	Identifier of the variable.
Default value	If defined, the variable will be initialized with this default value.
Input parameter	'Yes' must be checked in order to define this variable as an input parameter.
Output parameter	'Yes' must be checked in order to define this variable as an output parameter. Else, this variable will not be displayed in the list of output parameters, in the task definition interface.

33.4 Custom workflow execution views

The workflow execution views customization can be accessed from the navigation pane.

The customization allows configuring the specific columns of the work items and workflow views (inbox, work items monitoring, active workflows monitoring and completed workflows) so that the user can have custom values for his specific needs. Those values are dynamic thanks to the evaluation of variables of the data context if any.

33.5 Permissions on associated data workflows

Workflow administration	Defines the profile that is allowed to perform administration actions on the workflows. The administration actions include the following: replay a step, resume a workflow, terminate a workflow, disable a publication and unpublish. In order to perform these actions, this profile is automatically granted the "Visualize workflows" permission. The built-in administrator always has the workflow administration rights.
Workflow administration > Replay a step	Defines the profile that is allowed to replay a workflow step. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Active workflows" section is available to replay a step. A profile with the "Workflow administration" permission is automatically allowed to perform this specific action. The built-in administrator always has the rights to replay a step.
Workflow administration > Terminate workflow	Defines the profile that is allowed to terminate and clean a workflow. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Active workflows" section is available to terminate and clean an active workflow. A button in the "Completed workflows" section is available to delete a completed workflow. A profile with the "Workflow administration" permission is automatically allowed to perform this specific action. The built-in administrator always has the rights to terminate a workflow.
Workflow administration > Force a workflow to resume	Defines the profile that is allowed to force resuming a waiting workflow. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Active workflows" section is available to resume a workflow. A profile with the "Workflow administration" permission is automatically allowed to perform this specific action. The built-in administrator always has the right to resume a workflow.
Workflow administration > Disable a publication	Defines the profile that is allowed to disable a workflow publication. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Publications" section is available to disable a publication. It is only

	displayed on active publications. A profile with the "Workflow administration" permission is automatically allowed to perform this specific action. The built-in administrator always has the rights to disable a publication.
Workflow administration > Unpublish	Defines the profile that is allowed to unpublish a workflow publication. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Publications" section is available to unpublish disabled publications only. A profile with the "Workflow administration" permission is automatically allowed to perform this specific action. The built-in administrator always has the unpublish rights.
Allocation management	Defines the profile that is allowed to manage work items allocation. The allocation actions include the following: allocate work items, reallocate work items and deallocate work items. In order to perform these actions, this profile is automatically granted the "Visualize workflows" permission. The built-in administrator always has the allocation management rights.
Allocation management > Allocate work items	Defines the profile that is allowed to allocate work items. In order to perform these actions, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Work items" section is available to allocate a work item. It is only displayed on offered work items. A profile with the "Allocation management" permission is automatically allowed to perform this specific action. The built-in administrator always has the work items allocation rights.
Allocation management > Reallocate work items	Defines the profile that is allowed to reallocate work items. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Work items" section is available to reallocate a work item. It is only displayed on allocated work items. A profile with the "Allocation management" permission is automatically allowed to perform this specific action. The built-in administrator always has the work items reallocation rights.
Allocation management > Deallocate work items	Defines the profile that is allowed to deallocate work items. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Work items" section is available to deallocate a work item. It is only displayed on allocated work items. A profile with the "Allocation management" permission is automatically allowed to perform this specific

	action. The built-in administrator always has the work items deallocation rights.
Launch workflows	Defines the profile that is allowed to manually launch new workflows. This permission allows launching workflows from the active publications of the "Workflow launchers" section. The built-in administrator always has the launch workflows rights. Permissions to launch a workflow are always automatically verified, even by API. It is possible to bypass these automatic permission checks and manage specific authorization rules manually. For this, set the ebx.properties permissionCheckDelegated [p 596] to true and use the dedicated API `WorkflowEngine.isUserAuthorizedToLaunch`[API] to delegate permission evaluation to the developer.
Visualize workflows	Defines the profile that is allowed to visualize workflows. By default, the end-user can only see work items that have been offered or allocated to him in the "Inbox" section. This permission also allows visualizing the publications, workflows and work items associated with this workflow model in the "Monitoring" and "Completed workflows" sections. This profile is automatically granted the "Visualize completed workflows" permission. The built-in administrator always has the visualize workflows rights.
Visualize workflows > The workflow creator can visualize it	If enabled, the workflow creator has the permission to view the workflows he has launched. This restricted permission grants access to the workflows he launched and to the associated work items in the "Monitoring > Active workflows", "Monitoring > Work items" and "Completed workflows" sections. The default value is 'No'.
Visualize workflows > Visualize completed workflows	Defines the profile that is allowed to visualize completed workflows. This permission allows visualizing completed workflows in the "Completed workflows" section and accessing their history. A profile with the "Visualize workflows" permission is automatically allowed to perform this action. The built-in administrator always has the visualize completed workflows rights.

> **Note**
>
> A user who has no specific privileges assigned can only see work items associated with this workflow that are offered or allocated to that user.

See also *Workflow administration* *[p 239]*

33.6 Workflow model snapshots

The history of workflow model snapshots can be managed from **Actions > View publications history**.

The history table displays all snapshots which contain the selected workflow model and indicates if a workflow model is published. For each snapshot, the **Actions** button allows you to export or view the corresponding workflow model.

33.7 Deleting a workflow model

Workflow model can be deleted, however any associated publications remain accessible in the Data Workflows area. If a new workflow model is created with the same name as a deleted workflow model, publishing will prompt to replace the old publication.

See also *Publishing workflow models* *[p 221]*

CHAPTER 34

Publishing workflow models

This chapter contains the following topics:

1. About workflow publications
2. Publishing and workflow model snapshots
3. Sub-workflows in publications

34.1 About workflow publications

Once a workflow model is defined, it must be published in order to enable authorized users to launch associated data workflows.

Two services are available in order to publish :

- With the button **Publish** in the navigation pane: this service allows to publish only the model currently edited.
- Through the button **Mass publish** that can be found in the toolbar on the same level of the section title **Workflow models** : this service allows many models to be published at once. The first step allows to select many models to publish: only models not containing sub-workflows invocation will appear in this step. Indeed, a workflow having sub-workflows doesn't qualify for mass publish.

Workflow models can be published several times. A publication is identified by its publication name

34.2 Publishing and workflow model snapshots

When publishing a workflow model, a snapshot is taken of its current state. A label and a description can be specified for the snapshot to be created. The default snapshot label is the date and time of the publication. The default description indicates the user who published the workflow model.

For each workflow model being published, the specified publication name must be unique. If a workflow model has already been published, it is possible to update an existing publication by reusing the same publication name. The names of existing workflow publications associated with a given workflow model are available in a drop-down menu. In the case of a publication update, the old version is no longer available for launching data workflows, however it will be used to terminate existing workflows. The content of different versions can be viewed in the workflow model snapshot history.

See also *Workflow model snapshots* *[p 220]*

34.3 Sub-workflows in publications

When publishing a workflow model containing sub-workflow invocation steps, it is not necessary to separately publish the models of the sub-workflows. From an administration standpoint, the model of the main workflow (the one currently published by a user) and the models of the sub-workflows are published as a single entity.

The system computes the dependencies to workflow models used as sub-workflows, and automatically creates one publication for each dependent model. These technical publications are dedicated to the workflow engine to launch sub-workflows, and are not available in the Workflow Data area.

The multiple publication is not available for a workflow model containing sub-workflow invocation steps. This is why the first step of the publication (selection of workflow models to publish) is not offered in this case.

Republishing the main workflow model automatically updates the invoked sub-workflow models.

Although a sub-workflow model can be published separately as a main workflow model, this will not update the version used by an already published main workflow model using this sub-workflow.

Data workflows

CHAPTER 35

Introduction to data workflows

This chapter contains the following topics:

1. Overview

35.1 Overview

A data workflow is an executed step-by-step data management process, defined using a workflow model publication. It allows users, as well as automated procedures, to perform actions collaboratively on a set of data. Once a workflow model has been developed and published, the resulting publication can be used to launch a data workflow to execute the defined steps.

Depending on the workflow user permissions defined by the workflow model, a user may perform one or more of the following actions on associated data workflows:

- As a user with default permissions, work on and complete an assigned work item.
- As a user with workflow launching permissions, create a new data workflow from a workflow model publication.
- As a workflow monitor, follow the progress of ongoing data workflows and consult the history of completed data workflows.
- As a manager of work item allocation, modify work item allocations manually for other users and roles.
- As a workflow administrator, perform various administration actions, such as replaying steps, terminating workflows in progress, or rendering publications unavailable for launching data workflows.

See also

Work items *[p 231]*

Launching and monitoring data workflows *[p 237]*

Administration of data workflows *[p 239]*

Permissions on associated data workflows *[p 217]*

Related concepts *Workflow models* *[p 194]*

CHAPTER 36

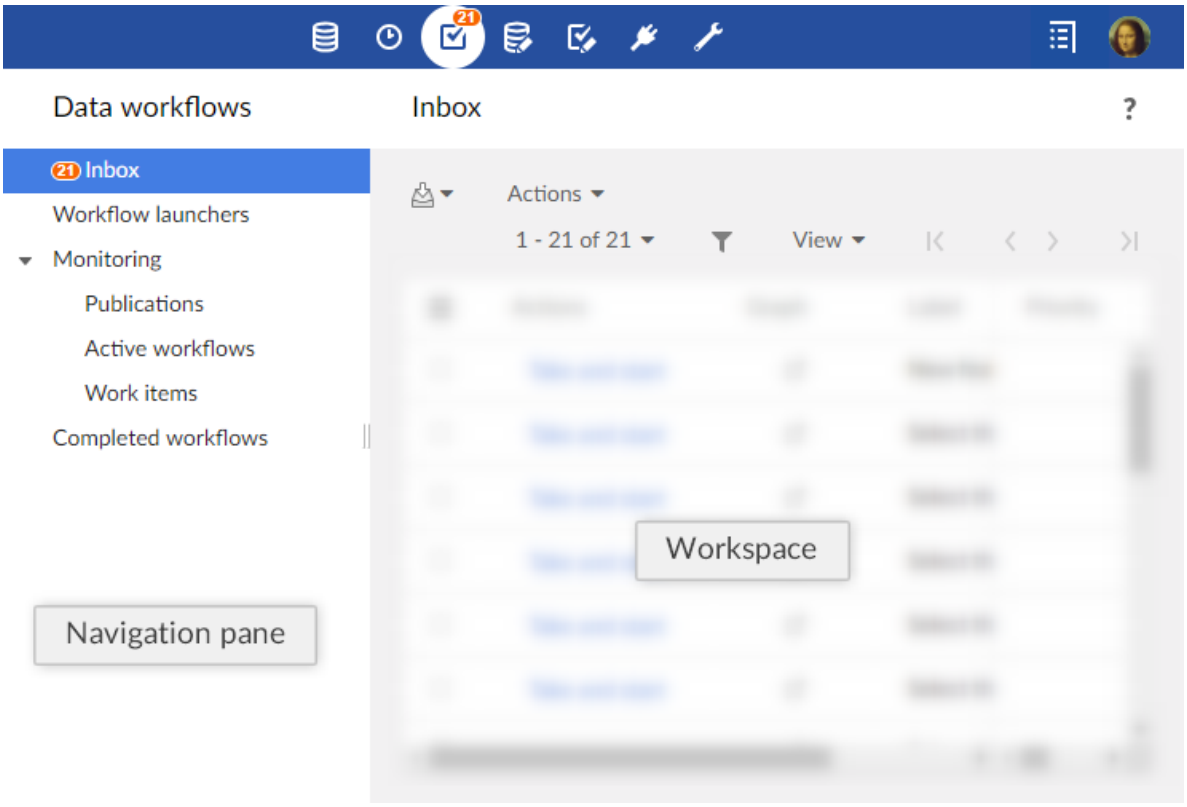
Using the Data Workflows area user interface

This chapter contains the following topics:

1. Navigating within the interface
2. Navigation rules
3. Custom views
4. Specific columns
5. Filtering items in views
6. Workflow progress view

36.1 Navigating within the interface

Data workflow functionality is located in the **Data Workflows** area of the TIBCO EBX® user interface.



> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective' or from a specifically configured perspective. Only authorized users can access these interfaces.

The navigation pane is organized into several entries. These entries are displayed according to their associated global permission. The different entries are:

Work items inbox	All work items either allocated or offered to you, for which you must perform the defined task.
Workflow launchers	List of workflow model publications from which you are allowed to launch data workflows, according to your user permissions.
Monitoring	Monitoring views on the data workflows for which you have the necessary viewing permissions.
Publications	Publications for which you have the necessary viewing permissions. If you have additional administrative permissions, you can also disable the ability to launch data workflows from specific publications from this view.
Active workflows	Data workflows in the process of execution for which you have the necessary viewing permissions. If you have additional administrative permissions, you can also perform actions such as replaying steps of data workflows, and terminating the execution of data workflows from this view.
Work items	Work items for which you have the necessary viewing permissions. If you have additional administrative permissions, you can also perform actions relevant to work item administration, such as allocating work items to specific users or roles from this view.
Completed workflows	Data workflows that have completed their execution, for which you have the necessary viewing permissions. You can view the history of the executions of the data workflows. If you have additional administrative permissions, you can also clean completed workflows from the repository from this view.

Note

Each section can be accessed through Web Components, for example, for portal integration, or programatically using the `ServiceKey` class in the Java API.

See also

Using TIBCO EBX® as a Web Component *[p 419]*

`ServiceKey`[API]

36.2 Navigation rules

Work items inbox

By default, once a work item has been executed, the work items inbox is displayed.

This behavior can be modified according to the next step progress strategy, which can allow to execute several steps in a row without going back to the work items inbox.

See the progress strategy of a workflow step [p 200] in workflow modeling.

Workflow launchers

By default, once a workflow has been launched, the workflow launchers table is displayed.

This behavior can be modified according to the model configuration, which can allow to directly open the first step without displaying the workflow launchers table.

See the automatic opening of the first workflow step [p 214] in workflow modeling.

36.3 Custom views

It is possible to define views on workflow tables and to benefit from all associated mechanisms (publication included).

Permissions to create and manage workflow table views are the same as the permissions for data table views. It may thus be necessary to change the permissions in the 'Administration' section in order to benefit from this feature, by selecting *Workflow management* > *Workflows*.

See the Views [p 133] for more information.

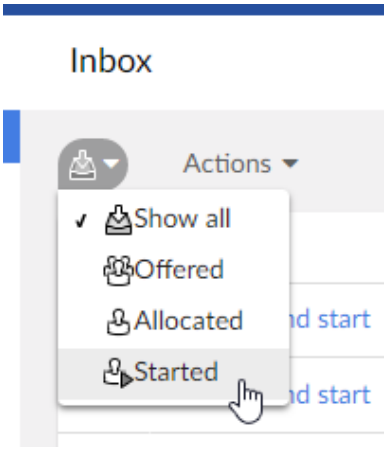
36.4 Specific columns

By default, specific columns are hidden in the views that can benefit from it (inbox, work items monitoring, active workflows monitoring and completed workflows).

A custom view should be created and applied in order to display the specific columns. For each work item or workflow, the matching defined in the associated workflow model is then applied. If an expression is defined for a column and contains data context variables, these variables are evaluated each time the workflow data context is updated.

36.5 Filtering items in views

In certain tables, such as the 'Work item inbox', you can narrow down the entries displayed in the tables by viewing only the entries of a certain state. In such views, there is a menu from which you can select a state to see its corresponding items.



36.6 Workflow progress view

Whether as a user with a work item to perform, or as a data workflow monitor or administrator, you can view the progress or the history of a data workflow execution by clicking the 'Preview' button that appears in the 'Data workflow' column of tables throughout the data workflows user interface. This opens a pop-up displaying an interactive progress view of the data workflow's execution. In this view, you can see the overall progress of the execution, as well as click on an individual step to view the details of its information.

If steps have been defined as hidden in the workflow modeling, they are automatically hidden in the workflow progress view for non-administrator users (non built-in administrators and non workflow administrators). A button is available to display hidden steps. The choice of users (show or hide steps) is saved by user, by publication during the user session.

For user tasks performed using the new mode (single work item), the main information about the single work item is directly displayed in the workflow progress view, when applicable: the avatar of the user associated with the work item, and the decision that has been taken for the work item (accepted or rejected).

CHAPTER 37

Work items

This chapter contains the following topics:

1. About work items
2. Working on work items as a participant
3. Work item priorities

37.1 About work items

A work item is a unit of work that must be performed by a human user as a part of a user task. By default, when a workflow model defines a user task, data workflows that are launched from that model's publications will generate an individual work item for each of the participants listed in the user task.

See also *Overview* *[p 1027]*

Work item states

When the data workflow spawns a work item for a modelled user task during execution, the work item passes through several possible states: offered, allocated, started, and completed.

Creation of work items

Default mode

By default, a single work item is generated regardless of the list of defined profiles.

By default, if a single user is defined in the list of profiles, the created work item is in the *allocated* state.

By default, in other cases, the created work item is in the *offered* state.

> **Note**
>
> The default behavior described above can be overridden by a programmatic extension defined in the user task. In this case, work items may be generated programmatically and not necessarily based on the user task's list of participants.

Legacy mode

By default, for each user defined as a participant of the user task, the data workflow creates a work item in the *allocated* state.

By default, for each role defined as a participant of the user task, the data workflow creates a work item in the *offered* state.

> **Note**
>
> The default behavior described above can be overridden by a programmatic extension defined in the user task. In this case, work items may be generated programmatically and not necessarily based on the user task's list of participants.

Variations of the work item states

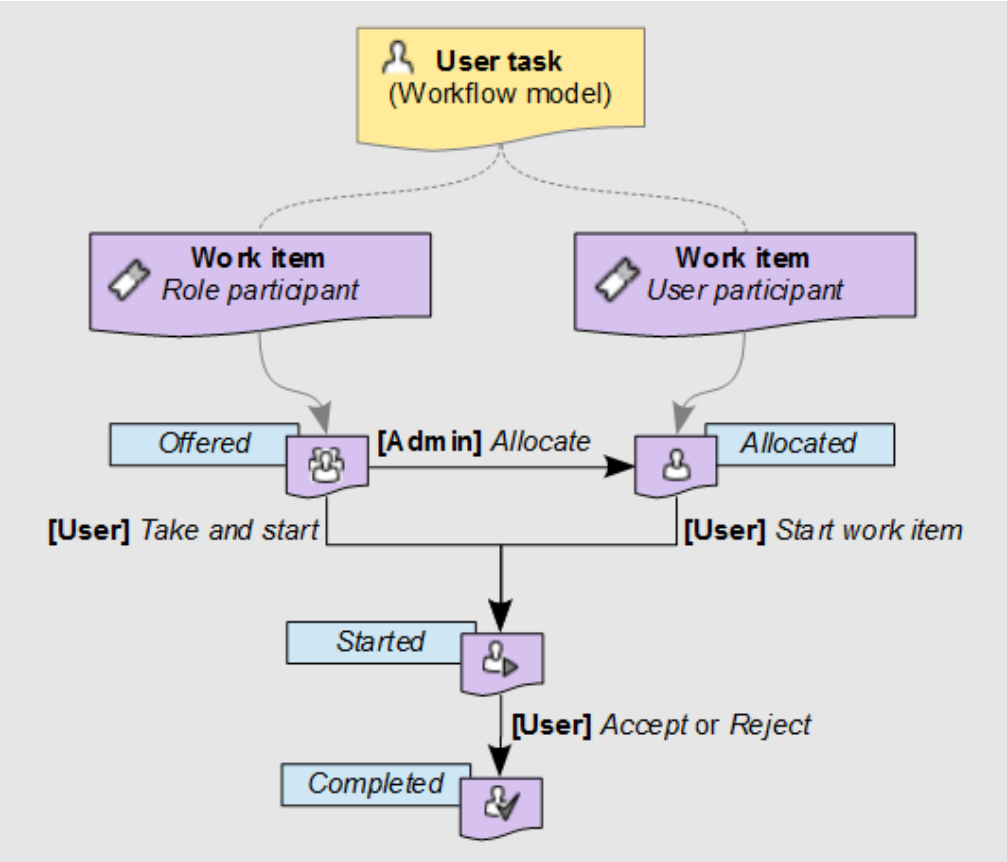
When the work item is in the *allocated* state, the defined user can directly start working on the allocated work item with the 'Take and start' action. The work item's state becomes *started*.

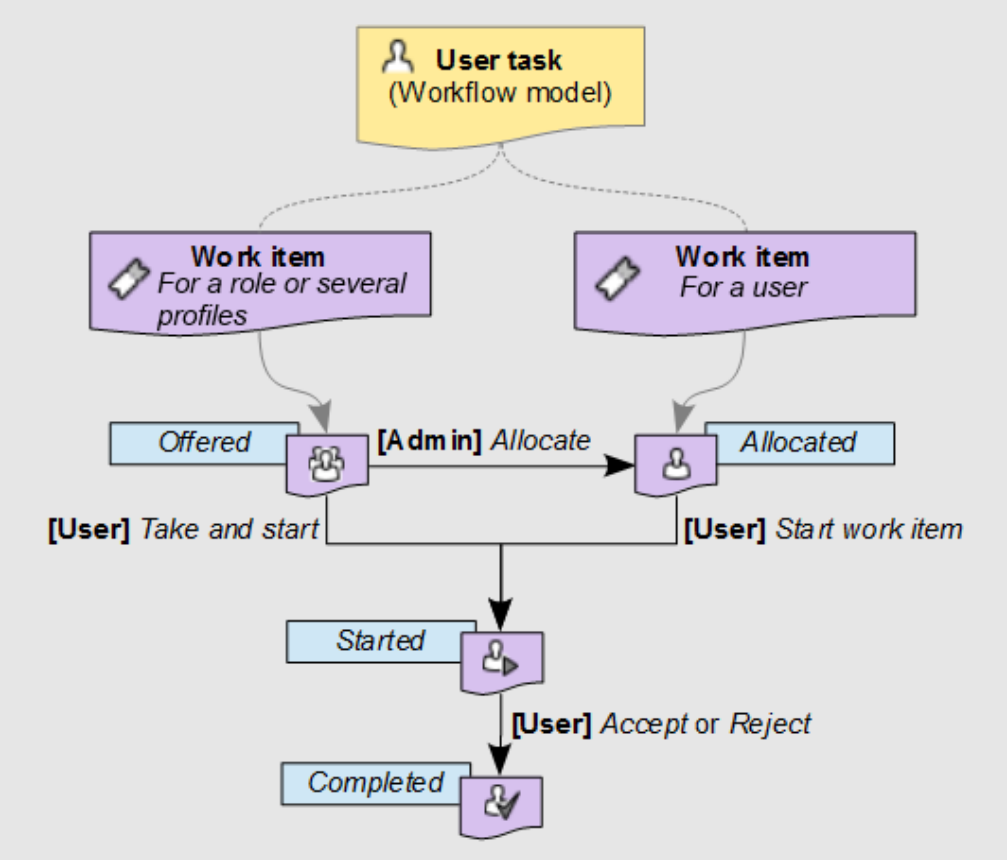
When the work item is in the *offered* state, any user or member of the roles to whom the work item is offered can take the work item with the 'Take and start' action'. The work item's state becomes *started*.

Before a user has claimed the offered work item, a workflow allocation manager can intervene to manually assign the work item to a single user, thus moving the work item to the *allocated* state. Then, when that user begins work on the work item by performing the action 'Start work item', the work item progresses to the *started* state.

Finally, after the user who started the work item has finished the assigned action, the concluding accept or reject action moves the work item to the *completed* state. Once a user completes a work item, the data workflow automatically progresses onto the next step defined in the workflow model.

Diagram of the work item states





37.2 Working on work items as a participant

All work items relevant to you as a user (either offered or allocated to you), appear in your work items inbox. When you begin working on a work item, you can add an associated comment that will be visible to other participants of the data workflow, as well as administrators and monitors of the workflow. As long as you are still working on the work item, you can go back and edit this comment.

After you have performed all the necessary actions assigned for the work item, you must signal its completion by clicking either the **Accept** or **Reject** button. The labels of these two buttons may differ depending on the context of the work item.

To review the current progress of a data workflow for which you have a waiting work item in your work item inbox, click its 'Preview' button in the 'Data workflow' column of the table. A pop-up will show an interactive progress view of the data workflow up until this point and the upcoming steps. You can view the details of a step by clicking on that step.

> **Note**
>
> If you interrupt the current session in the middle of a started work item, for example by closing the browser or by logging out, the current work item state is preserved. When you return to the work item, it continues from the point where you left off.

37.3 Work item priorities

Work items may carry a priority value, which can be useful for sorting and filtering outstanding work items. The priority of a work item is set at the level of its data workflow, rather than being specific to the individual work item itself. Thus, if a data workflow is considered urgent, all its associated open work items are also considered to be urgent. By default, there are six priority levels ranging from 'Very low' to 'Urgent', however the visual representation and naming of the available priority levels depend on the configuration of your TIBCO EBX® repository.

See also *user task (glossary)* *[p 40]*

Related concepts *User tasks* *[p 200]*

CHAPTER 38

Launching and monitoring data workflows

This chapter contains the following topics:

1. Launching data workflows
2. Monitoring activities
3. Managing work item allocation

38.1 Launching data workflows

If a workflow model has given you the permissions to launch data workflows from its publications, you can create new data workflows from the 'Workflow launchers' entry in the navigation pane. To create a new data workflow from a workflow model publication, click the **Launch** button in the entry of the publication.

You can optionally define localized labels and descriptions for the new data workflow you are launching.

38.2 Monitoring activities

If a workflow model's permissions have configured your user or role for workflow monitoring, you have the ability to follow the progress of data workflows that are currently executing. You can access your monitoring views from the 'Monitoring' section of the navigation panel. If you have additional workflow management permissions, you can also perform the associated actions from these views.

Once the data workflows that you monitor have completed execution, they appear under 'Completed data workflows', where you can consult their execution history.

38.3 Managing work item allocation

If a workflow model defines special allocation management permissions for you or a role that you belong to, you have the ability to manually intervene for work item allocations during the execution of associated data workflows. In this case, you are able to perform one or more of the actions listed below on work items.

Select 'Work items' in the 'Monitoring' section of the navigation pane. The actions that you are able to perform appear in the **Actions** menu of the work item's entry in the table, depending on the current state of the work item.

Allocate	Allocate a work item to a specific user. This action is available for work items in the *offered* state.
Deallocate	Reset a work item in the *allocated* state to the *offered* state.
Reallocate	Modify the user to whom a work item is allocated. This action is available for work items in the *allocated* state.

See also

Work items *[p 231]*

Permissions on associated data workflows *[p 217]*

Related concepts *Workflow models* *[p 194]*

CHAPTER 39

Administration of data workflows

If you have been given permissions for administration activities associated with data workflows, any relevant publications, active data workflows, and work items, will appear under the entries of the 'Monitoring' section in the navigation panel. From these monitoring views, you can directly perform administrative tasks from the **Actions** menus of the table entries.

> **Note**
>
> When a workflow model gives you administrative rights, you automatically have monitoring permissions on all of the relevant aspects of data workflow execution, such as publications, active data workflows, and work items.

This chapter contains the following topics:

1. Overview of data workflow execution
2. Data workflow administration actions

39.1 Overview of data workflow execution

When a data workflow is launched, a *token* that marks the step currently being executed is created and positioned at the start of the workflow. As each step is completed, this token moves on to the next step as defined in the workflow model on whose publication the data workflow is based.

At any given point during the execution of a data workflow, the token is positioned on one of the following:

- a script task, which is run automatically and requires no user interaction. The script task is completed when the defined actions finish running.
- a user task, which spawns one or more work items to be performed manually by users. Each work item is completed by an explicit 'Accept' or 'Reject' action from a user, and the completion of the umbrella user task is determined according to the task termination criteria defined for the user task in the workflow model.
- a condition, which is evaluated automatically in order to determine the next step in the execution of the data workflow.
- a sub-workflows invocation, which launches associated sub-workflows and waits for the termination of the launched sub-workflows.
- a wait task, which pauses the workflow until a specific event is received.

The token can be in the following states:

- **To execute:** The token is the process of progressing to the next step, based on the workflow model.
- **Executing:** The token is positioned on a script task or a condition that is being processed.
- **User:** The token is positioned on a user task and is awaiting a user action.
- **Waiting for sub-workflows:** The token is positioned on a sub-workflow invocation and is awaiting the termination of all launched sub-workflows.
- **Waiting for event:** The token is positioned on a wait task and is waiting for a specific event to be received.
- **Finished:** The token has reached the end of the data workflow.
- **Error:** An error has occurred.

See also *Workflow management* *[p 719]*

39.2 Data workflow administration actions

Actions on publications

Disabling a workflow publication

To prevent new data workflows from being launched from a given workflow publication, you can disable it. Select the 'Publications' entry from the navigation pane, then select **Actions > Disable** in the entry for the publication you want to disable.

Once disabled, the publication will no longer appear in the 'Workflow launchers' view of users, but any data workflows already launched that are in progress will continue executing.

> **Note**
>
> Once a publication has been disabled, it cannot be re-enabled from the Data Workflows area. Only a user with the built-in repository 'Administrator' role can re-enable a disabled publication from the Administration area, although manually editing technical tables is not generally recommended, as it is important to ensure the integrity of workflow operations.

Unpublishing a workflow publication

If a workflow publication is no longer required, you can remove it completely from the views in the Data Workflows area by unpublishing it. To do so,

1. Disable the workflow publication to prevent users from continuing to launch new data workflows from it, as described in Disabling a workflow publication [p 240].
2. Unpublish the workflow publication by selecting **Actions > Unpublish** from the workflow publication's entry in the same table of publications.

> **Note**
>
> When you choose to unpublish a workflow publication, you will be prompted to confirm the termination and cleaning of any data workflows in progress that were launched from this workflow publication, and any associated work items. Any data that is lost as a result of forcefully terminating a data workflow cannot be recovered.

Actions on data workflows

From the tables of data workflows, it is possible to perform actions from the **Actions** menu in the record of a given data workflow.

Replaying a step

In the event of an unexpected failure during a step, for example, an access rights issue or unavailable resources, you can "replay" the step as a data workflow administrator. Replaying a step cleans the associated execution environment, including any related work items and sub-workflows, and resets the token to the beginning of the current step.

To replay the current step in a data workflow, select **Actions > Replay the step** from the entry of the workflow in the 'Active workflows' table.

Terminating and cleaning an active data workflow

In order to stop and clean a data workflow that is currently in progress, select **Actions > Terminate and clean** from the entry of the workflow in the 'Active workflows' table. This will stop the execution of the data workflow and clean the data workflow and all associated work items and sub-workflows.

> **Note**
>
> This action is not available on workflows in the 'Executing' state, and on sub-workflows launched from another workflow.

> **Note**
>
> Workflow history data is not deleted.

Forcing termination of an active data workflow

In order to stop a data workflow that is currently in progress, select **Actions > Force termination** from the entry of the workflow in the 'Active workflows' table. This will stop the execution of the data workflow and clean any associated work items and sub-workflows.

> **Note**
>
> This action is available for sub-workflows, and for workflows in error blocked on the last step.

> **Note**
>
> Workflow history data is not deleted.

Forcing resumption of a waiting data workflow

In order to resume a data workflow that is currently waiting for an event, select *Actions > Force resumption* from the entry of the workflow in the 'Active workflows' table. This will resume the data workflow. Before doing this action, it is the responsibility of the administrator to update the data context in order to make sure that the data workflow can execute the next steps.

> **Note**
>
> This action is only available for workflows in the 'waiting for event' state.

Cleaning a completed data workflow

When a data workflow has completed its execution, its history is viewable by monitors and administrators of that workflow in the view 'Completed workflows'. To remove the completed workflow, you can perform a clean operation on it. To do so, select **Actions > Clean** from the entry of the workflow in the 'Completed workflows' table.

When cleaned a workflow is no longer visible in the view 'Completed workflows' but its history is still available from the technical administration area.

> **Note**
>
> This action is not available on sub-workflows launched from another workflow.

See also *Workflow management* *[p 719]*

Modifying the priority of a data workflow

After a data workflow has been launched, an administrator of the workflow can alter its priority level. Doing so changes the priority of all existing and future work items created by the data workflow. To change the priority level of a data workflow, select **Actions > Modify priority** from the entry of the workflow in the 'Active workflows' table.

See also *Permissions on associated data workflows* *[p 217]*

Data services

CHAPTER 40

Introduction to data services

This chapter contains the following topics:

1. Overview
2. Using the Data Services area user interface

40.1 Overview

What is a data service?

A data service [p 41] is:

- a standard Web service that interacts with TIBCO EBX®.

  SOAP data services can be dynamically generated based on data models from the 'Data Services' area.

- a RESTful service that allows interaction with EBX®.

  The built-in REST data services have an OpenAPI interface, they are also self-descriptive through the returned metadata.

They can be used to access some of the features available through the user interface.

See also

WSDL/SOAP [p 1060]

REST [p 1114]

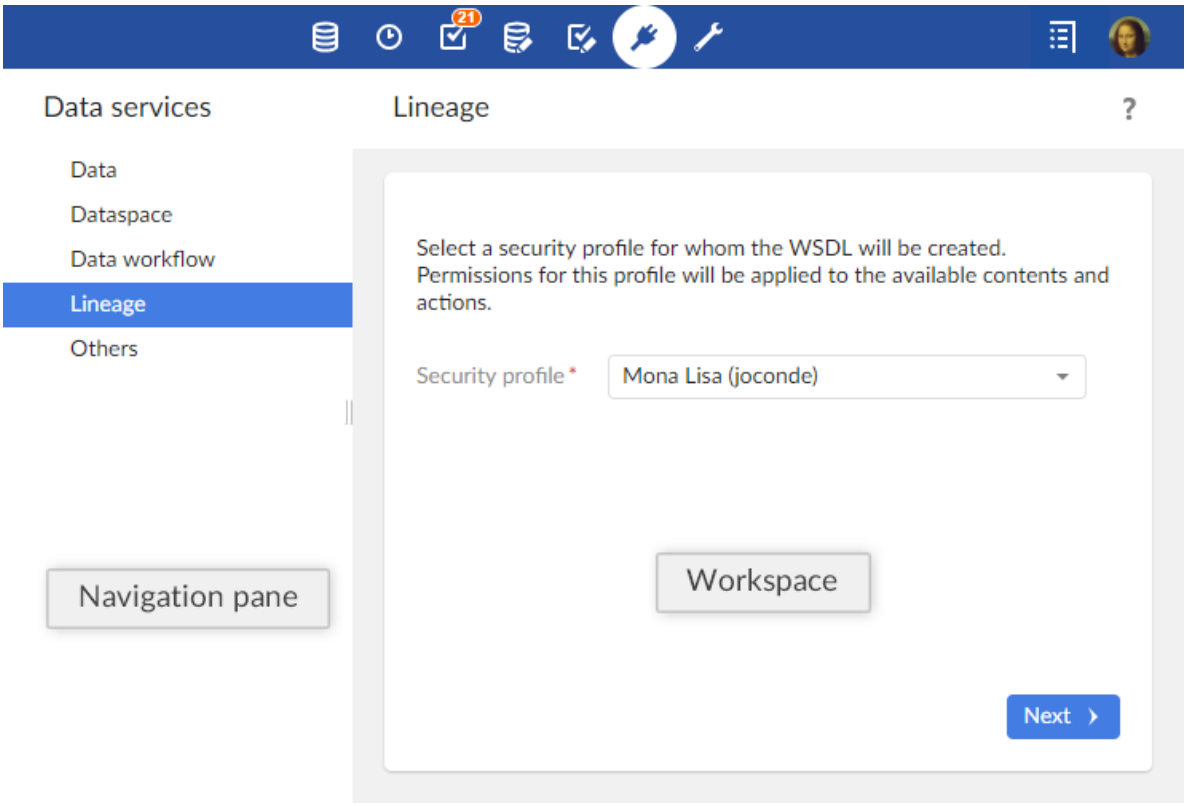
Lineage

Lineage [p 42] is used to establish user permission profiles for non-human users, namely data services. When accessing data using WSDL interfaces, data services use the permission profiles established through lineage.

Glossary

See also *Data services [p 41]*

40.2 Using the Data Services area user interface



> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective'.

Related concepts

Dataspace [p 104]

Dataset [p 122]

Data workflows [p 224]

Introduction [p 1060]

CHAPTER 41

Generating data service WSDLs

This chapter contains the following topics:

1. Generating a WSDL for operations on data
2. Generating a WSDL for dataspace operations
3. Generating a WSDL for data workflow operations
4. Generating a WSDL for lineage
5. Generating a WSDL for administration
6. Generating a WSDL to modify the default directory

41.1 Generating a WSDL for operations on data

To generate a WSDL for accessing data, select 'Data' in the navigation panel in the **Data Services** area, then follow through the steps of the wizard:

1. Choose whether the WSDL will be for operations at the dataset level or at the table level.
2. Identify the dataspace and dataset on which the operations will be run
3. Select the tables on which the operations are authorized, as well as the operations permitted.
4. Download the generated WSDL file by clicking the button **Download WSDL**.

Operations on datasets

The following operations can be performed using the WSDL generated for operations at the dataset level:

- Select dataset content for a dataspace or snapshot.
- Get dataset changes between dataspaces or snapshots
- Replication unit refresh

Operations on tables

The following operations, if selected, can be performed using the WSDL generated for operations at the table level:

- Insert record(s)
- Select record(s)

- Update record(s)
- Delete record(s)
- Count record(s)
- Get changes between dataspace or snapshot
- Get credentials
- Run multiple operations on tables in the dataset

See also

WSDL download from HTTP protocol [p 1073]

Operations generated from a data model [p 1079]

41.2 Generating a WSDL for dataspace operations

To generate a WSDL for dataspace-level operations, selecting 'Dataspace' in the navigation panel of the **Data Services** area. The generated WSDL is generic to all dataspaces, thus no additional information is required.

Download the generated WSDL file by clicking the button **Download WSDL**.

Operations on dataspaces

The following operations can be performed using the WSDL generated for operations at the dataspace level:

- Create a dataspace
- Close a dataspace
- Create a snapshot
- Close a snapshot
- Merge a dataspace
- Lock a dataspace
- Unlock a dataspace
- Validate a dataspace or a snapshot
- Validate a dataset

See also

WSDL download from HTTP protocol [p 1073]

Operations on datasets and dataspaces [p 1100]

41.3 Generating a WSDL for data workflow operations

To generate a WSDL to control data workflows, select 'Data workflow' from the **Data Services** area. The generated WSDL is not specific to any particular workflow publication, thus no additional information is required.

Download the generated WSDL file by clicking the button **Download WSDL**.

Operations on data workflows

- Start a data workflow
- Resume a data workflow
- End a data workflow

See also

WSDL download from HTTP protocol *[p 1073]*

Operations on data workflows *[p 1106]*

41.4 Generating a WSDL for lineage

To generate a WSDL for lineage, select 'Lineage' from the **Data Services** area. It will be based on authorized profiles that have been defined by an administrator in the 'Lineage' section of the **Administration** area.

The operations available for accessing tables are the same as for WSDL for operations on data [p 247].

Steps for generating the WSDL for lineage are as follows:

1. Select the profile whose permissions will be used. The selected user or role must be authorized for use with lineage by an administrator.
2. Identify the dataspace and dataset on which the operations will be run
3. Select the tables on which the operations are authorized, as well as the operations permitted.
4. Download the generated WSDL file by clicking the button **Download WSDL**.

See also *Lineage* *[p 244]*

41.5 Generating a WSDL for administration

This action is only available to administrators.

To generate a WSDL for:

- managing the user interface
- getting system information

select 'Administration' from the **Data Services** area.

Operations for administration

- Close user interface
- Open user interface
- Get system information

See also

WSDL download from HTTP protocol *[p 1073]*

User interface operations *[p 1110]*

System information operation *[p 1110]*

41.6 Generating a WSDL to modify the default directory

This action is only available to administrators, and only if using the default directory.

To generate a WSDL to update the default directory, select 'Directory' from the **Data Services** area.

Operations on the default directory

The operations available for accessing tables are the same as for WSDL for operations on data [p 247].

See also

WSDL download from HTTP protocol [p 1073]

Directory services [p 1109]

EBX® Metadata Management

CHAPTER 42

Introduction to metadata management

This chapter contains the following topics:

1. Understanding metadata and its importance
2. Challenges and solutions
3. Audience
4. Approaches to metadata management
5. Where to go from here

42.1 Understanding metadata and its importance

Metadata provides information about data itself, like what it is, what it's for, and how it's organized. It also includes information about the business and technical processes related to the data, and the connections between different pieces of data.

Managing metadata well helps an organization understand and control its operations, systems, and how work gets done. This is important for making the most of the data you have and sometimes it's even required by law.

If you don't manage metadata properly, you might end up with messy and confusing data. It would be like trying to find a specific book in a library without a catalog; you wouldn't know where to start looking, or even if the book is there.

In summary, metadata tells you what data you have and helps you find and use it effectively. Otherwise, it can be hard to make sense of all the information you have.

42.2 Challenges and solutions

One of the biggest challenges is in implementing a metadata management strategy that unifies the way you describe, manage, map, and discover your data assets. Critical to the success of strategy implementation is a mutual understanding of the strategy between business parties, a controlled data flow, and a solid definition of high-quality data.

EBX®'s metadata management capabilities ensure end-to-end management so you can govern, master, map, and exchange data. Using the features included in the EBX® Metadata Management application, you can implement:

- **A Data catalog**: helps you document your information system by classifying your data sources. The goal is to achieve a well-defined view of your information system through different lenses: technical, business, and data specialists profiles. When you document and map your data, you can identify and bridge potential gaps. This allow users to find the data they need, when they need it.
- **A Business glossary**: allows you to document business terms, their definitions, and the relationships between terms. A unified vocabulary facilitates communication in your organization by supplying a common understanding of business concepts. Issues solved by creating a business glossary include: an inconsistent business vocabulary, data redundancy and duplication, data freshness, and data quality.
- **Metadata management**: ensures that you obtain value from data by using metadata to inventory, assess and analyze data assets. For example, you can analyze metadata and data for potential added value such as identification and capture of additional metadata or datasets. In other words, it can help you capture data you might not realize you have. Metadata management enhances the experience that users have while interacting with business solutions or products. This creates a collaborative experience and ensures that different types of users find the appropriate virtual environment in which to work.

42.3 Audience

The EBX® Metadata Management application can address the needs of the most diverse organizations and facilitate collaboration across teams. Business users, data architects, data engineers, data stewards, analysts, application architects, application owners, etc. are provided with collaborative workflows. EBX® uses profiles to support role-based access which ensures appropriate exposure of key data elements and provides needed compliance.

42.4 Approaches to metadata management

There are two main types of approaches to metadata management with EBX®:

- Bottom up examples: starting with mapping out the information system, a metadata catalog initialization or an application owner who wants to retro-document their application.
- Top down examples: the Architecture Board launches an information system cartography project, from the business/applicative architecture to the technical architecture.

42.5 Where to go from here

It is important to understand the Key terms [p 256] used in the documentation. If you are a business user, you might want to familiarize yourself with an overview of the User interface [p 261] components and how to use some of the generic features. If you are an administrator, you might benefit from understanding the configuration options [p 796] available to administrators.

CHAPTER 43

Key terms

The following table defines key terms commonly used when discussing the EBX® Metadata Management application's features:

Term	Description
Application	Software, services, reports, or other documents that exploit, manipulate, or expose data. Applications can serve business processes and be subject to rules.
Asset	Any container or provider of structured data that can be referenced and documented. System instances can contain or expose assets and assets can be comprised of other assets. For instance, an Excel file is an asset. It contains sheets which contain tables, columns, lines and cells—all of which are assets. Assets can supply and be supplied by other assets via a technical flow.
Business process	A collection of related activities or tasks completed in a specific sequence. They produce business value in the form of a product or a service for a customer or user.
Business term	Word or phrase that describes a concept used in a branch or domain. A business glossary governs and communicates business terms.
Data element	An atomic unit of data that has precise meaning or semantics. It can represent a complex object such as a Customer or a Product, or it can define a simple value such as a Birth date or a First name. Simple data elements are combined to create more complex elements. For example, a Customer is composed of a Person, which is composed of a First name and other data elements. A data element can also have a relationship with other data elements.
Dataset	A collection of data of the same structure that defines a population of the same data element, or set of data elements. A dataset can be an exhaustive list of countries or a list of French customers aged 18-24. In these 2 examples, we have a natural link between the dataset and one high level data element which are respectively Country or Customer. However, it can be more complex or less natural, such as a list of product entries from an ERP which can represent different data elements with the same structure for technical convenience.
Environment	Any application can exist in or on many environments for different reasons. It can be due to a geographical organization where an application must be redundant for non-functional requirements. It can also be due to the business or structural organization where the segregation of duties implies the creation of many separated environments. But most likely, it is required for the build and configuration of the application that needs to be tested and qualified before to go to production. An environment is then composed of all the system instances its architecture relies on.
Flow	The movement of data between two applications. Exchange of data can include many individual flows between sources and targets. At a logical level, technical flows define and allow data exchange to occur. Additionally, they specify a flow's non-functional requirements and any information that could help understand or build them.

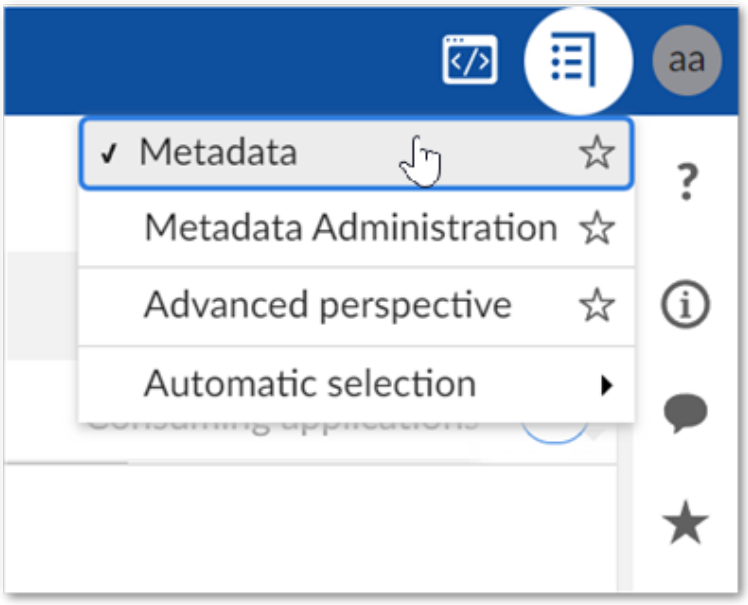
Term	Description
Group	Group of persons, includes users of the application as well as non-users, but all persons of the group are meant to have a role in the management of data: dataset owner, system manager, etc. Examples of groups are: Marketing group, IT group, etc. Identifying this group of persons will be used to assign them the ownership of a documented object, e.g.: system, instance, etc. The responsibilities linked to the management of the various objects are also linked to groups.
Infrastructure	Foundation or frameworks that supports systems. It is composed of physical or logical resources that support storage, processing, analysis and exchanges of data. An infrastructure is in most case a physical or virtual server running an operating system.
Instance	Concrete installation of a given system, it can be the installation of a software, a file system, a service provided by a service provider, an internal service, or any other container of provider of data assets or components of any application. It runs on an infrastructure.
Link	It materializes the relation between two data elements in data assets. It then defines how the links between two assets are ensured in the containing system. It can be a database foreign key, an hyperlink, or just a code or any other technical representation.
Relation	Linking two data elements, it makes one the representation of the other. It defines why the data elements are related and in which conditions. It can also carry rules. There can be many relationships between two same data elements. A "Company" can be both the "Supplier" and the "Customer" of another "Company". As different rules can apply to these two cases, it is interesting to be able to create two different relations instead of one with a multiple condition.
Role	This is not the definition of users roles within EBX® but roles upon data in the information system. A role is not necessarily active on data governance or data management but can define, for instance, a passive notified role. Each element of a RACI matrix can then be defined as a role given to a list of groups of persons.
Rule	Defines or limits some aspects of a given data category to control or influence the behavior of business data. It is used for making decisions and for governing programs or policies.
Provider	Any company providing one or many systems. It can be a software vendor, a system integrator or any other outsourcing company, but it can also be an internal organization.
System	Any technical component that participates to the elaboration of an application, but also any technical component that provides or uses data. A system is then materialized by an instance.
Technical flow	A data aims to be used but is also sourced. It then comes from an origin going though intermediaries to reach a final destination before reaching another in the future. Each step in this journey that is a flow between two or many applications is a technical flow. It defines the bridge between two assets, telling at any level what is the source of any target.

CHAPTER 44

Accessing the metadata perspective

You can access the EBX® Metadata Management application and its related features from the **Metadata** perspective. This perspective is only available to you if the metadata module was deployed and access was granted to you by an administrator.

To access the metadata perspective, click the **Select perspective icon** in the top-right of the main toolbar, and select **Metadata**. Note that the **Metadata administration** perspective is only available for administrator roles.



CHAPTER 45

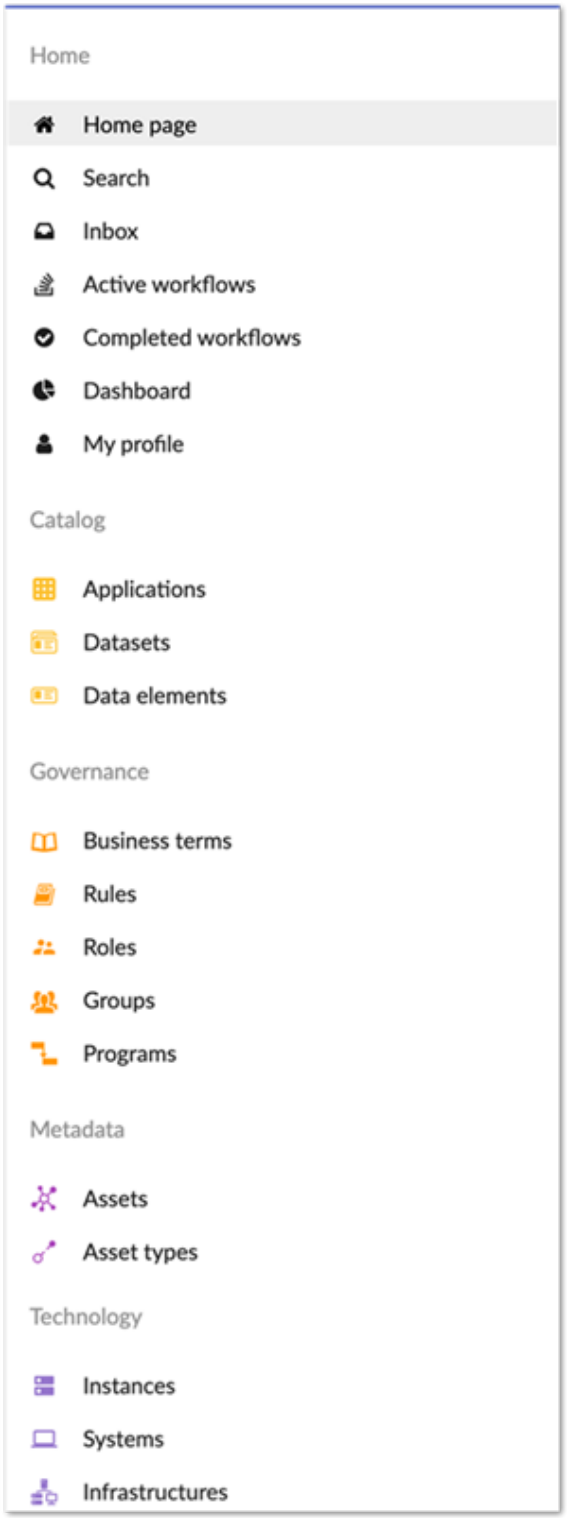
Overview of the User Interface

This chapter contains the following topics:

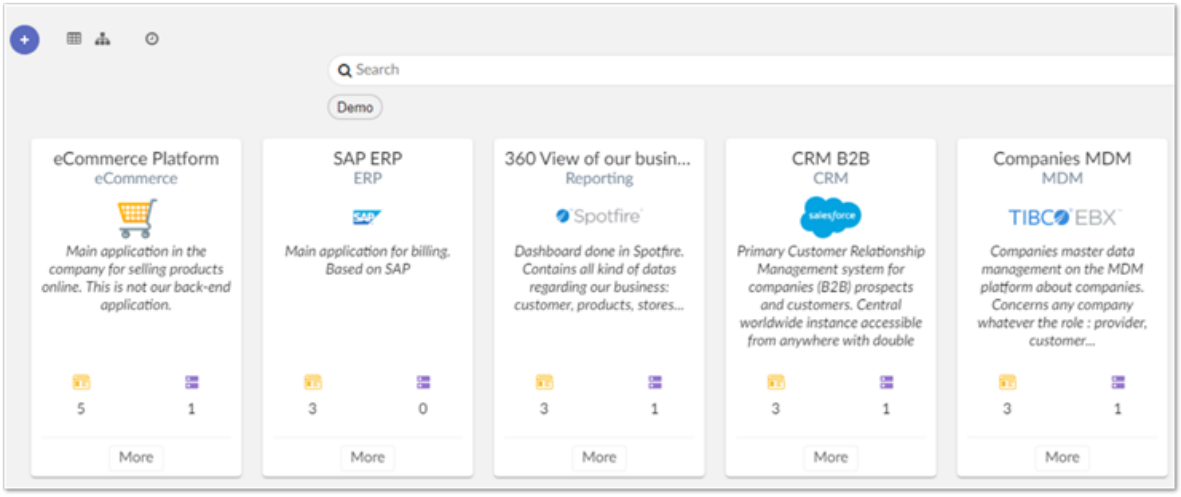
1. Main UI

45.1 **Main UI**

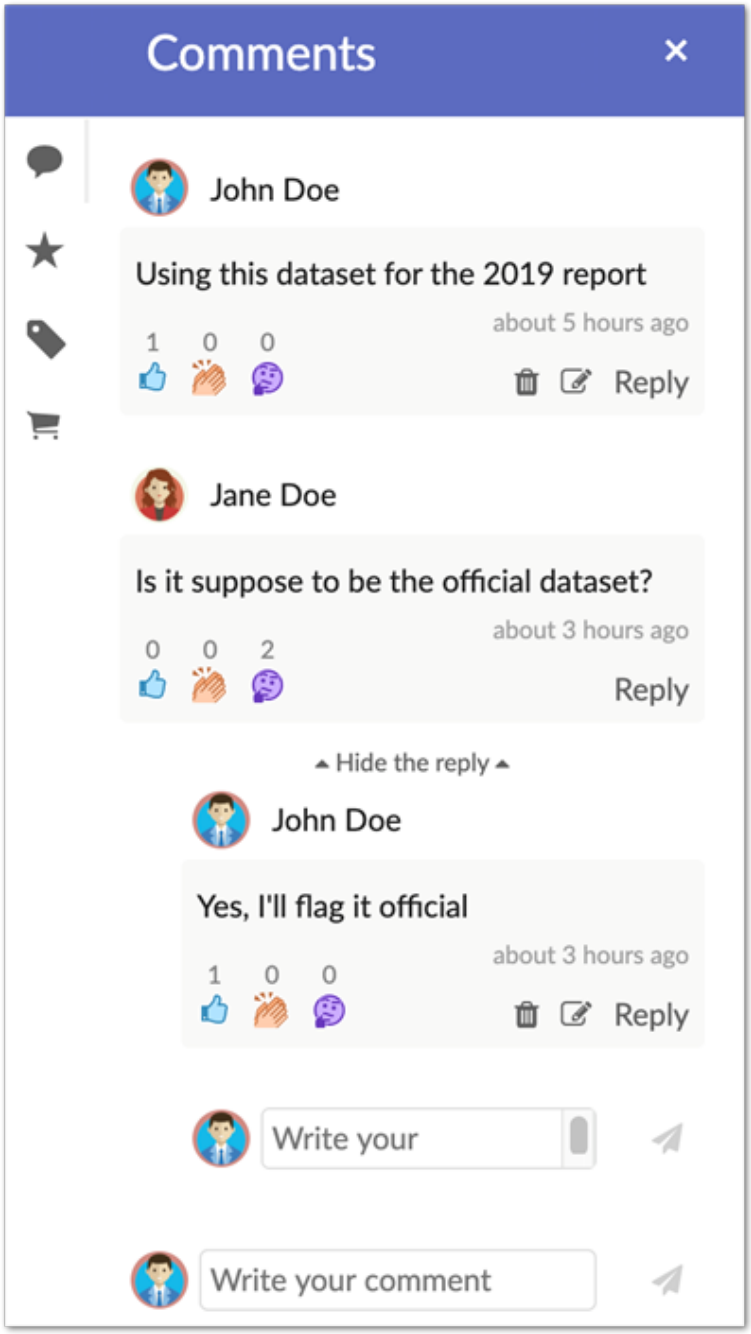
The left of the screen is the main navigation pane:



The main part of the screen is the workspace:



To the right of the workspace is the sidebar that contains contextual services according to the content of the workspace, such as comments, or information about a selected asset.



CHAPTER 46

Navigation pane overview

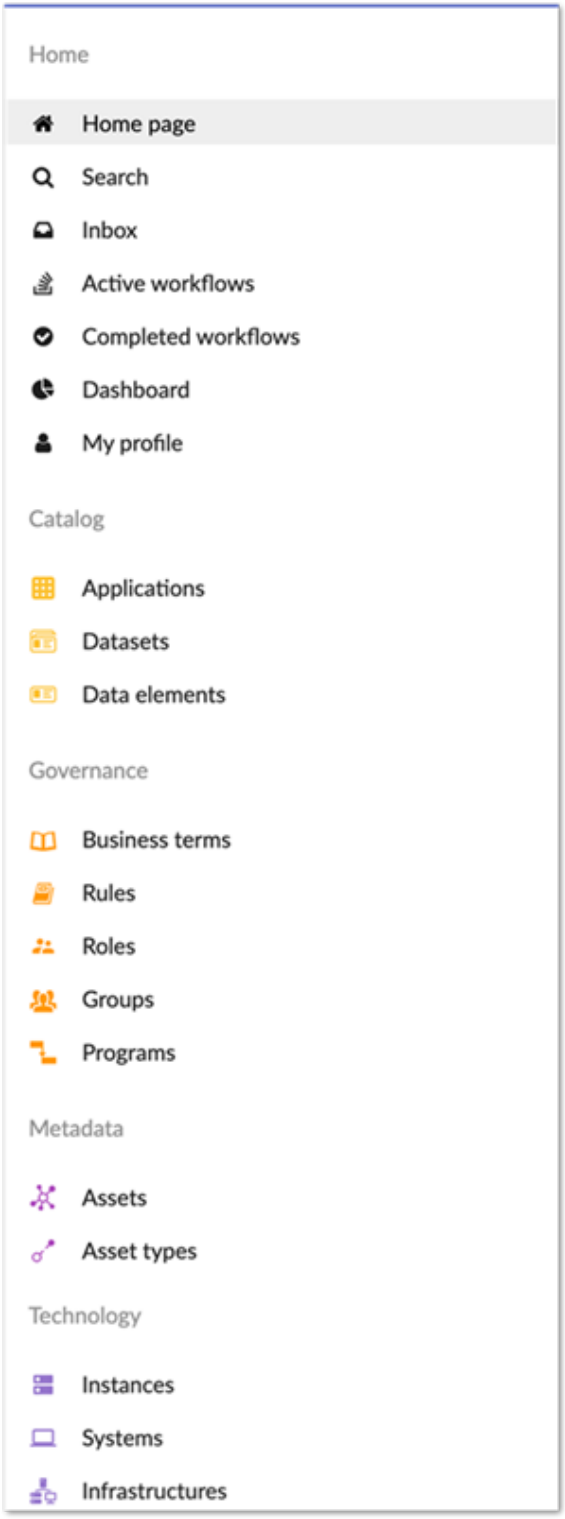
This chapter contains the following topics:

1. Overview

46.1 Overview

The **Navigation pane**, is located at the left of the screen. It allow you to access to all features available to your profile. Administrators use permission settings to determine accessible features. Each section contains a color-coded group of related features.

To change the size of the **Navigation pane**, drag its right border. Additionally, you can collapse display to show icons only by clicking the arrow in the upper right corner.



Home section

The options in the **Home** section are not associated to a specific type of metadata. The section includes an home page, a global search, accesses to workflows, dashboards and your profile where you can modify your details.

Catalog section

The **Catalog** section groups all logical descriptions including applications, datasets and data elements. A logical description is a way to represent entities that is easier to understand. Logical descriptions can apply to entities that do not exist yet. For example, you can describe an application before it is created, or document a dataset before populating it with data values that are stored in, or provided by, a physical asset.

Governance section

The **Governance** section includes documentation about how data is managed by different business domains. Each domain can have its own data related terminology and roles. This section also defines user groups, their related roles, and data rights. Comprehensive metadata management programs aim to register any program, initiative or project around data.

Metadata section

The materialization of the dataset described in the catalog is documented in the **Metadata** section. This section includes assets, which are physical resources that contain or provide data. Assets are the materialization of the logical descriptions in the Catalog Asset technical structures and properties are stored according to their types.

Technology section

The **Technology** section allows you to access the more technical aspects of metadata management. One of the main components is instances. Instances are comprised of one or more systems that expose data. A system can also be any system that is included in the process of data management and governance. Systems are supplied by a provider which is a vendor, external, or internal organization.

Privacy section

The management of data privacy, as is required by many regulatory authorities, stipulates an inventory of processing activities and personal consents. It also requires that any request coming from a personal data owner is registered and that data breaches are communicated to owners. Connected to physical assets containing them, systems processing them and concepts defining their nature, all request conditions are unified to ensure they comply with any rules that their usage implies.

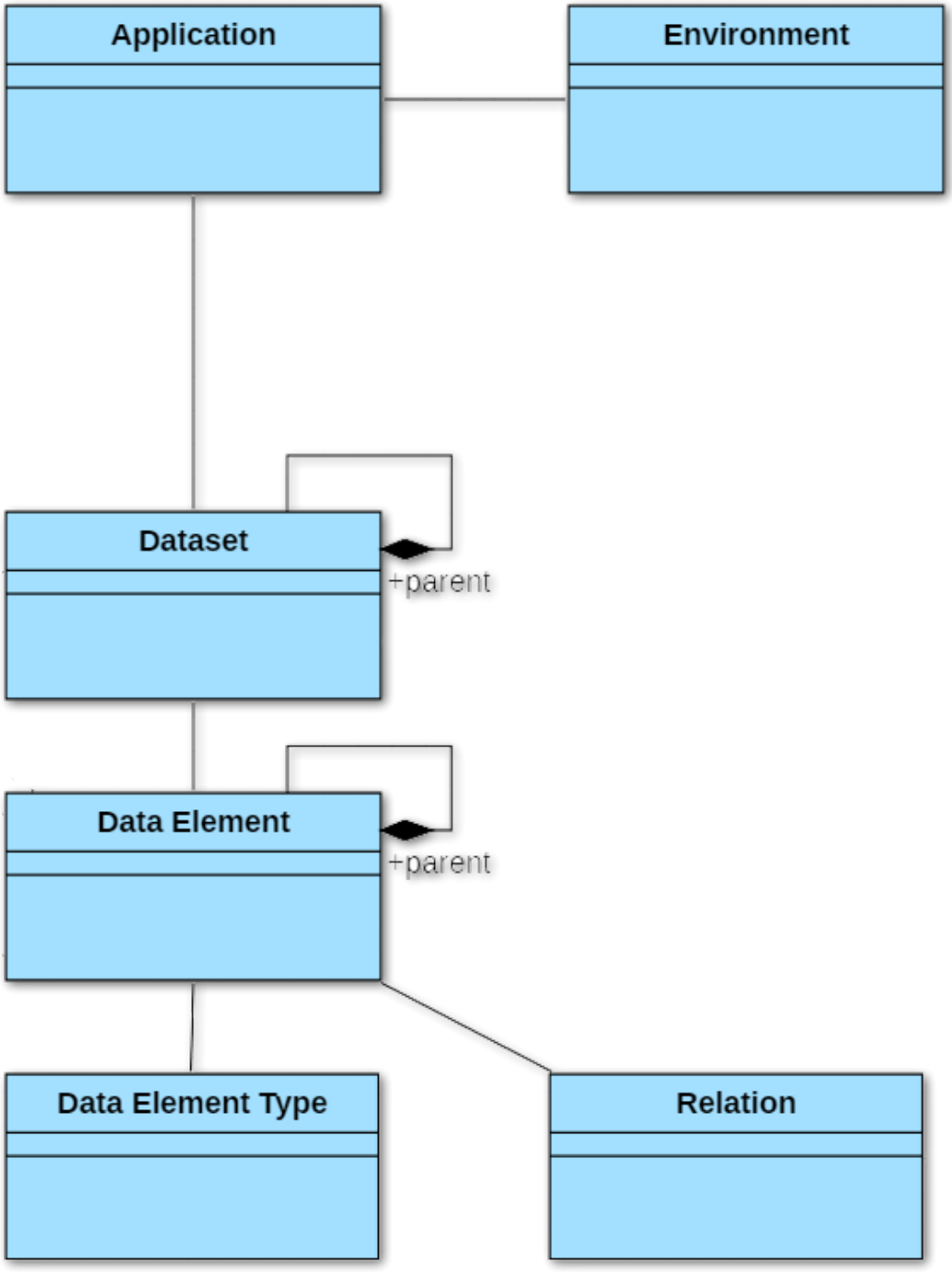
CHAPTER 47

Catalog overview

This section covers logical descriptions for applications, datasets and data elements. A logical description is a universal and common description of entities. These descriptions can exist before entities are created. for example:

- An application can be *described* before it exists.

- A dataset can be *documented* before the corresponding data values are stored in, or provided by, a physical asset.



CHAPTER 48

Creating applications

This chapter contains the following topics:

1. Overview
2. Creating a new application
3. Creating a new Data protection level (CIA) entry
4. Creating a new application type

48.1 Overview

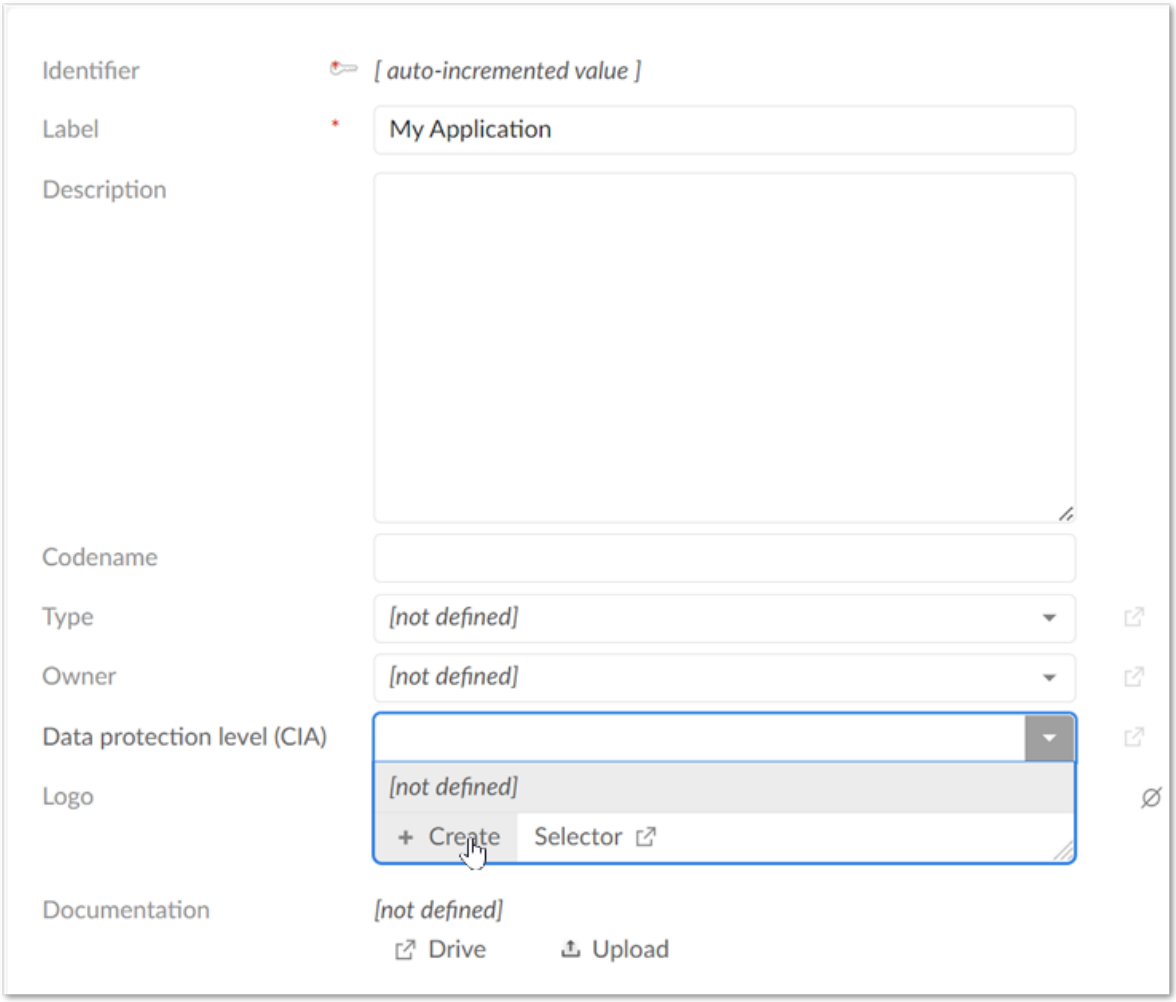
In the Catalog menu of the main control panel, select **Applications** to access the application screen. The main page listing all existing applications displays.

48.2 Creating a new application

After selecting the + icon in the top-left corner, you can create an application by providing the required information. Use the tooltip to view descriptions of any properties that you have questions about. When you have completed the creation of your application, select **Save and close**.

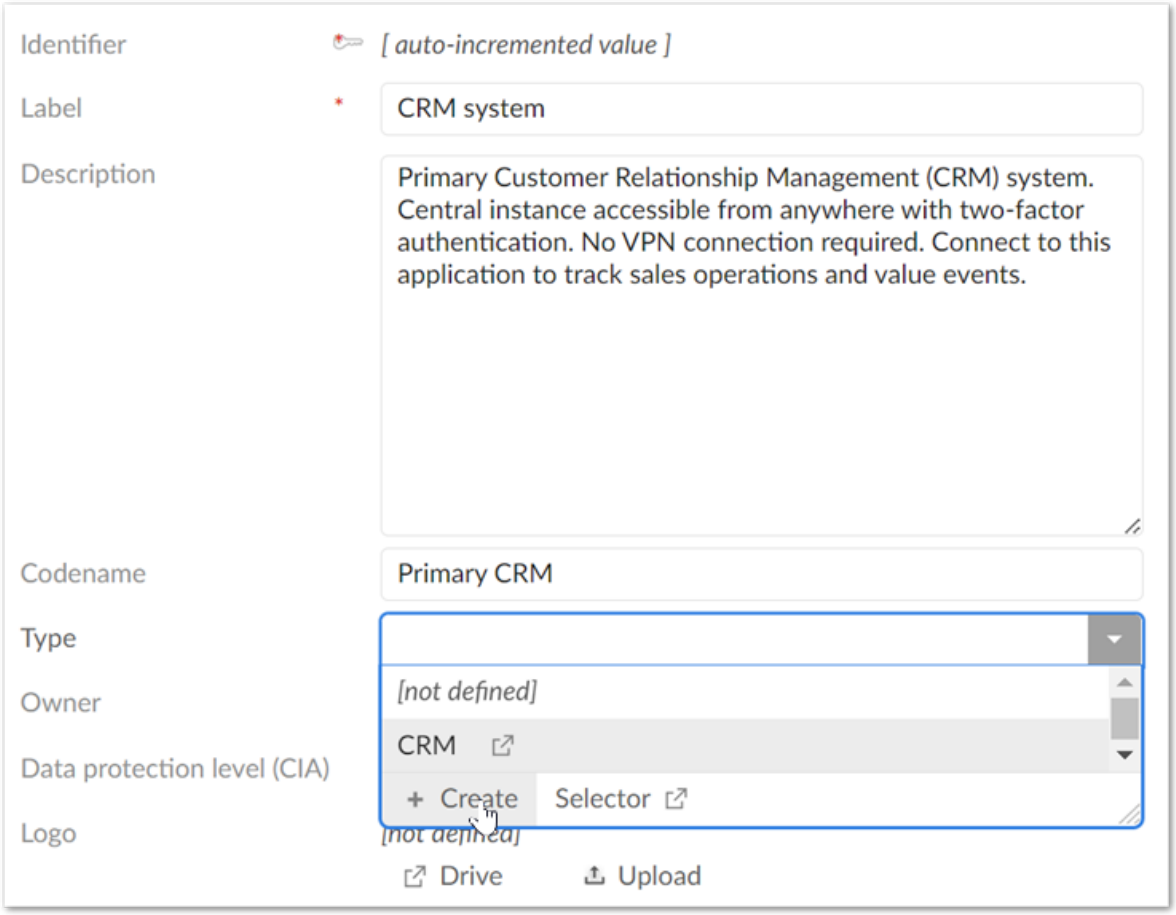
48.3 Creating a new Data protection level (CIA) entry

You can create predefined data protection level configurations to apply to your applications. When creating or updating an application, select **+ Create** from the **Data protection level (CIA)** menu.



48.4 Creating a new application type

As shown below, you can create a new application type from the **Type** menu by selecting **+ Create**.



CHAPTER 49

Creating data elements

This chapter contains the following topics:

1. Overview
2. Creating a new data element
3. Creating a new data element type
4. Creating a new data element status

49.1 Overview

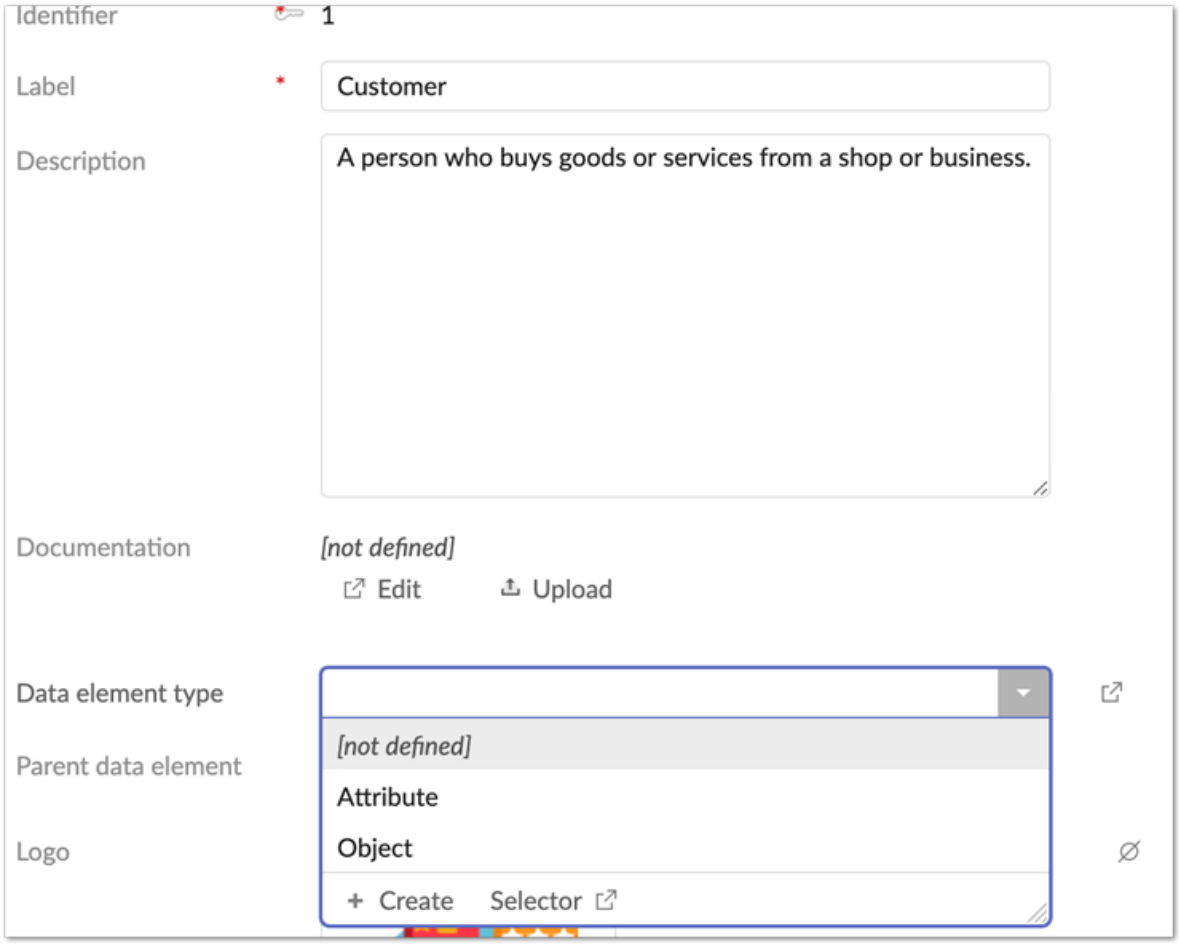
In the **Metadata** perspective's **Catalog** menu, select **Data elements**. The workspace changes focus and displays all existing data elements.

49.2 Creating a new data element

Select the + to create a new data element record. In the screen that displays you can use the fields to provide information about the data element. Use the tooltip to view descriptions of any properties that you have questions about. When you have completed the creation of your application, select **Save and close**.

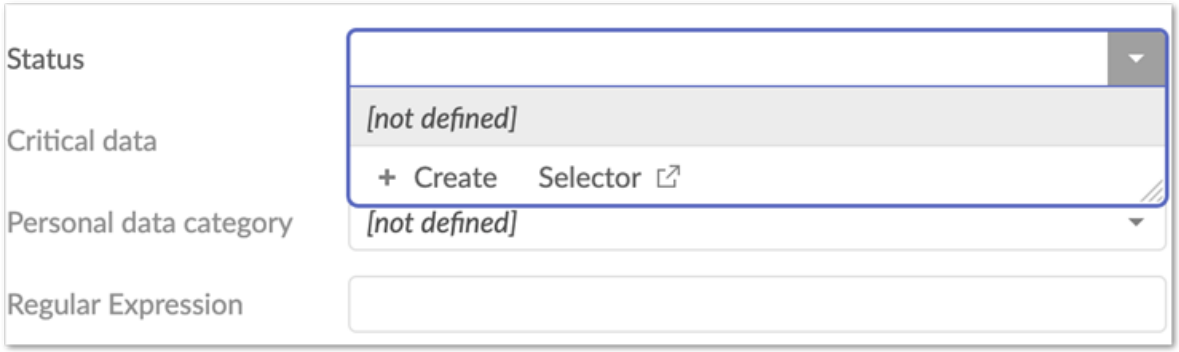
49.3 Creating a new data element type

When creating a data element, you can use the **Data element type** menu to create a new data element type.



49.4 Creating a new data element status

When creating a data element, you can create a new data element status using the **Status** menu.



CHAPTER 50

Provisioning

This chapter contains the following topics:

1. Launch Provisioning Workflow
2. First step: content analysis
3. Second step: fill view elements
4. Third step: review
5. Forth step: approvals
6. Fifth step: view finalization
7. Validation
8. Published View Updates

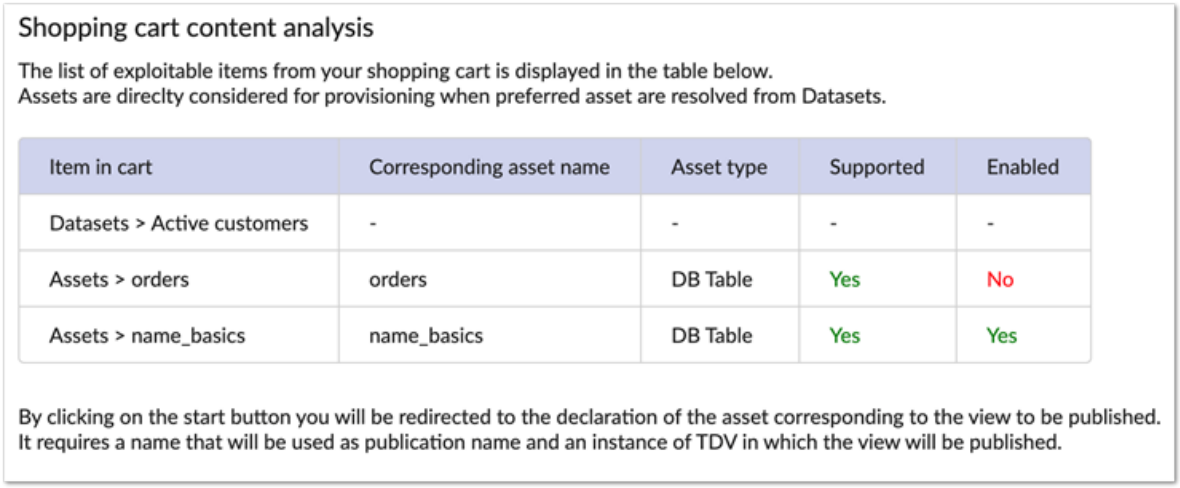
50.1 Launch Provisioning Workflow

Provisioning is a process that allows you to define a view that aggregates selected assets or datasets. In the case of datasets, the preferred asset will be used.

Select the datasets you want to include, and then click the **Request provisioning** icon (cart) in the toolbar to launch provisioning.

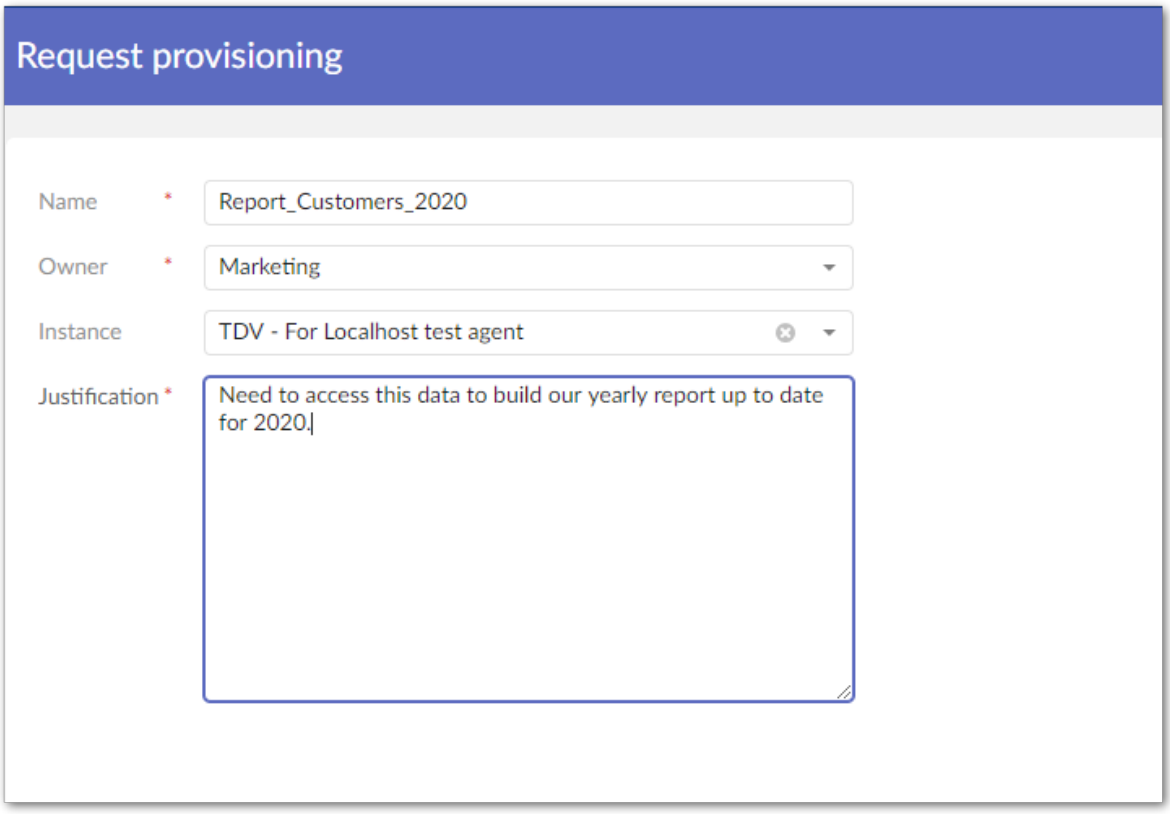
50.2 First step: content analysis

EBX® displays what can be done with the contents of the shopping cart.



After reviewing the elements select **Start**.

50.3 Second step: fill view elements



On this page, you'll have to fill in key information for the view to be created. This view will be created as an EBX® Metadata Management application asset. This asset must have a owner and must be

located in an instance of TDV where the view will be created. A justification is required by reviewers and approvers to qualify you request.

The instances of TDV available are the ones that are automatically created on agent registration.

50.4 Third step: review

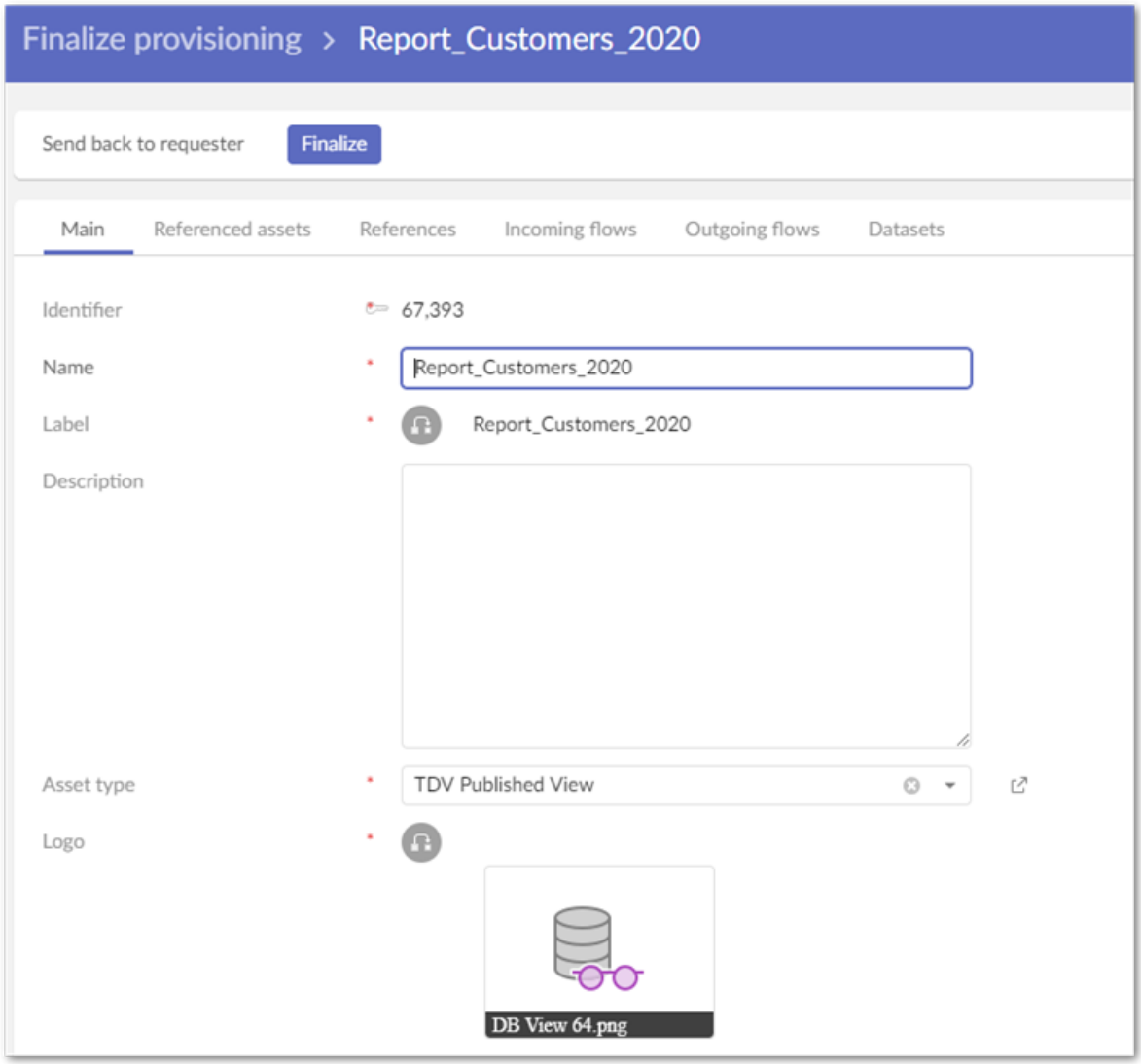
This step occurs only when a reviewer has been configured by an administrator.

50.5 Forth step: approvals

This step occurs only when at least one approver has been configured by an administrator.

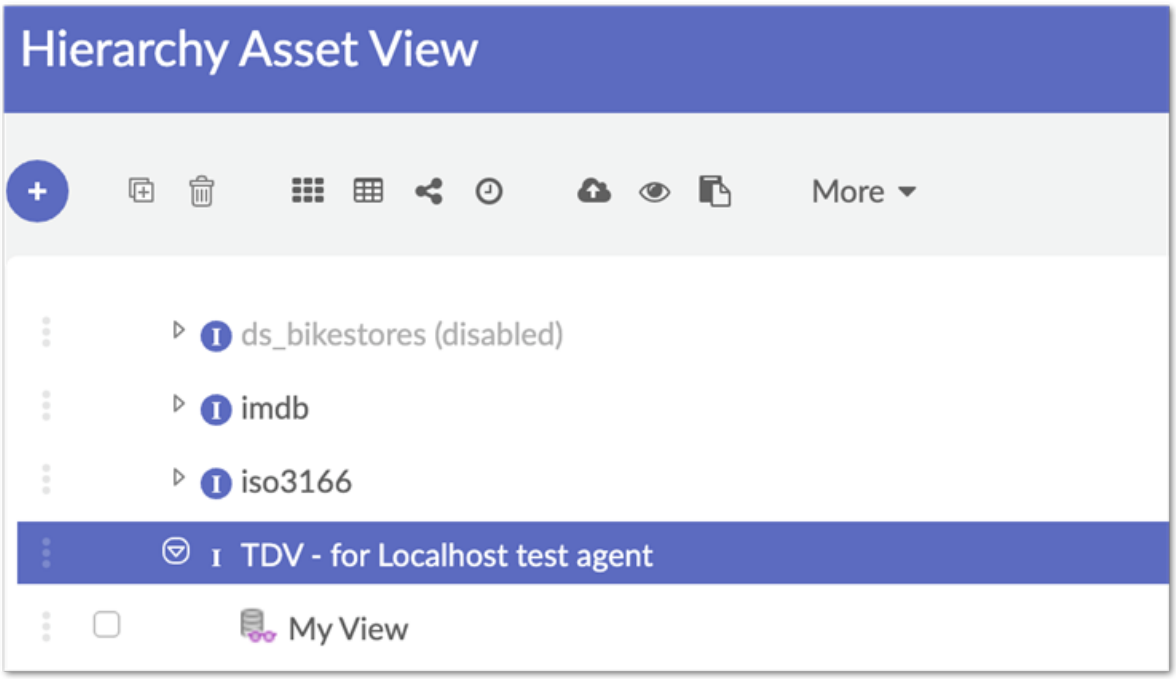
50.6 Fifth step: view finalization

When all elements are set, select **Finalize**.



50.7 Validation

You can check the created view in the EBX® Metadata Management application's assets table.



You can also check the creation of this Published TDV View in your TDV Studio.

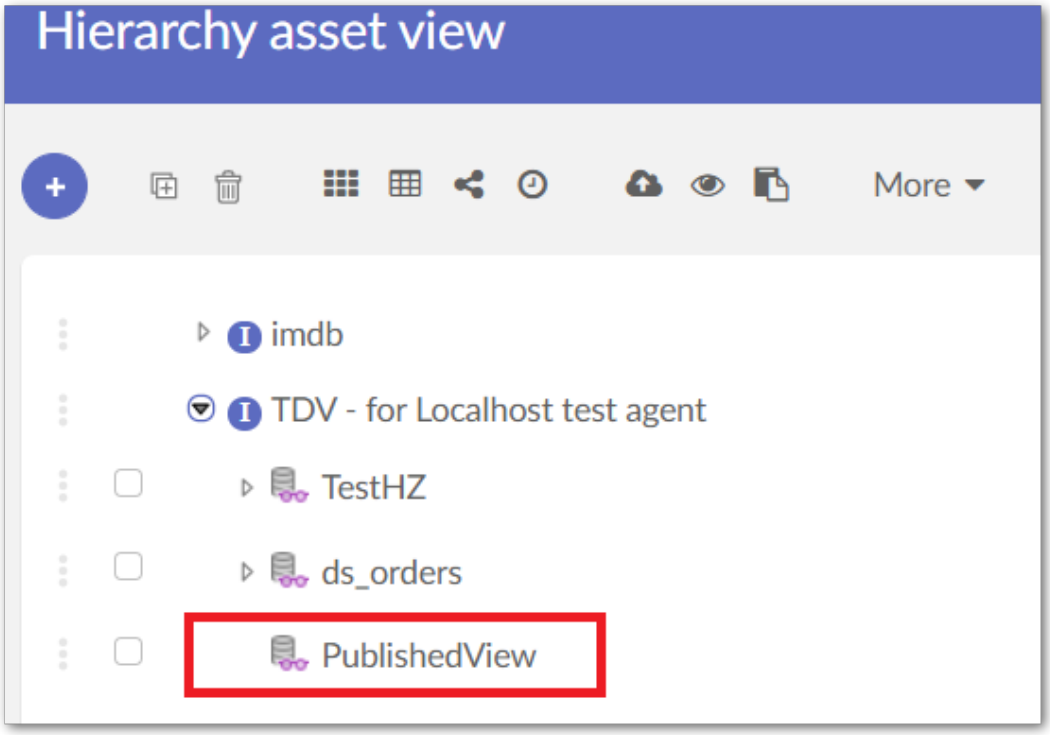
50.8 Published View Updates

For TDV published views that were created using provisioning, the owner can perform update operations from EBX®.

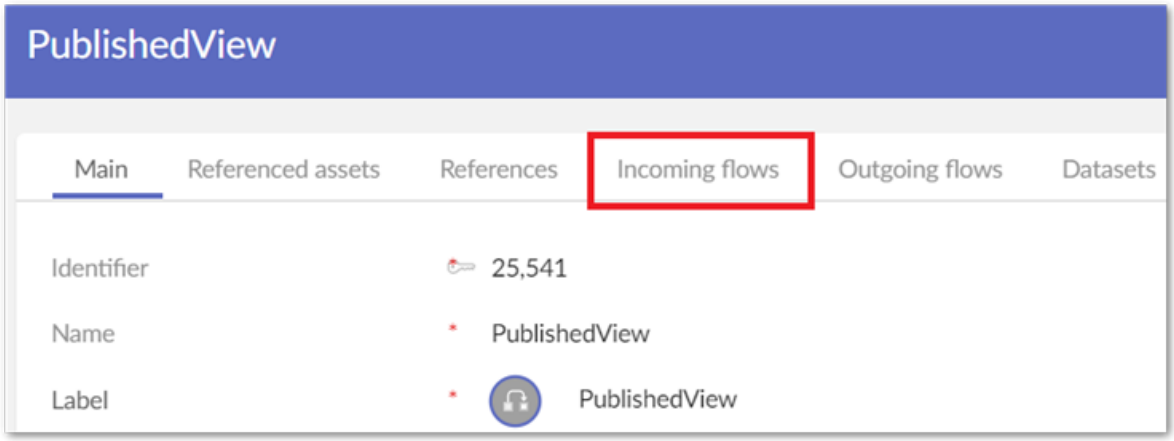
Add incoming flows to a published view

If you add an incoming flow to an asset of type TDV Published View, this triggers a publication of the corresponding resource in TDV.

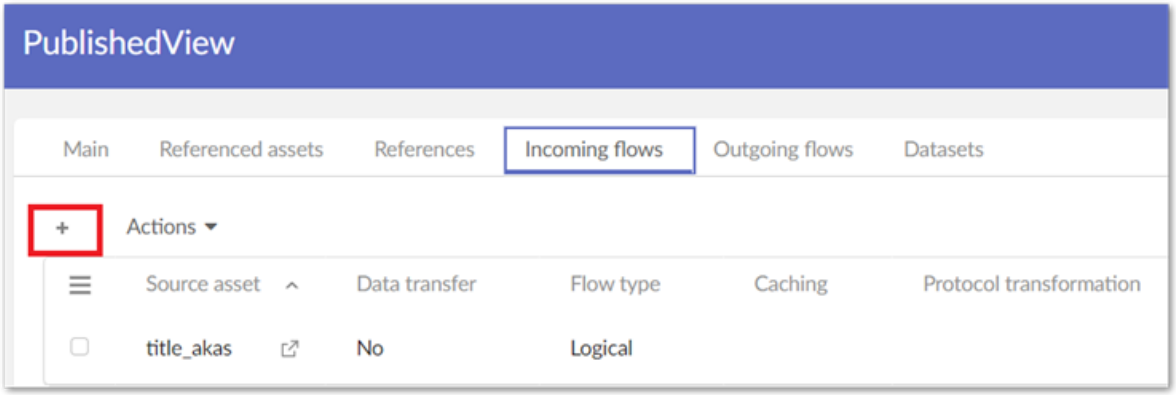
In the **Metadata** menu of the main control panel, select **Asset** to display the main page listing all existing assets. Then, open the TDV Published View asset that you wish to edit.



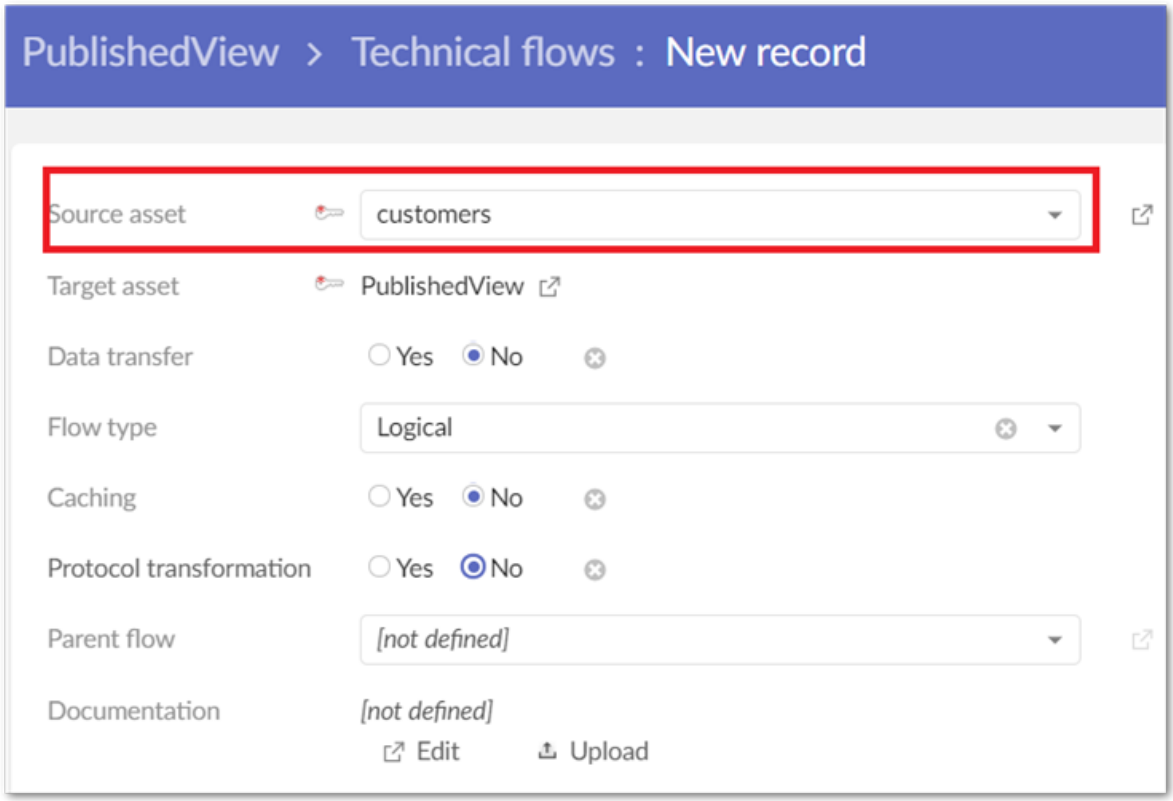
Select the **Incoming flows** tab.



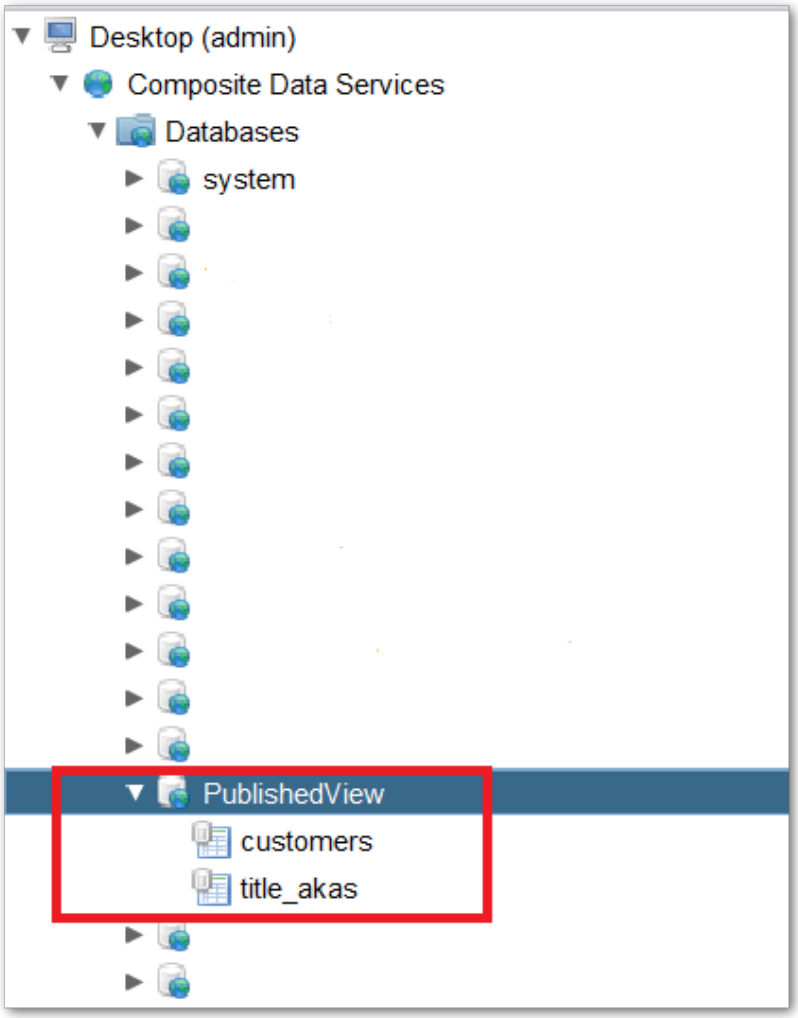
Select the + icon in the upper-left corner to create a new technical flow.



Select a source asset. Only assets that are the results of harvesting and can be published are presented in this list.



Select **Save** to validate creation of a new flow. Saving may take a little time because it will also create a corresponding publication in TDV.



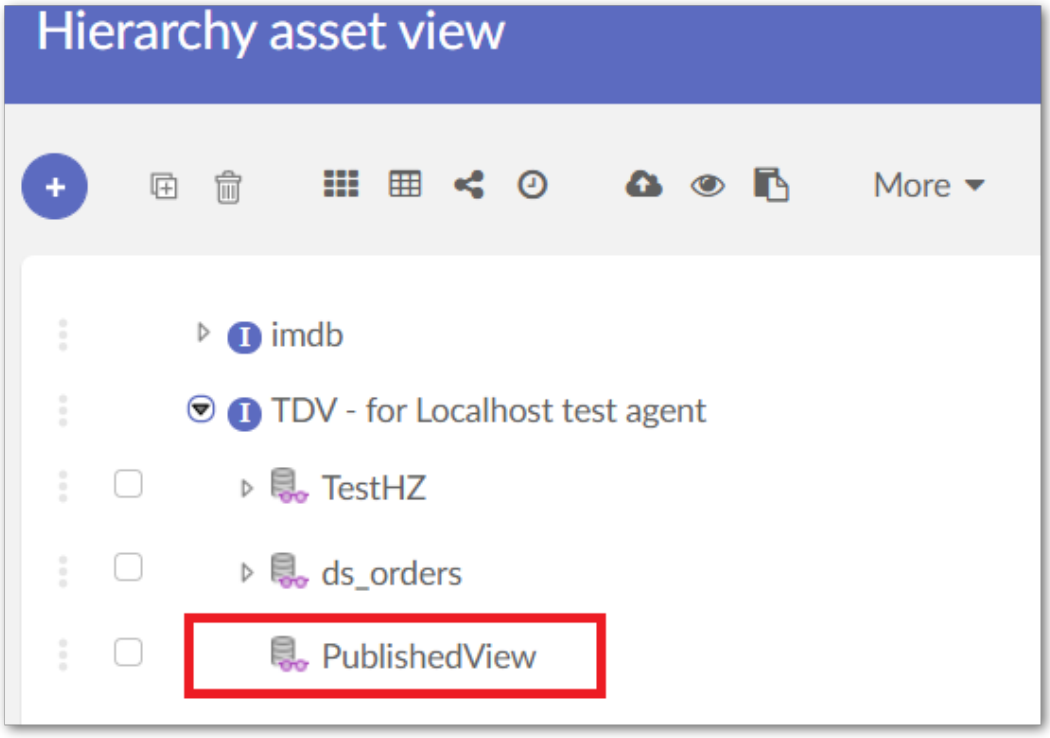
Remove incoming flows from a published view

Delete incoming flow

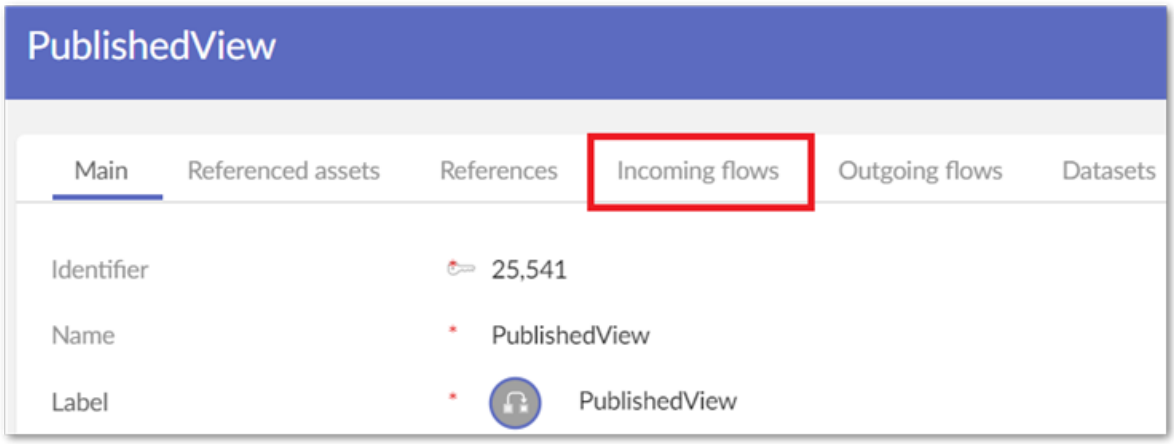
If you remove an incoming flow from an asset of type TDV Published View, this will trigger a removal of the corresponding published resource in TDV.

In the **Metadata** menu of the main control panel, select **Asset** to display the main page listing all existing assets.

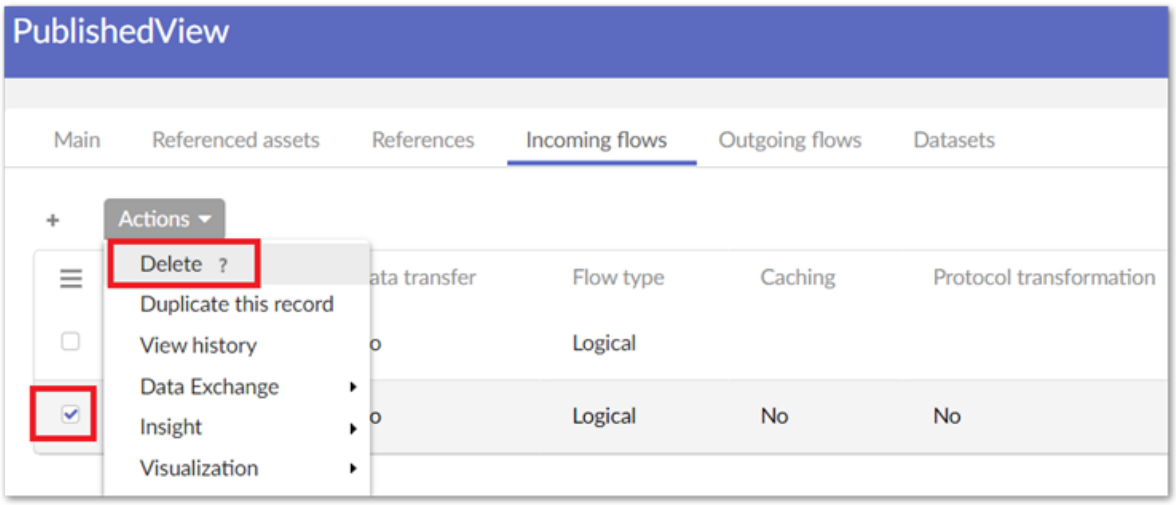
Open the TDV Published View asset that you wish to edit.



Select **Edit** and then the **Incoming flows** tab.



Select the flow you want to remove and select **Delete** in the **Actions** menu.



This will delete the technical flow and also the corresponding published element in TDV, if one exists.

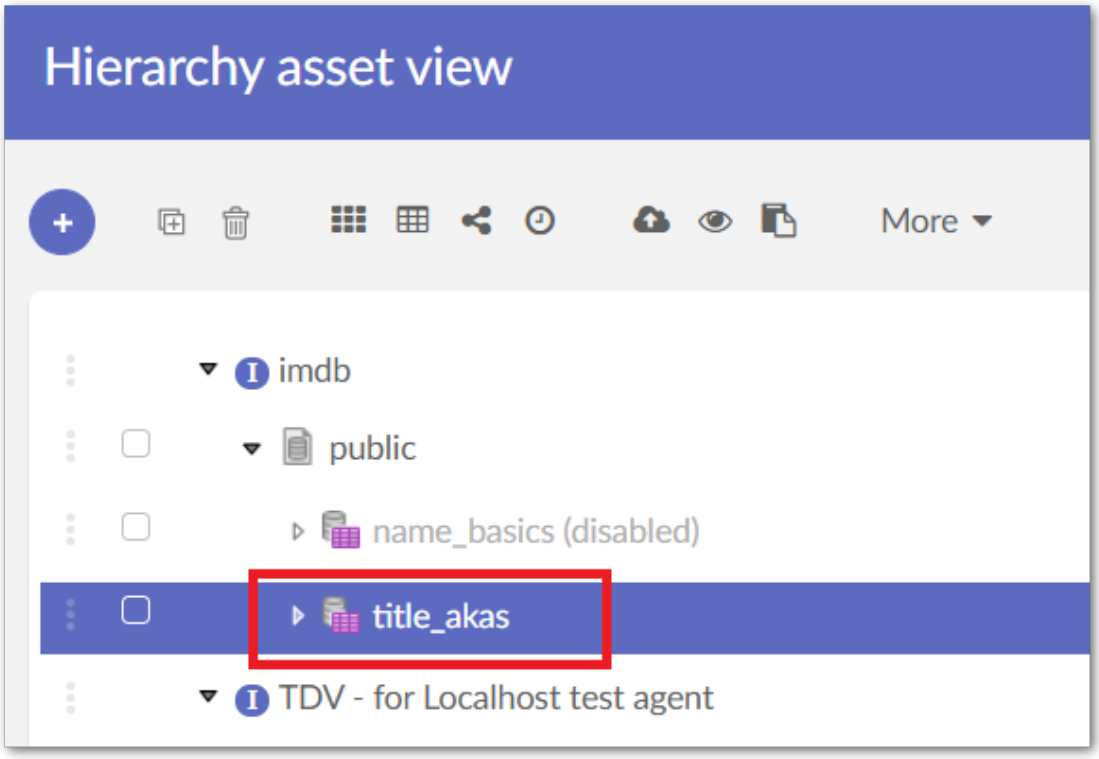


Disable asset used as incoming flow

If you manually disable an asset that is used as incoming flow for a published view, this triggers a removal of the corresponding published resource in TDV.

In the **Metadata** menu of the main control panel, select **Asset** to display the main page listing all existing assets will be displayed.

Select the asset you wish to disable and edit it.



Set the **Disabled** field to **Yes**.

A warning displays to indicate that this asset is used as an incoming flow for a TDV Published View.



This will delete the corresponding published element in TDV if one exists.

CHAPTER 51

Regular expressions

This chapter contains the following topics:

1. Overview
2. Characters
3. Character classes
4. Predefined character classes
5. POSIX character classes (US-ASCII only)
6. Classes for Unicode scripts, blocks, categories and binary properties
7. Boundary matchers
8. Greedy quantifiers
9. Reluctant quantifiers
10. Possessive quantifiers
11. Logical operators
12. Back references
13. Quotation
14. Special constructs (named-capturing and non-capturing)

51.1 Overview

Regular expressions are used to classify assets and tie them to data elements depending on whether the sample data of an asset matches with the regular expressions defined on data elements.

51.2 Characters

- `x` The character x
- `\\` The backslash character
- `\0n` The character with octal value 0n `(0 <= n <= 7)`
- `\0nn` The character with octal value 0nn `(0 <= n <= 7)`
- `\0mnn` The character with octal value 0mnn `(0 <= m <= 3, 0 <= n <= 7)`
- `\xhh` The character with hexadecimal value 0xhh

- `\uhhhh` The character with hexadecimal value 0xhhhh
- `\x{h...h}` The character with hexadecimal value 0xh…h (Character.MIN_CODE_POINT <= 0xh…h <= Character.MAX_CODE_POINT)
- `\t` The tab character (`\u0009`)
- `\n` The newline (line feed) character (`\u000A`)
- `\r` The carriage-return character (`\u000D`)
- `\f` The form-feed character (`\u000C`)
- `\a` The alert (bell) character (’007`)
- `\e` The escape character (`'\u001B`)
- `\cx` The control character corresponding to x

51.3 **Character classes**

- `[abc]` a, b, or c (simple class)
- `[^abc]` Any character except a, b, or c (negation)
- `[a-zA-Z]` a through z or A through Z, inclusive (range)
- `[a-d[m-p]]` a through d, or m through p: `[a-dm-p]` (union)
- `[a-z&&[def]]` d, e, or f (intersection)
- `[a-z&&[^bc]]` a through z, except for b and c: `[ad-z]` (subtraction)
- `[a-z&&[^m-p]]` a through z, and not m through p: `[a-lq-z]` (subtraction)

51.4 **Predefined character classes**

- . Any character (may or may not match line terminators)
- `\d` A digit: `[0-9]`
- `\\D` A non-digit: `[^0-9]`
- `\s` A whitespace character: `[ \t\n\x0B\f\r]`
- `\\S` A non-whitespace character: `[^\s]`
- `\w` A word character: `[a-zA-Z_0-9]`
- `\W` A non-word character: `[^\w]`

51.5 **POSIX character classes (US-ASCII only)**

- `\p{Lower}` A lower-case alphabetic character: `[a-z]`
- `\p{Upper}` An upper-case alphabetic character: `[A-Z]`
- `\p{ASCII}` All ASCII:`[\x00-\x7F]`
- `\\p{Alpha}` An alphabetic character:`[\p{Lower}\p{Upper}]`
- `\p{Digit}` A decimal digit: `[0-9]`
- `\p{Alnum}` An alphanumeric character:`[\p{Alpha}\p{Digit}]`

- `\p{Punct}` Punctuation: One of `!"#$%&'()*+,-./:;<=>?@[\]^_``{|}~`
- `\p{Graph}` A visible character: `[\p{Alnum}\p{Punct}]`
- `\p{Print}` A printable character: `[\p{Graph}\x20]`
- `\p{Blank}` A space or a tab: `[ \t]`
- `\p{Cntrl}` A control character: `[\x00-\x1F\x7F]`
- `\p{XDigit}` A hexadecimal digit: `[0-9a-fA-F]`
- `\p{Space}` A whitespace character: `[ \t\n\x0B\f\r]`

51.6 Classes for Unicode scripts, blocks, categories and binary properties

- `\p{IsLatin}` A Latin script character (script)
- `\p{InGreek}` A character in the Greek block (block)
- `\p{Lu}` An uppercase letter (category)
- `\p{IsAlphabetic}` An alphabetic character (binary property)
- `\p{Sc}` A currency symbol
- `\P{InGreek}` Any character except one in the Greek block (negation)
- `[\p{L}&&[^\p{Lu}]]` Any letter except an uppercase letter (subtraction)

51.7 Boundary matchers

- `^` The beginning of a line
- `$` The end of a line
- `\b` A word boundary
- `\B` A non-word boundary
- `\A` The beginning of the input
- `\G` The end of the previous match
- `\Z` The end of the input but for the final terminator, if any
- `\z` The end of the input

51.8 Greedy quantifiers

- `X?` X, once or not at all
- `X*` X, zero or more times
- `X+` X, one or more times
- `X{n}` X, exactly n times
- `X{n,}` X, at least n times
- `X{n,m}` X, at least n but not more than m times

51.9 **Reluctant quantifiers**

- `X??` X, once or not at all
- `X*?` X, zero or more times
- `X+?` X, one or more times
- `X{n}?` X, exactly n times
- `X{n,}?` X, at least n times
- `X{n,m}?` X, at least n but not more than m times

51.10 **Possessive quantifiers**

- `X?+` X, once or not at all
- `X*+` X, zero or more times
- `X++` X, one or more times
- `X{n}+` X, exactly n times
- `X{n,}+` X, at least n times
- `X{n,m}+` X, at least n but not more than m times

51.11 **Logical operators**

- `XY` X followed by Y
- `X|Y` Either X or Y
- `(X)` X, as a capturing group

51.12 **Back references**

- `\n` Whatever the nth capturing group matched
- `\k<name>` Whatever the named-capturing group name matched

51.13 **Quotation**

- `\` Nothing, but quotes the following character
- `\Q` Nothing, but quotes all characters until `\E`
- `\E` Nothing, but ends quoting started by `\Q`

51.14 **Special constructs (named-capturing and non-capturing)**

- `(?<name>X)` X, as a named-capturing group

- `(?:X)` X, as a non-capturing group
- `(?idmsuxU-idmsuxU)` Nothing, but turns match flags `i d m s u x U` on - off
- `(?idmsux-idmsux:X)` X, as a non-capturing group with the given flags `i d m s u x` on - off
- `(?=X)` X, via zero-width positive lookahead
- `(?!X)` X, via zero-width negative lookahead
- `(?<=X)` X, via zero-width positive lookbehind
- `(?<!X)` X, via zero-width negative lookbehind
- `(?>X)` X, as an independent, non-capturing group

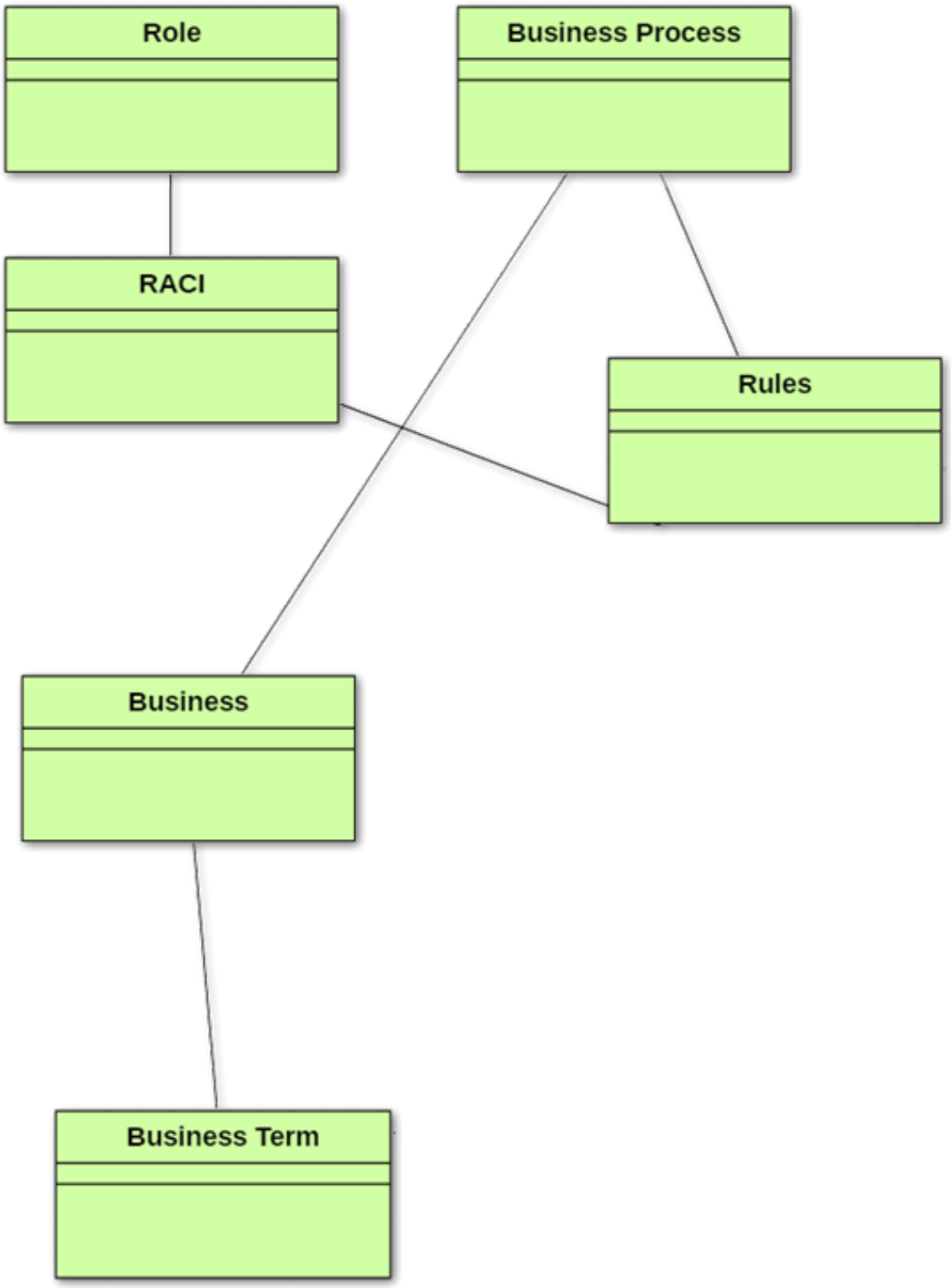
CHAPTER **52**

Governance overview

This section describes how data is managed by the different business domains.

Each domain can have its own terminology and roles. Additionally, groups of users along with their roles and entitlements over data are defined here.

Data programs aim to register any program, initiative or project around data.



CHAPTER 53

Defining a business term

To define a business term, use the **Governance** menu in the **Metadata** and select **Business Term** to open the corresponding screen. This screen displays all existing terms and gives access to the creation menu to add new terms to the list.

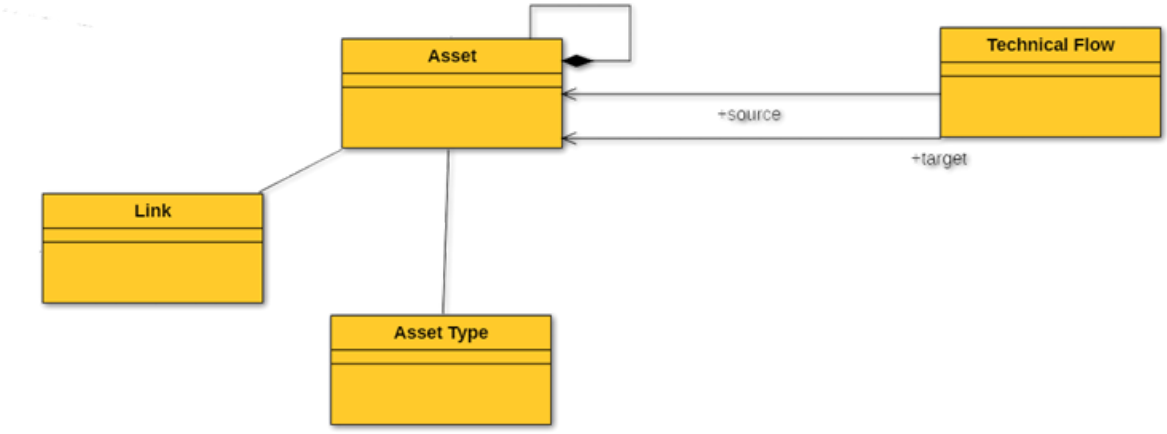
Click on the + icon in the top left corner to access the creation screen.

When you have completed the creation of your business term, click on the **Save and close** button.

CHAPTER 54

Metadata section overview

The materialization of the dataset described in the catalog is documented in the **Metadata** section.
Asset are physical resources containing or providing data.
Their technical structures and properties according to their types are located in this section.



CHAPTER 55

Managing assets

This chapter contains the following topics:

1. Create an asset
2. Modify an asset
3. Complex data types
4. Asset types
5. Create a new asset type
6. Duplicate asset type

55.1 Create an asset

Under **Metadata** in the **Metadata** perspective, select **Assets** to view your assets.

Click on the **+ Asset** button in the top left corner to access the asset creation screen.

55.2 Modify an asset

If you want to modify the current asset, click on the **Edit** button on the top left corner.

You will see the asset form pre-filled with existing data. You can modify what you want.

In the form view, you may see embedded tables. Each embedded table represent an association (way to link your current asset to other elements). In this case, you can open the **Actions** menu and:

- Select one or multiple record(s) an detach it (them).
- Select one or multiple record(s) an delete it (them).
- Create a new target record.
- Associate to existing record.

55.3 Complex data types

Some asset types are defined as containing data. These assets can also contain a data type attribute. The values that this attribute can carry are enumerated as data types and can be selected in a list. A new occurrence can also be created. A data type is made of a label and a description and is mainly informative. However, a flag can be set to define a type as complex. As soon as a data is complex, all

assets linked to this data type will share the same structure. It means that the sub tree starting from this asset will be synchronized will all assets of the same type.



Here, Test_1 and Test_1_prime have the same complex type. Their structure are synchronized from the creation of Test_1_prime which happened after the creation of Test_1. Any change on any attributes of a child of Test_1 will affect its counterpart under Test_1_prime. Any deletion or creation will also be mirrored by any assets of the same data type.

55.4 **Asset types**

Name	Description	Asset type's parent	Specific included fields
DB table	Collection of related data held in a table format within a database. It consists of columns and rows. DB Schema	- Encoding- Partitioned DB table- TableSpace- Encryption- Sample file	N/A
DB Column	Set of data values of particular simple type. May contain text values, numbers or even pointers to files in the operating system.	DB Table	- Field type- Precision- Encryption- Column index- Sample file
DB Schema	Database structure described in a formal language supported by the database management system.	- Tablespace	N/A
DB View	Searchable object in a database that is defined by a query.	DB Schema	- Tablespace
DB Function	Perform basic operations, such as Sum, Average, Count, etc., and additionally use criteria arguments, that allow you to perform the calculation only for a specified subset of the records in your database.	DB Schema	- Tablespace
DB Synonym	Represent an alias or alternate name for a table, view, sequence or other schema object.	DB Schema	- Tablespace
DB Sequence	Database object which allows users to generate unique integer values.	DB Schema	- Tablespace
REST Service	REST web services allow querying systems to manipulate web resources via their textual representations through a set of uniform, stateless, predefined operations.	- Version- URL (endpoint or URI)	N/A
REST Resource	Object with a type, associated data, relationships to other resources, and a set of methods that operate on it.	REST Service	- Field type- URL (endpoint or URI)
SOAP Service	Simple object access protocol service for exchanging structured information.	- Version- URL (endpoint or URI)	N/A

Name	Description	Asset type's parent	Specific included fields
SOAP Operation	Operation in a SOAP service like getRecord, count, Create, etc.	SOAP Service	- URL (endpoint or URI)
SOAP Request	Request to access soap operation.	SOAP Operation	N/A
SOAP Response	Response returned from a soap request link to a specific soap operation.	SOAP Operation	N/A
Excel Workbook	Collection of one or more spreadsheets also called worksheets.	- Version- Encoding- Locale- Attribute value- Encryption- File name- File size- File extension- Header present- Protected- Sample file	N/A
Excel Worksheet	Single page in a file created with Excel.	Excel Workbook	- Protected- Order- Sample file
Excel Worksheet Array	Array of worksheet.	Excel worksheet	- Number of lines- Start line- Protected- Sample file
CSV file	Comma separated value file which allows data to be saved in a tabular format.	- Version- Encoding- Encryption- File name- File size- File extension- Line separator- String delimiter- Header present- Start line- Protected- Sample file	N/A
Column	Represent a column in a CSV file or excel worksheet.	- CSV file - Excel Worksheet	- Format- Field type- Default value- String delimiter- Column index- Row index- Formula- Orientation- Sample file
XML File	Extensible markup language data file. Formatted much like an HTML document but uses customs tags to define object and the data within each object.	- Version- Encoding- XML Encoding- Encryption- File name- File size- File extension- Protected- Sample file	N/A
XML Stream	Stream of xml content.	- Encoding- XML Encoding	N/A
XML Element	Is everything from the element's start tag to the element's end tag.	- XML file - XML Stream	- Field type- Node type- Default value- TagName (attribute name)- Pattern- Range- Enumeration- Maximum length- Minimum length- Sample file
XML Attribute	Contains data related to a specific element.	XML Element	- Field type- Default value- Sample file
JSON File	File that stores simple data structures and objects in JavaScript Object Notation(JSON) format, which is a standard data exchange format.	- Version- Encoding- Encryption- File name- File size- File extension- Protected- Sample file	N/A

Name	Description	Asset type's parent	Specific included fields
JSON Element	Data structure in a JSON. Data can be a json object, text, number, boolean, etc.	- JSON file- JSON stream	- Format- Field type- Default value- Array length- Media encoding- Content media type- Pattern- Range- Enumeration- Maximum length- Minimum length- Sample file
JSON Property	Each JSON Element is identified by its JSON property.	JSON Element	- Field type- Default value- Sample file
JSON Stream	Stream of JSON content.	- Encoding	N/A
TDV Published view	Specific TIBCO® Data Virtualization published view.	- Encoding- Sample file	N/A
Undefined	Default asset type for unknown format.	N/A	N/A

55.5 Create a new asset type

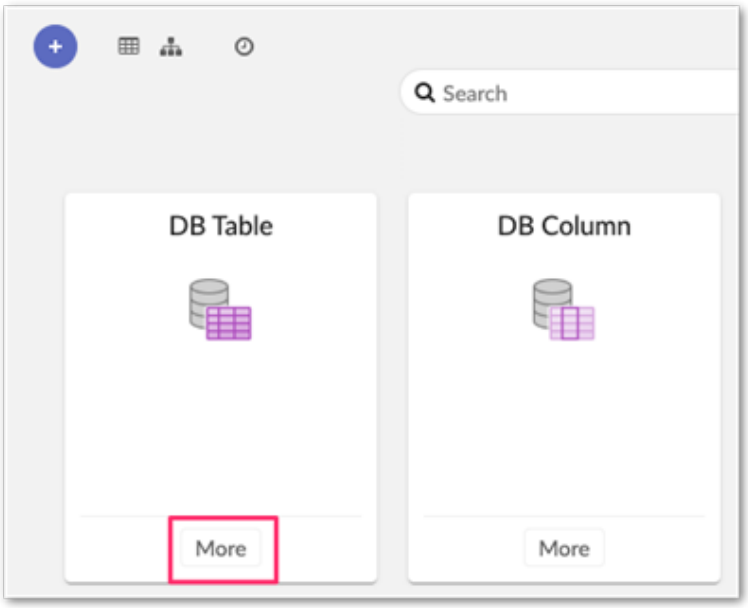
You can create a new asset type by clicking the + button on the top left corner of the asset type main view.

You will see a form with fields to complete.

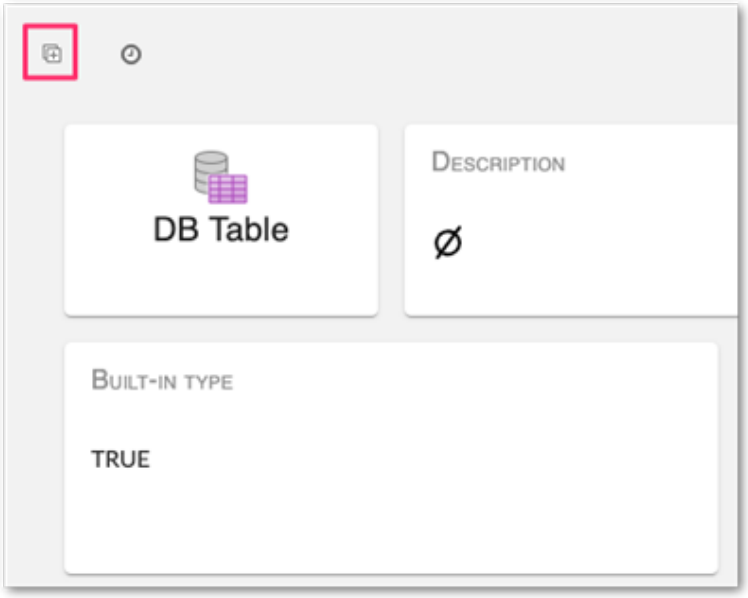
55.6 Duplicate asset type

If you want to create your own asset type from an existing one, you can click on the duplicate action button.

First of all, you will need to display the asset type you want to duplicate by clicking on it if you are in the hierarchical view or clicking on the **More** button in the card view mode.



In the asset type view form, click on the duplicate button on the top left corner.



You will see a pre-filled form in which you can make your updates.

CHAPTER 56

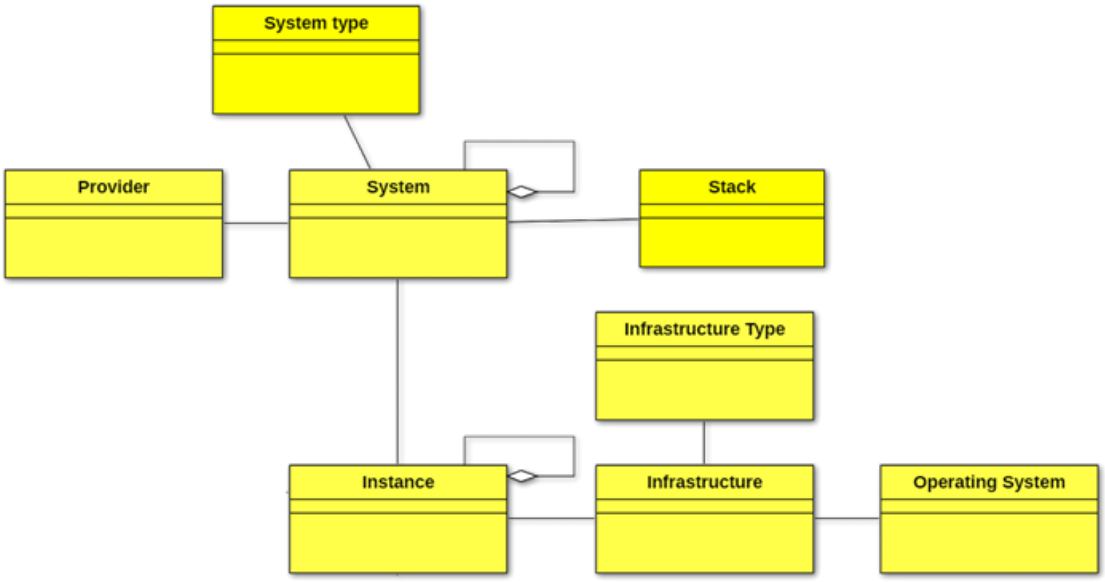
Technology overview

Data management relies on instances that are represented as many systems.

This is an instance of a system running on a given infrastructure which exposes data.

Systems can also be any system entering in the process of data management and governance.

They are provided by a provider which can be a vendor or any other external or internal organization.



CHAPTER 57

Create an instance

In the **Technology** group of the **Main perspective**, select **Instances** to access the instance screen. The main page listing all existing instances displays.

From there you have multiple choices for instance creation:

- Click on the + icon in the top left corner to access the creation screen and create an instance from scratch.
- Click on the squared + icon in the top left corner to access the creation screen from the Agent.

For the instances created from scratch you have a possibility to create a corresponding data source in TDV and harvest metadata automatically. See Create data source from instance [p 375] for more information.

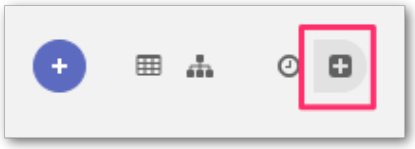
This chapter contains the following topics:

1. Create an instance from the Agent

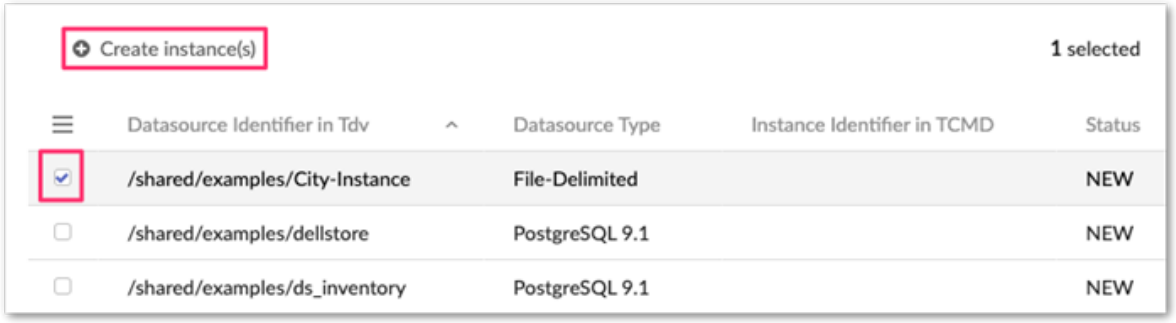
57.1 Create an instance from the Agent

To create an instance from the agent you will need to have an available agent connected to your TIBCO® Data Virtualization application.

On the instance's main page, click on the + icon at the top.



The main page listing all available data sources to add displays. In this page, you can select one or more data sources you want to add by clicking on the checkbox and selecting **+ Create instance(s)** in the top left corner.



At the end of the creation, you will see a success message and the instance will be automatically added to the instance list.

Attributes description displayed on the data source selection list:

- Datasource identifier in TDV: TDV identifier of the data source in your application.
- Datasource type: Type of the data source like Oracle, PostgreSQL …
- Instance identifier in the EBX® Metadata Management application: Identifier of the selected instance in the instance table of theEBX® Metadata Management application.

  The identifier is empty if the corresponding instance does not exist in the EBX® Metadata Management application.
- Status: Status of the TDV data source in the EBX® Metadata Management application. If equals to NEW, the data source does not exist in the application. If equals to STORED, the data source already exits.
- Created at: Datasource creation date.
- Updated at: Datasource last updated date.

CHAPTER **58**

Create a system

This chapter contains the following topics:

1. Overview
2. Create a new system
3. Create a new stack

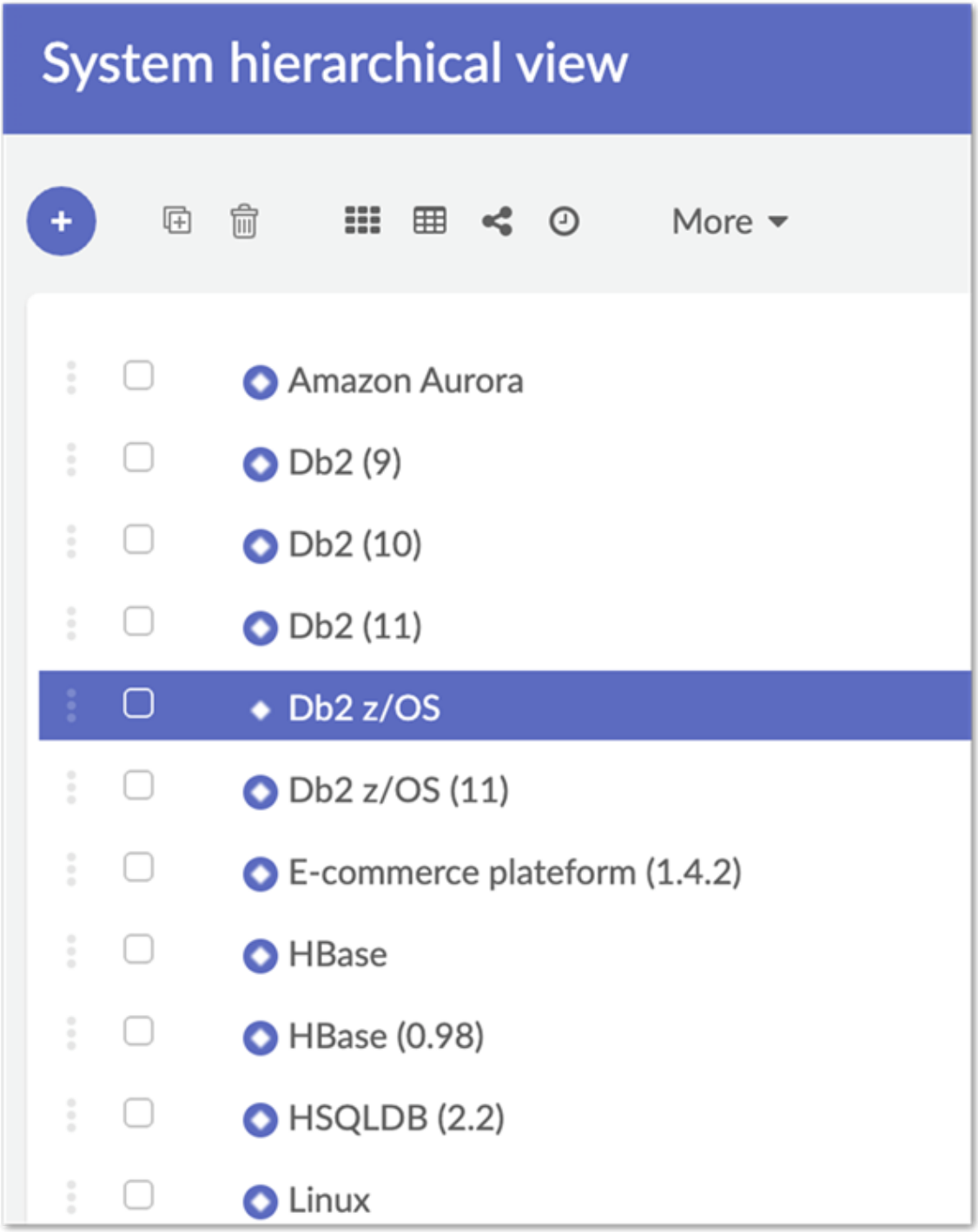
58.1 Overview

In the **Technology** group of the **Main perspective**, select **Systems** to access the system screen. A main page listing all existing systems displays.

If you click on Hierarchy view, you will see a few built-in systems needed for harvesting and provisioning features.

- TIBCO Data Virtualization
- IBM DB2 (several versions)
- HBase (several versions)
- HSQLDB
- Microsoft Excel
- Microsoft SQL Server (several versions)
- Oracle MySQL (several versions)
- Oracle Database (several versions)

- PostgreSQL (several versions)



These built-in systems cannot be deleted as are needed by harvesting & provisioning features. You are able to create other systems and system types, according to your needs.

58.2 Create a new system

After clicking on the + icon in the top left corner, you have access to the creation screen. In this screen you have to fill system information.

58.3 Create a new stack

A stack is a way to rationalize information system and technical architecture governance by associating different systems together in a structured manner and trying to limit the combinations between them.

In the EBX® Metadata Management application, a stack is defined as a system referencing another system.

To define a new Stack, create a new System using System type = **Stack** and fill the other System attributes.

Then you have two possibilities to define the stack components

- From the Stack itself, in the **Stack components** tab, associate the components (Systems) of this stack (for instance, a database + an application server and a web server, to define a web application stack)
- From each component, in the **Stacks** tab, associate this component to the wanted stack.

CHAPTER 59

Create a system type

The system types have multiple uses. For example:

- Sorting systems into families
- Defining stacks
- Defining operating systems for infrastructures
- Typing TDV infrastructure to allow provisioning

In the Technology menu of the main control panel, select **Systems** to access the system screen. When creating a System, you have to define its type.

A few built-in system types are defined:

- Operating System: this one is used to filter available operating systems in infrastructure definition page
- Database System
- Middleware
- Application Server
- Web Server
- Enterprise Service Bus
- Infrastructure Service
- Security Service
- Storage System
- Messaging System
- Event Management System
- Caching System
- File System
- Agent
- Datawarehouse
- Datamart
- Datalake
- ETL

- Hadoop-based Platform
- Registry/Directory
- Gateway
- Stack: this one is used to define [stacks]./manage_systems.html)
- Data Virtualization (TIBCO® Data Virtualization for instance)
- Cloud Platform
- Spreadsheet
- Master Data Management
- ERP
- Business Intelligence
- Undefined

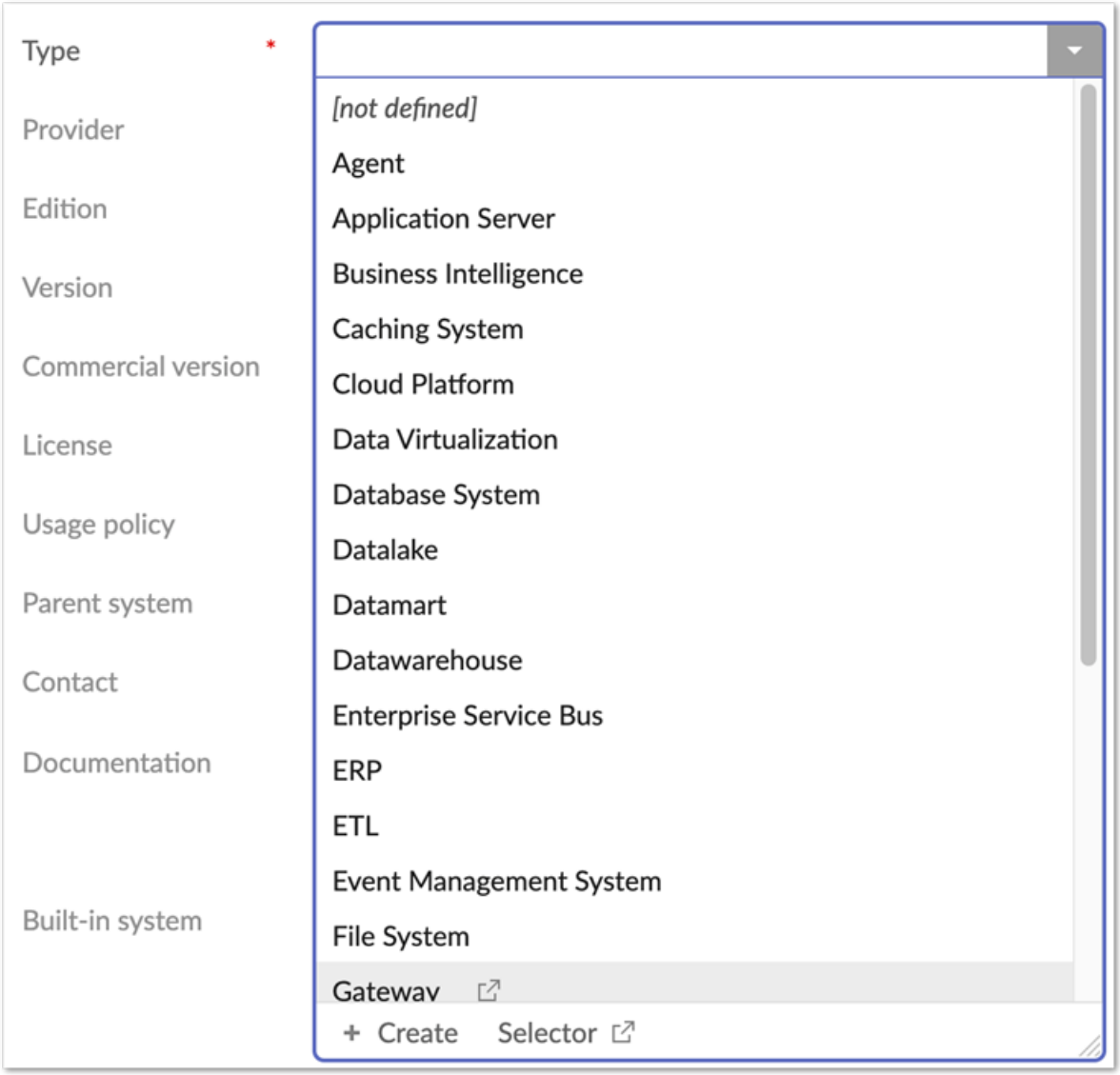
This chapter contains the following topics:

1. Create a new system type

59.1 Create a new system type

You can create other system types if built-in types are not sufficient for your usage.

On system creation/update page, on system type selection you can create a new system type



CHAPTER 60

Create an infrastructure

This chapter contains the following topics:

1. Create an infrastructure type
2. Create a new infrastructure type

60.1 Create an infrastructure type

The infrastructure types are used to classify infrastructures between families. In the **Technology** group in the **Main perspective**, select **Infrastructures**to access the infrastructure screen. When creating an infrastructure, you have to define its type.

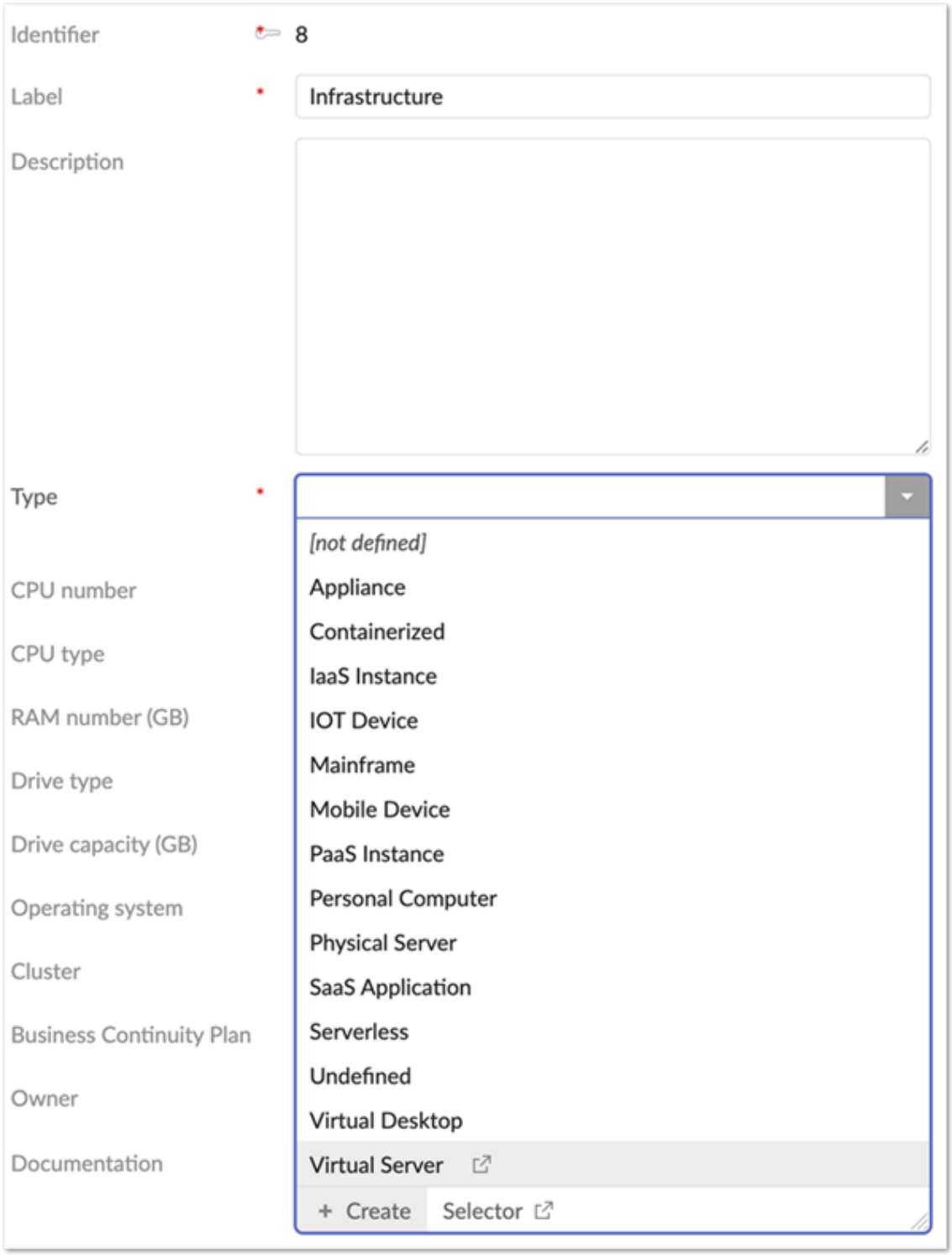
A few built-in infrastructure types are defined

- SaaS Application
- Paas Instance
- IaaS Instance
- Physical Server
- Virtual Server
- Containerized: use of Docker technology, for instance
- Serverless
- Appliance: Oracle Exadata, for instance
- Mainframe
- IOT Device
- Mobile Device
- Virtual Desktop
- Personal Computer
- Undefined

60.2 Create a new infrastructure type

You can create other infrastructure types if built-in types are not sufficient for your usage.

On infrastructure creation/update page, on infrastructure type selection you can create a new infrastructure type:



CHAPTER 61

Create a provider

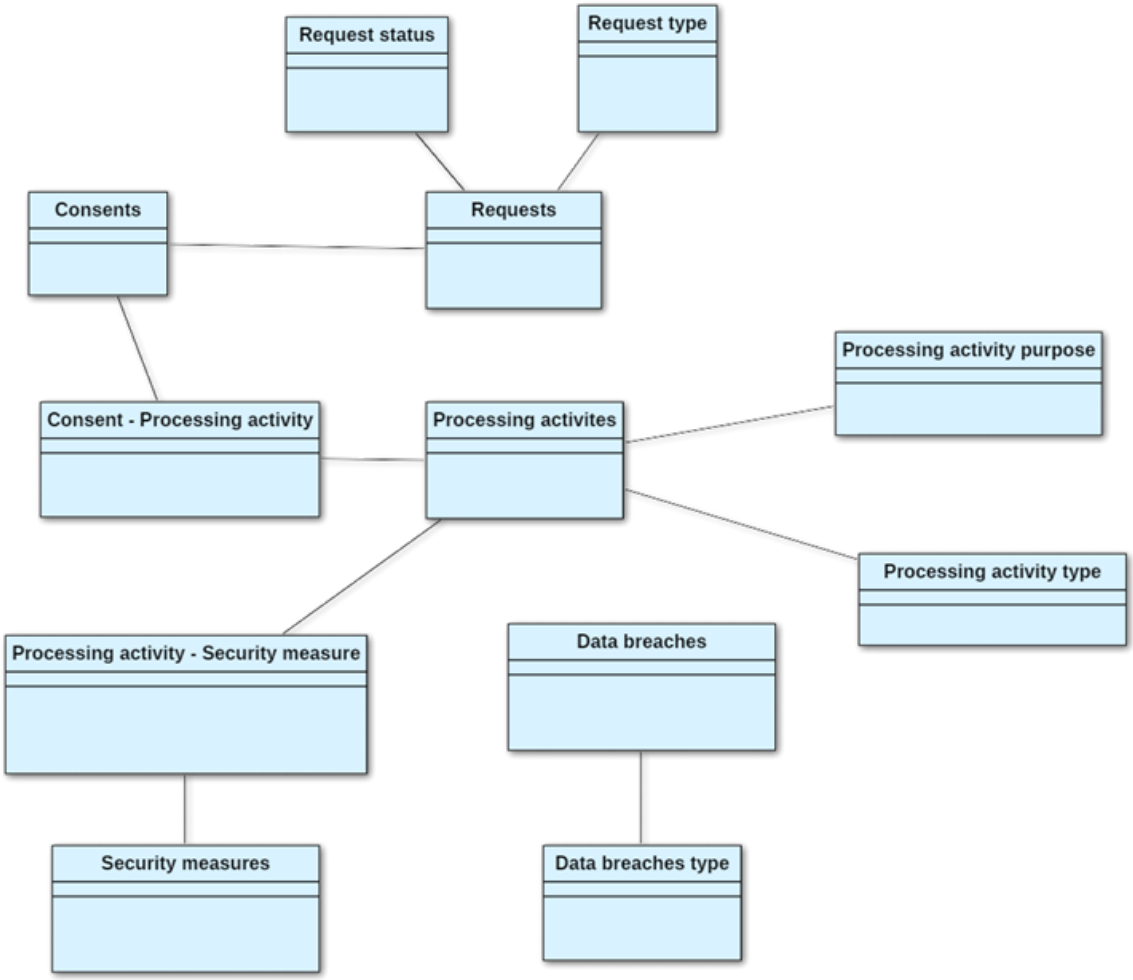
In the **Technology** menu in the **Metadata** perspective, select **Providers** to access the provider screen. The main page listing all existing providers displays.

CHAPTER 62

Privacy overview

The management of data privacy, as it is required by many regulation authorities, means keeping an inventory of processing activities and personal consents. It also requires you to register any request coming from personal data owners and communicate data breaches to them.

Connected to physical assets containing them, systems processing them and concepts defining their nature, all conditions are unified to ensure compliance with rules that their usage implies.



CHAPTER 63

Processing activities

Located in the **Metadata** perspective under **Privacy**, the **Processing activities** table registers all processing activities for your company. For regulation compliance purposes, it is important that companies can provide this registry of activities. This is especially true for activities related to personal information.

Each processing activity gathers information such as the purpose, the responsibilities, the impact assessment, the related data categories and personal data categories.

CHAPTER **64**

Consents

Located in the **Metadata** under **Privacy**, the **Consents** table allows you to track individual consent for given processing activities and personal data.

CHAPTER **65**

Data breaches

Located in the **Metadata** under **Privacy**, the **Data breaches** table handles reporting of detected data breaches. A breach concerns a given instance. Most regulation requires that a notification is submitted to supervisory authorities that includes: breach status, discovery date, submission date, impacted customers, etc.

CHAPTER **66**

Requests

Most data privacy regulations give individuals the right to request that a company disclose what data from the individual the company manages and processes. For instance, chapter III of the GDPR text lists the access, rectification, portability, erasure, processing restriction and object rights of a person. Located in the **Metadata** perspective under **Privacy**, the **Requests** table allows you to keep track of such requests and their progress.

CHAPTER 67

Workspace

This chapter contains the following topics:

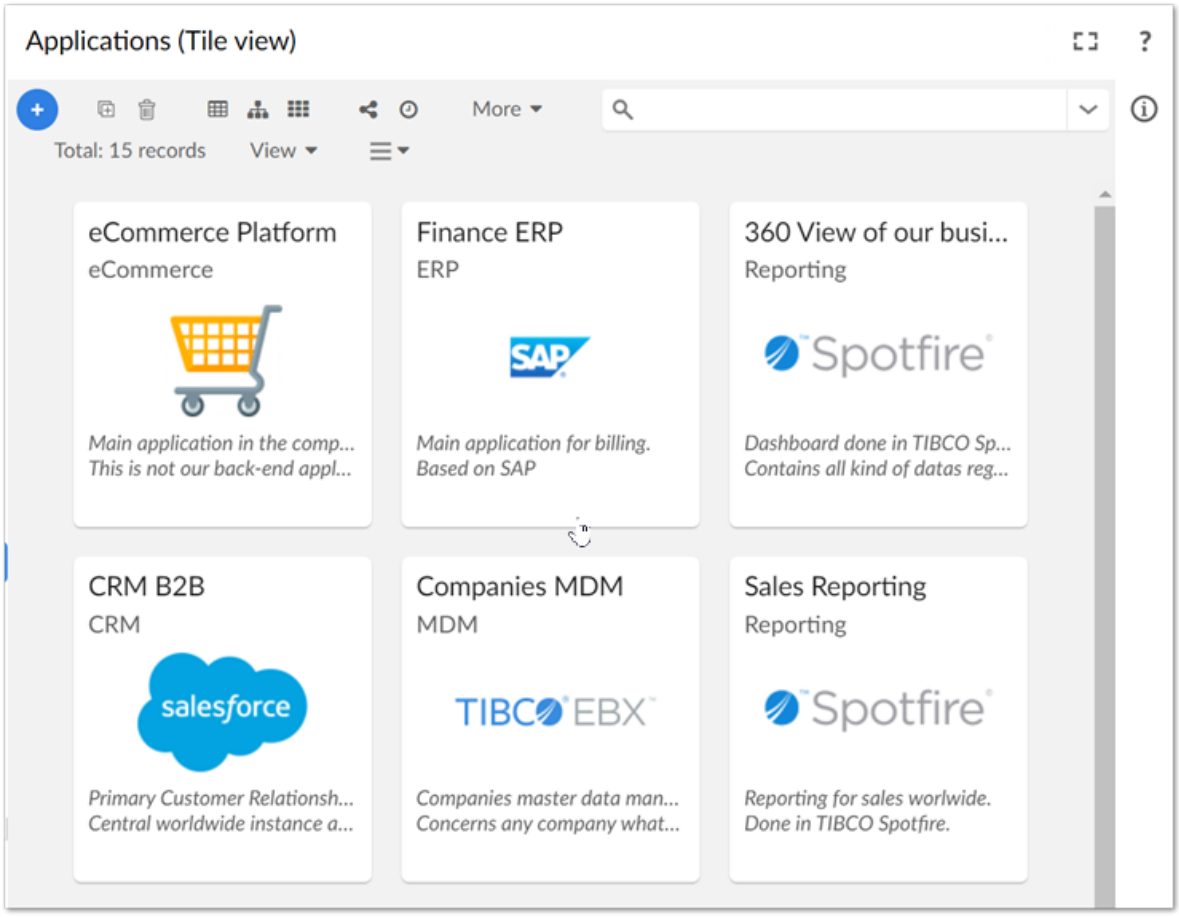
1. Overview
2. Data views

67.1 Overview

The workspace displays metadata and allows you to perform related actions. Except for the home section, all menu items display a default view to access a list of entities. A grid contains cards that represent an entity of the same kind, such as an application or a business term. A hierarchical view is used to provide a default view of assets.

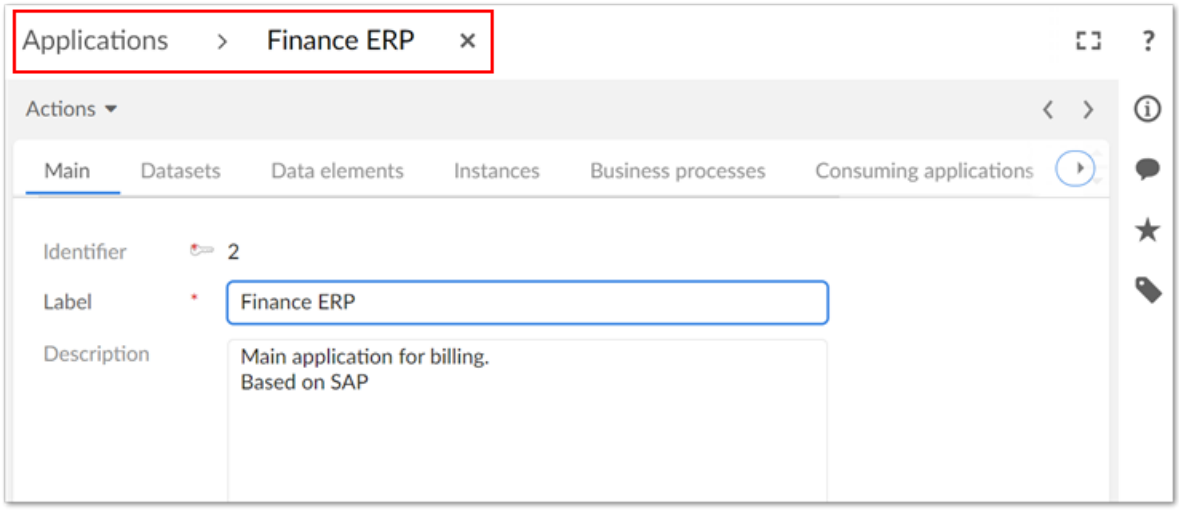
A toolbar displays in the upper left corner of the workspace. It allows an authorized user to create records by clicking on the plus icon. When the created item is subject to a workflow, this button

launches a creation workflow. This toolbar also allows you to switch to the default tabular view or to the default hierarchical view. Select the clock icon to access modification history for the current entity.



67.2 Data views

From an initial view, you can access other views and services to display metadata or perform actions. When browsing in views, a built-in breadcrumb allows you to navigate to previous pages. This is the recommended method of navigation as browser navigation buttons are not fully supported.



When in the **Metadata** perspective, you have access to all of the standard EBX® data views:

- Tabular view
- Hierarchical view
- Tile view
- Graphical view
- Details view
- Custom views you create [p 357]

CHAPTER 68

Home page

This chapter contains the following topics:

1. Overview
2. Home page widgets
3. Customize your home page

68.1 Overview

The **Home page** displays a global metadata view using widgets that allow quick access to services and data.

As shown below, the first time you access the **Home page** you can choose to:

- Use the default home page: Selecting this option displays the default home page, which you can customize later. See customize your home page [p 350] for more information.
- Create your home page: Selecting **Configure my home page** displays an unpopulated home page and an **Add widget** button in the top left corner. Use this button to choose from and add available widgets to your home page. Use the **Save** button to save changes.

68.2 Home page widgets

The Home page displays information on widgets. You can create your own widget from a widget type or use a pre-configured widget. The following section describes the types of widgets available.

Widget types

There are multiple widget types available. For example, there is a widget for quick access to search globally, another one for quick access to a specific table. The following sections describe the available widgets.

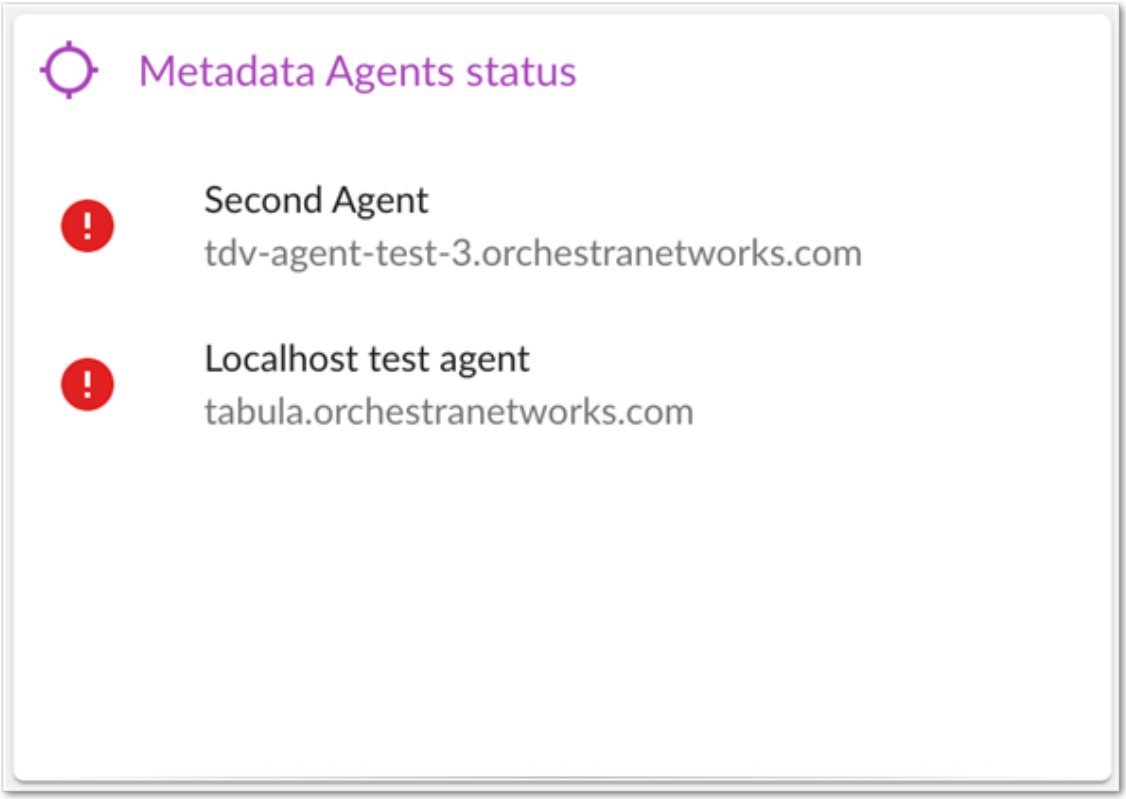
Global search widget

This widget displays a search bar where you can enter search terms to search using the Global search service. Configuration options are not available for this widget.

Metadata agent status widget

This widget displays metadata agent status. You can see if an agent is connected and, if not, the reason why. This widget includes the following configuration options:

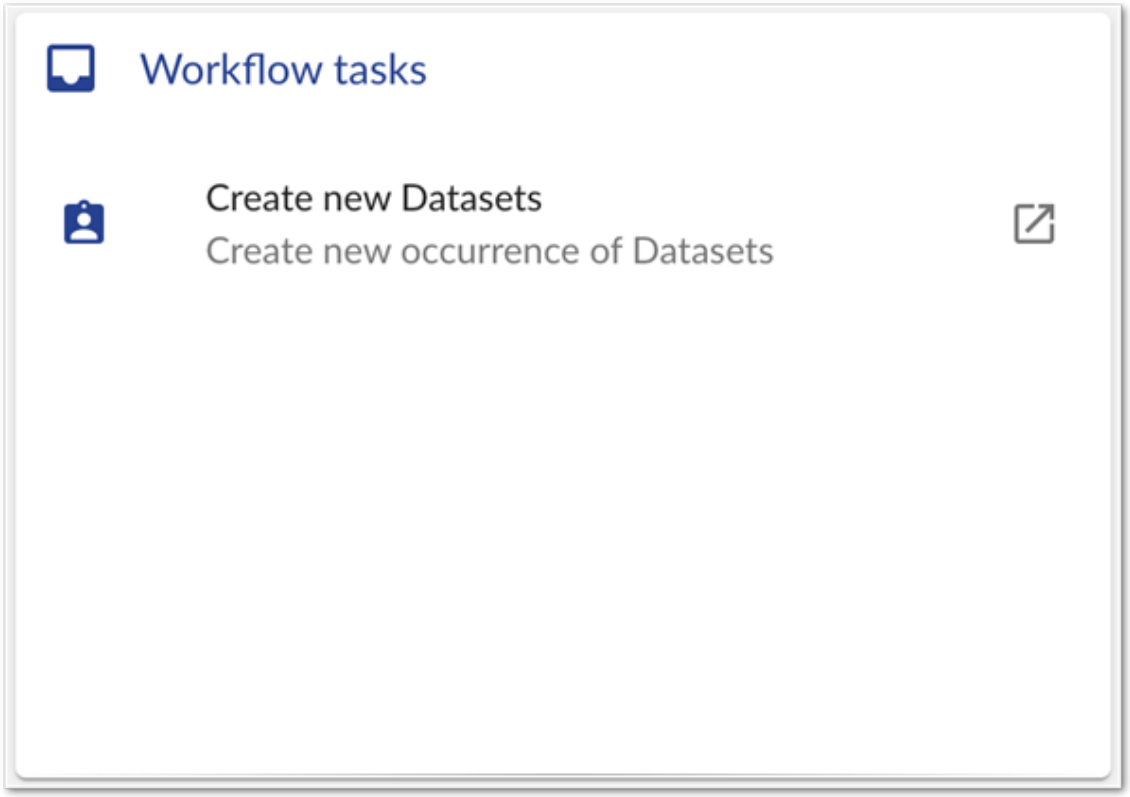
- Title: widget title
- Description: widget description
- Agent: Optional parameter to limit the displayed agents to one or many selected agents



Workflow tasks widget

This widget displays the workflow tasks that are waiting for you or offered to you. From this widget you can click on a task to access the workflow view or the workflow graph view. If a task is offered

to you but not already allocated, a pop-up message displays to inform you that it will be allocated to you. Configuration options are not available for this widget.

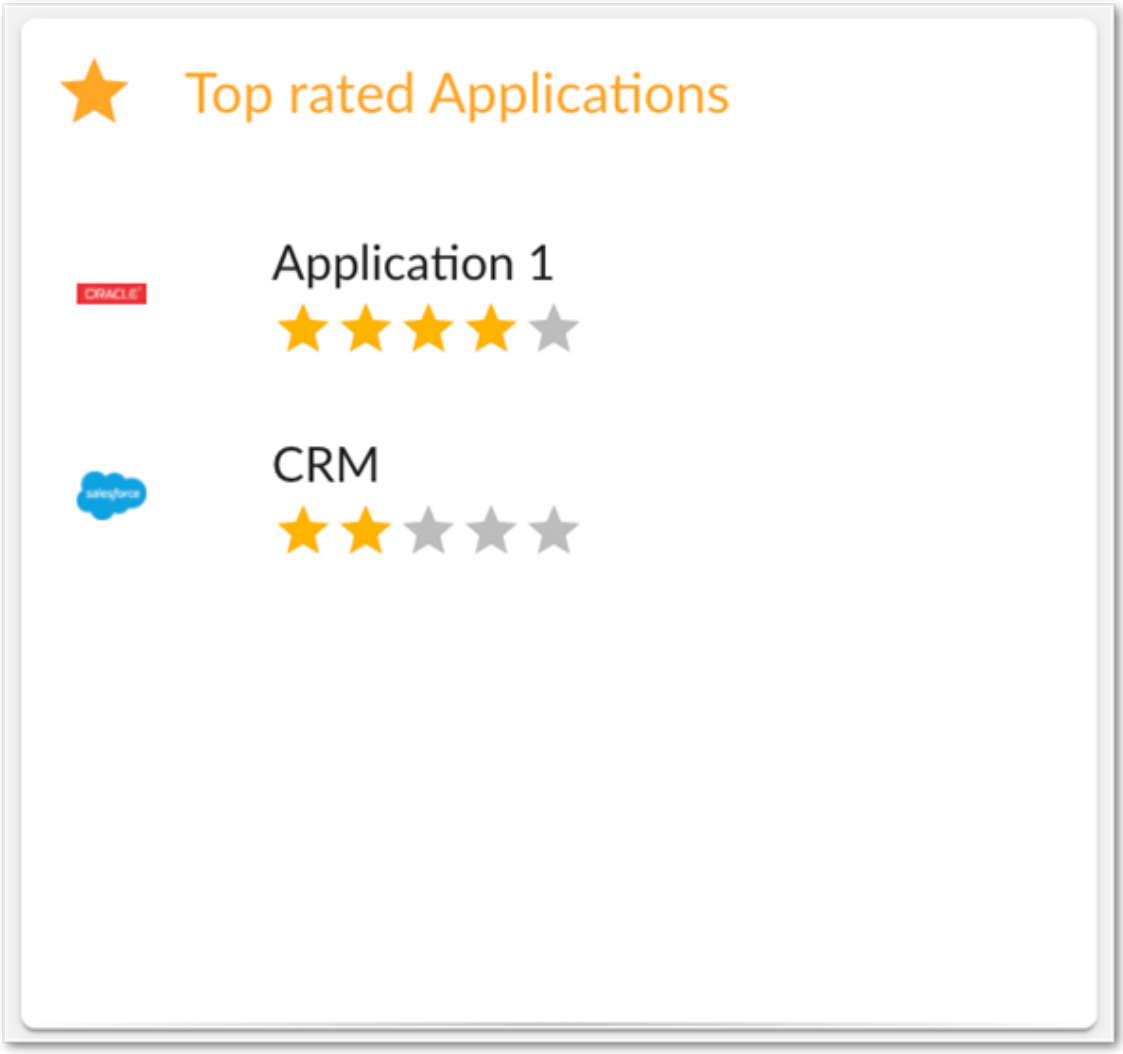


Number of elements widget

This widget displays an entity's top rated elements. You can click on an element to access it. If no element is rated, this widget displays an empty sign. This widget includes the following configuration options:

- Title: widget title
- Description: widget description
- Entity: select the entity you want to display
- Number of elements: number of top rated elements to be displayed

- Predicate: Xpath predicate format to filter the selected entity

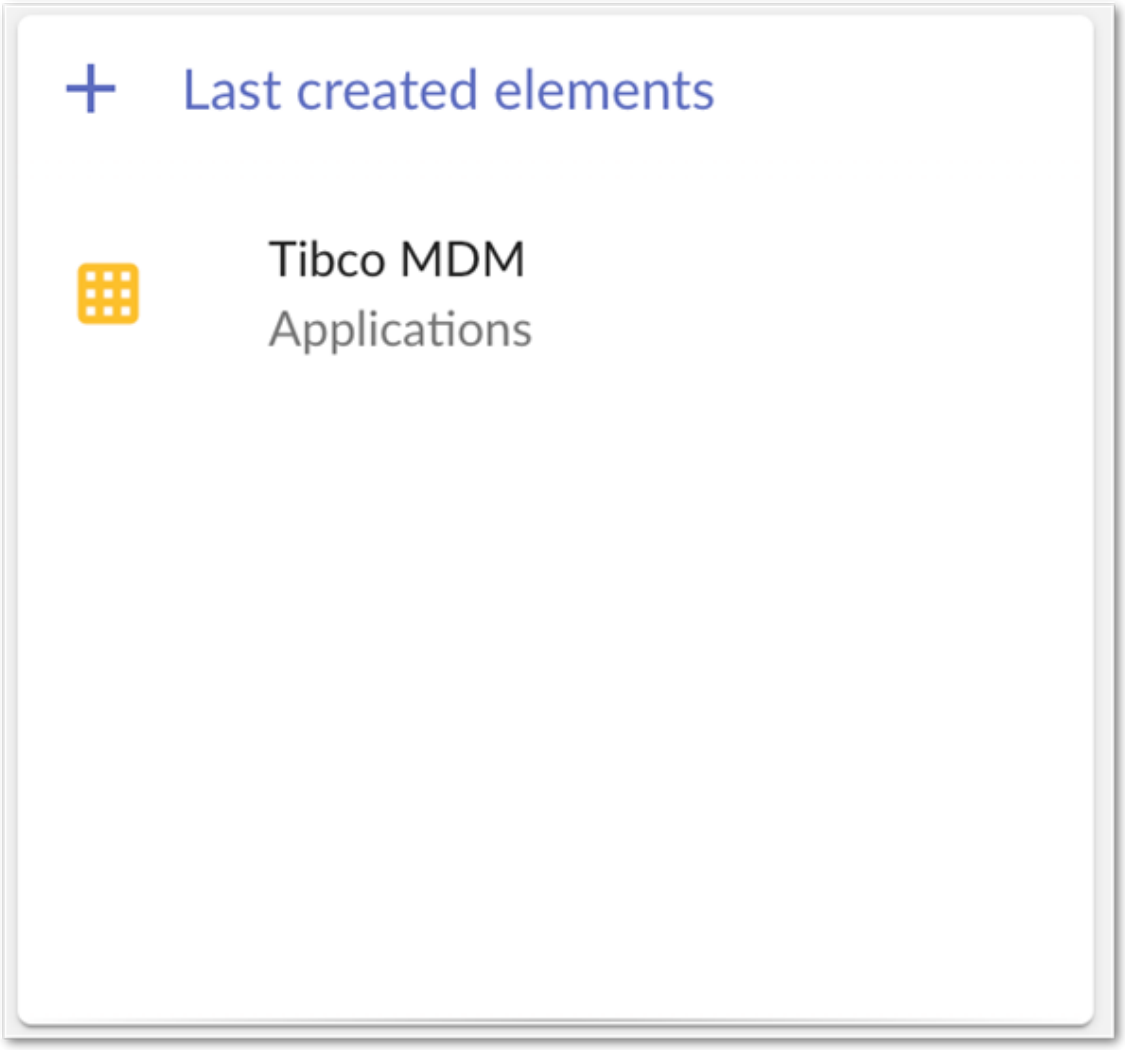


Last created elements

This widget displays last created elements from selected entities. You can access the element by clicking on it. Hover your mouse over an element to display a tooltip that includes the creation time and creator's user name. This widget includes the following configuration options:

- Title: widget title
- Description: widget description
- Entity: select one or more entities that you want to display

- Number of elements: maximum number of elements to display

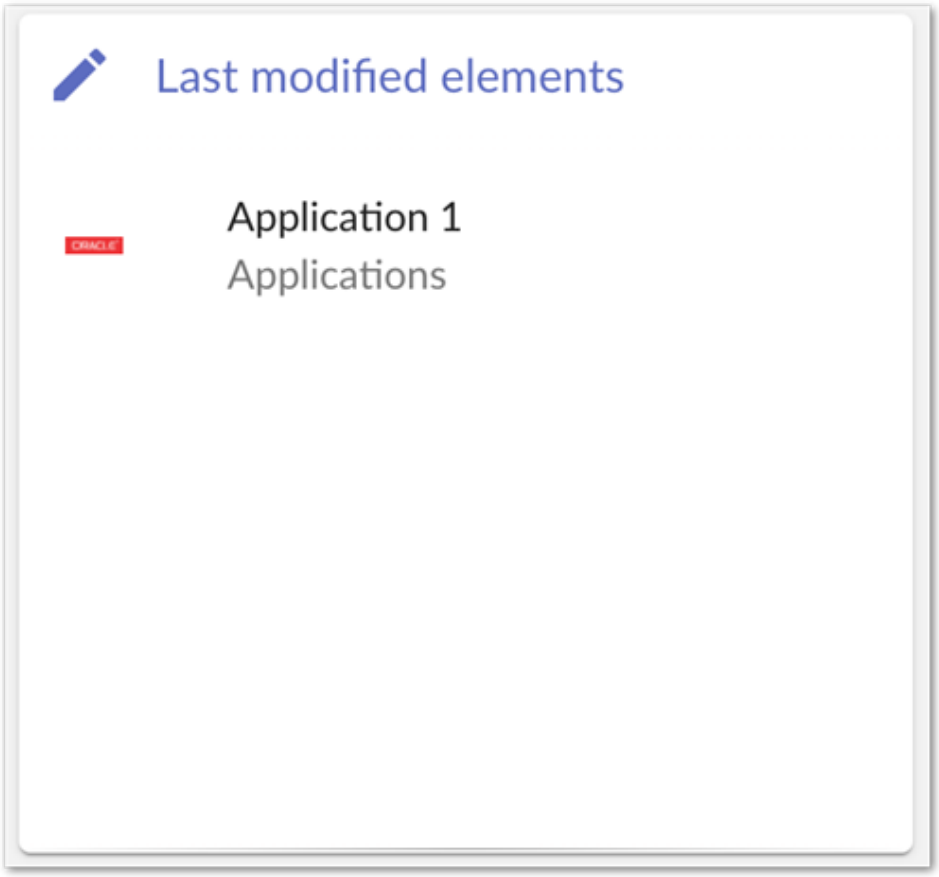


Last modified elements

This widget displays the most recently modified elements from selected entities. You can access the element by clicking on it. Hover your mouse over an element to display a tooltip that includes the modification time and and the user name that made the modification. This widget includes the following configuration options:

- Title: widget title
- Description: widget description
- Entity: select one or more entities that you want to display

- Number of elements: maximum number of elements to display by entity



Widget “Library” and “My widget”

On the **Add widget** dialog you can choose:

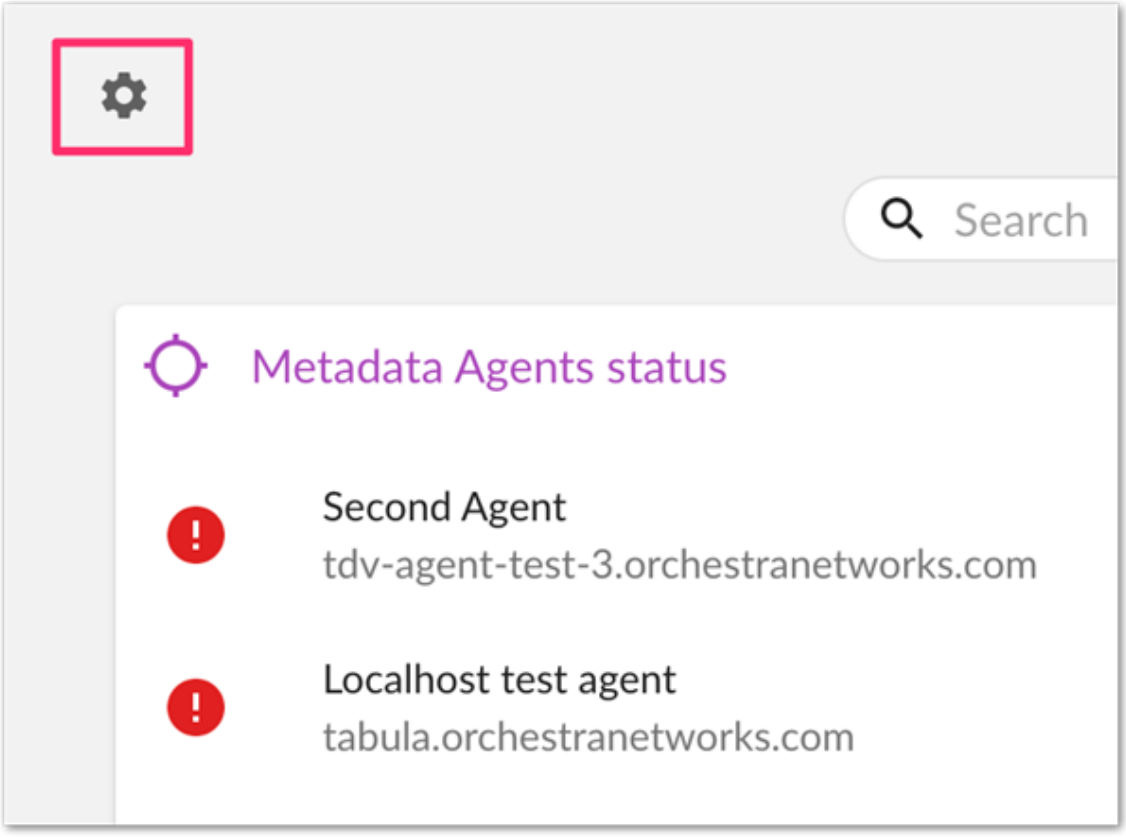
- The **Library** tab to display pre-configured widgets or shared widgets.
- The **My widget** tab to display your widgets.

Create/Duplicate/Update/Delete a widget

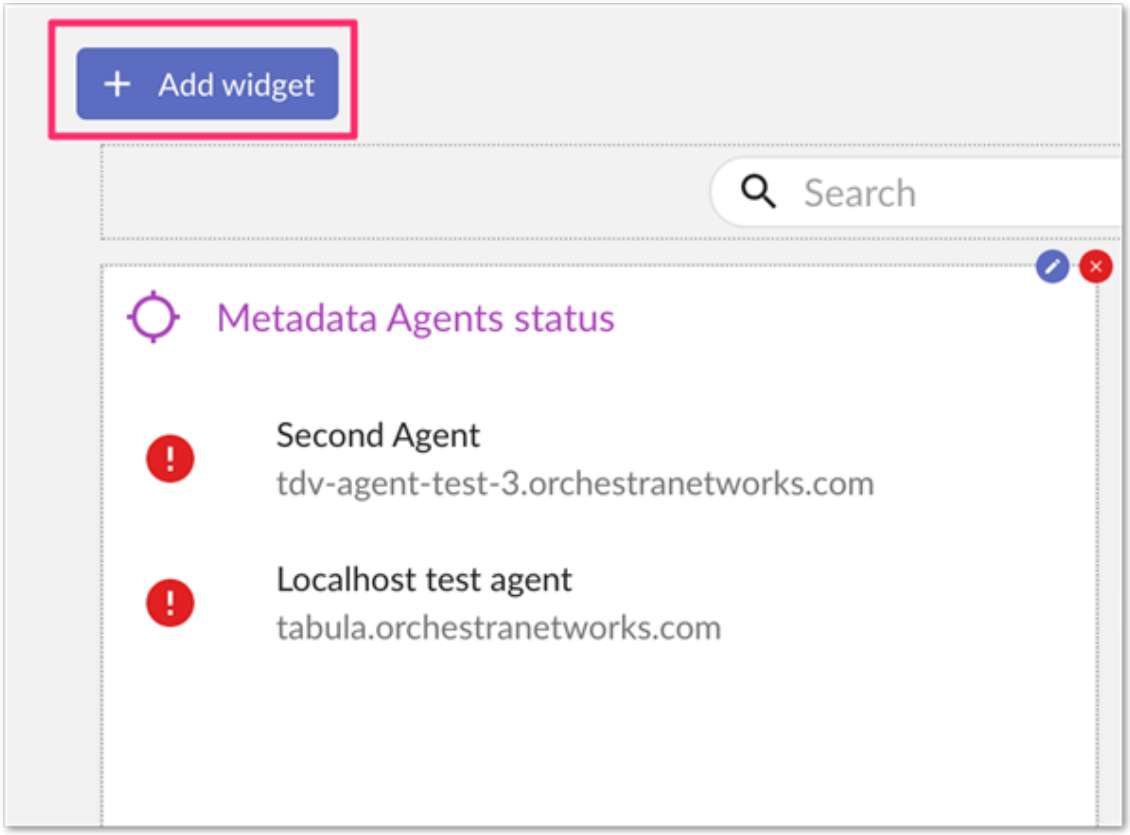
You can create any widget type you want. You can duplicate an existing widget from your widget library or from the shared ones. You can only update or delete your own widgets.

Create a widget

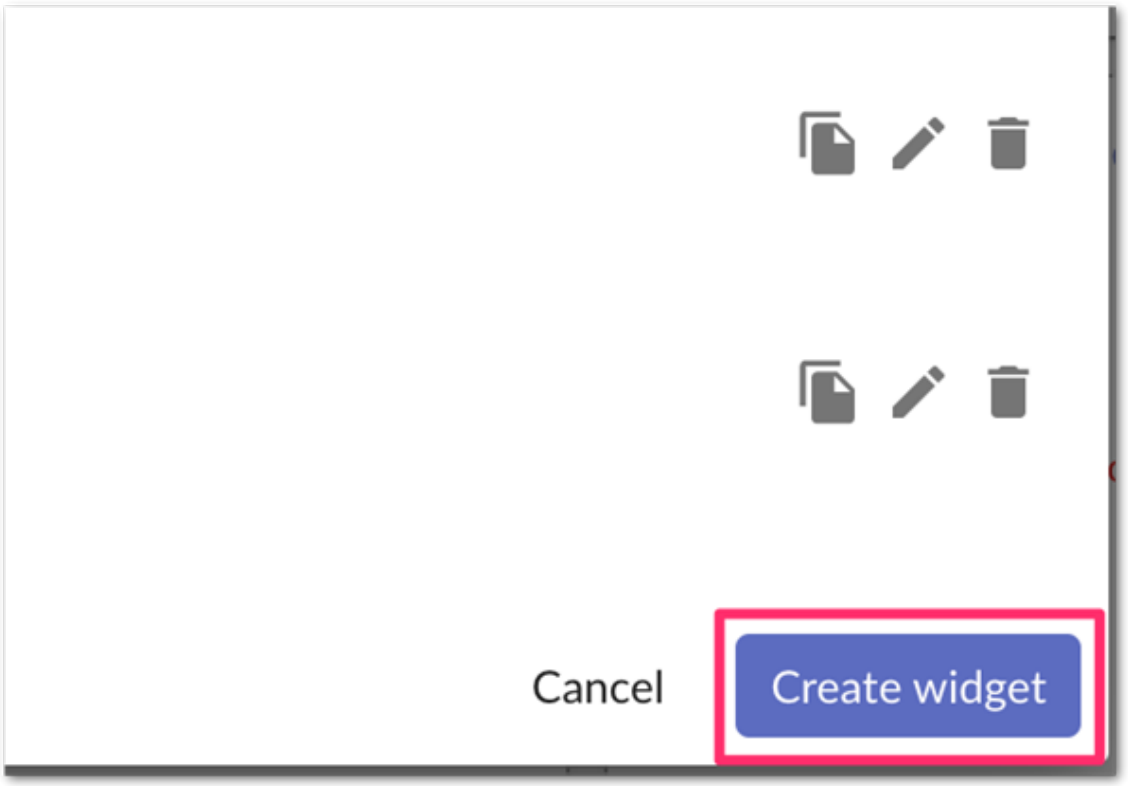
To create a new widget, click on the gear icon in the top left corner of the home page.



You are now in **home page edit mode**. Click on **Add widget** in the top left corner.



A dialog displays that allows you to create a widget. Select **Create widget** to open the widget configuration wizard and begin the creation process.

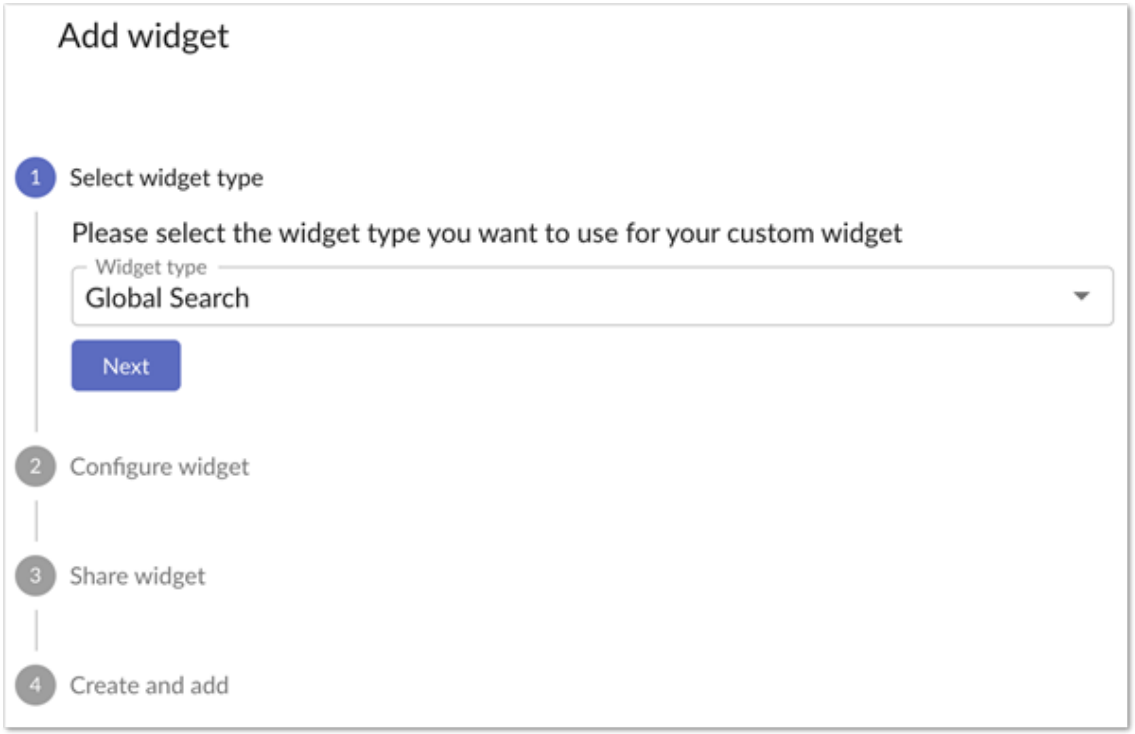


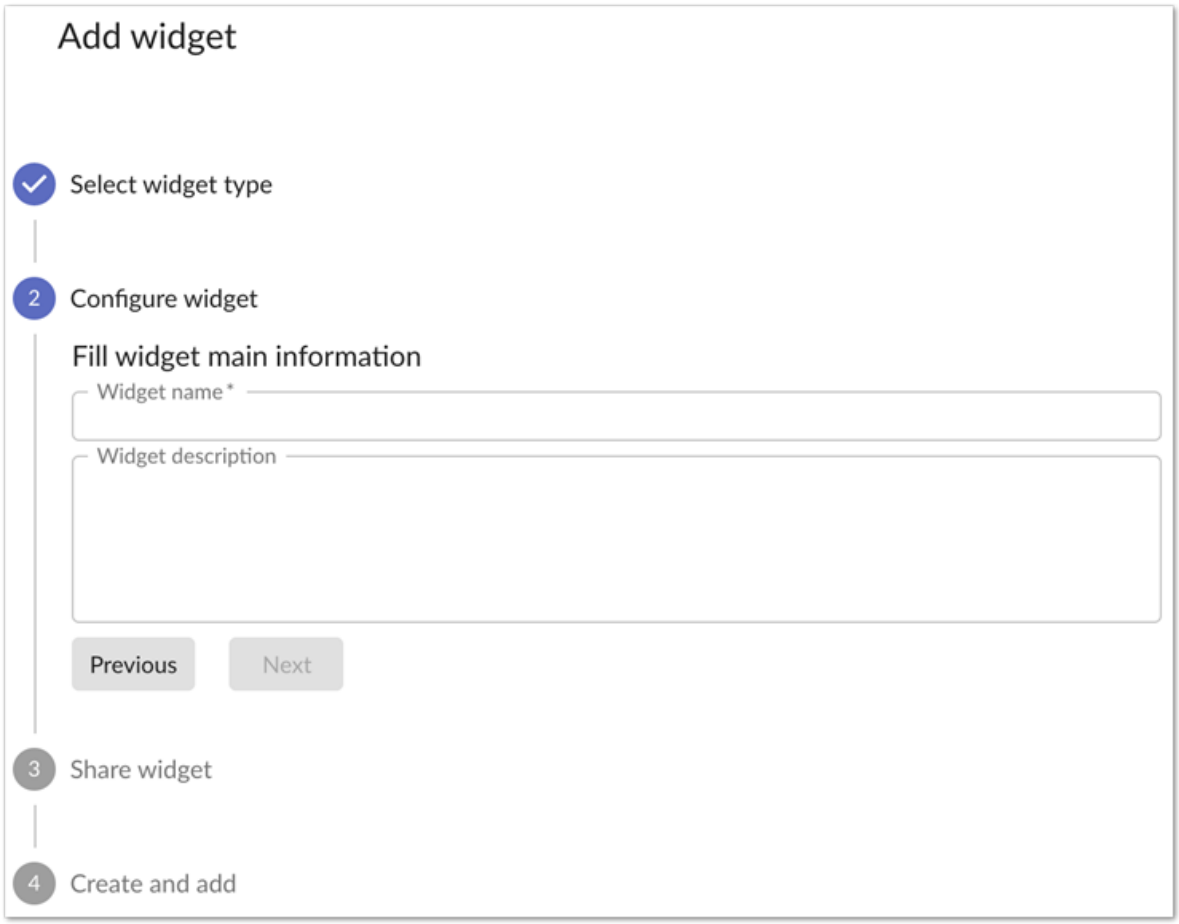
The widget configuration wizard includes the following steps:

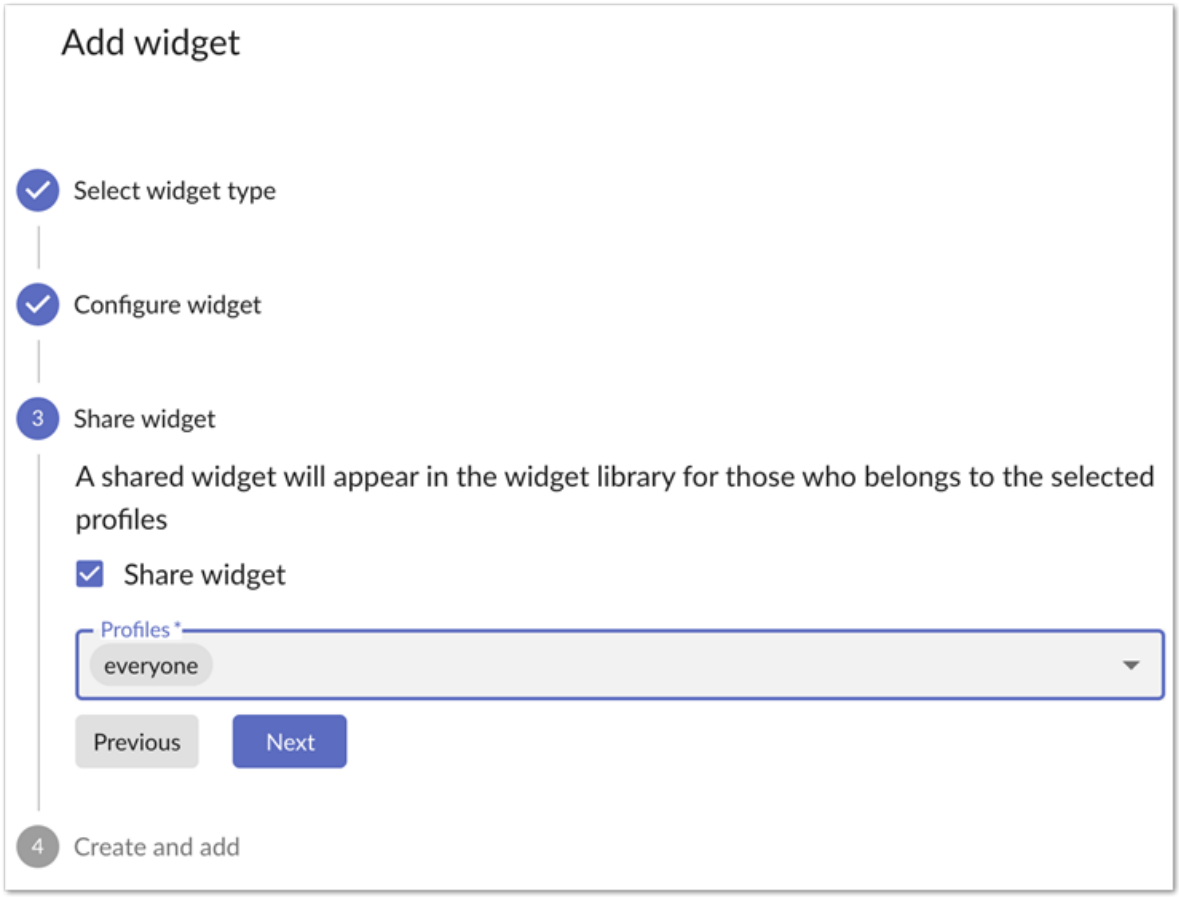
- Select the widget type.
- Configure widget: configure widget values and specific widget type attributes.
- Share widget: configure the widget sharing parameter. A shared widget is available for other users.
- Verify widget: verify widget information and process creation.

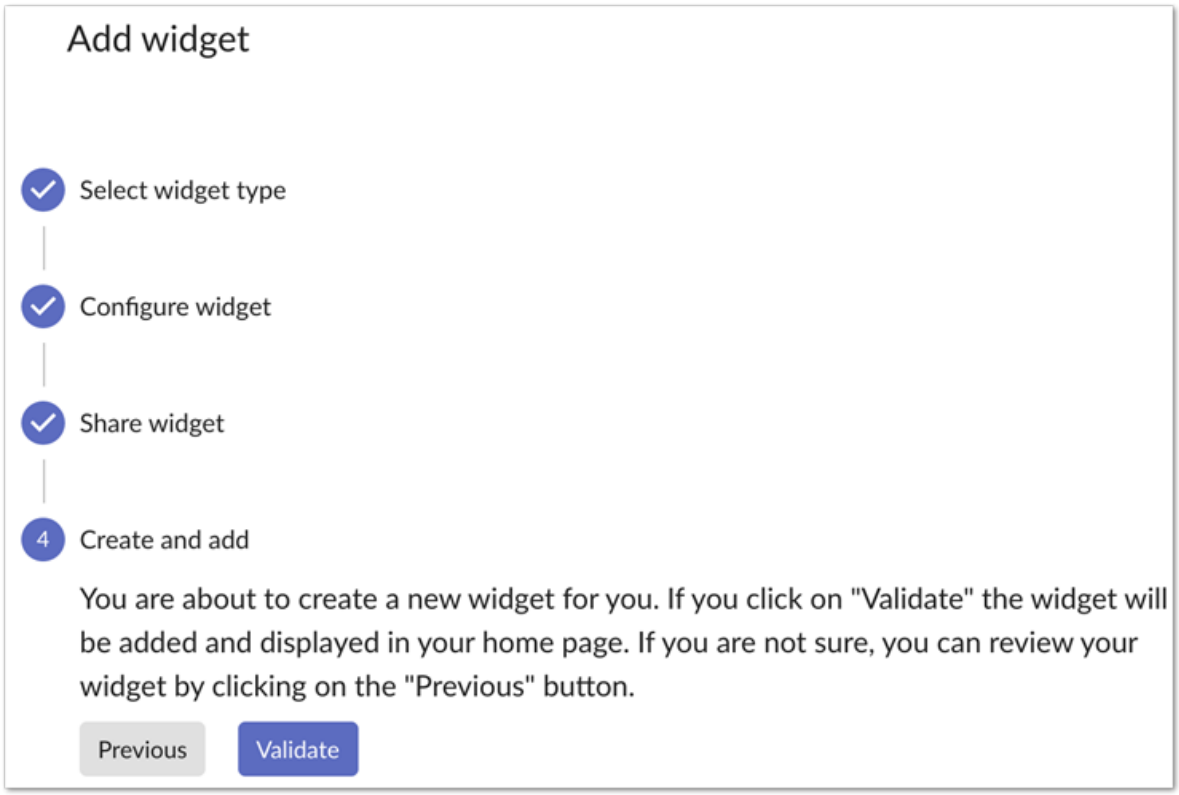
> **Note**
>
> You must enter all required information at each step to move to the next. The **Next** button is disabled until you do.

The following sequence of images shows a sample widget creation:






Add widget
Select widget type
Configure widget
3
Share widget
A shared widget will appear in the widget library for those who belongs to the selected profiles
Share widget
Profiles *
everyone
Previous
Next
4
Create and add


Add widget
Select widget type
Configure widget
Share widget
4
Create and add
You are about to create a new widget for you. If you click on "Validate" the widget will be added and displayed in your home page. If you are not sure, you can review your widget by clicking on the "Previous" button.
Previous
Validate

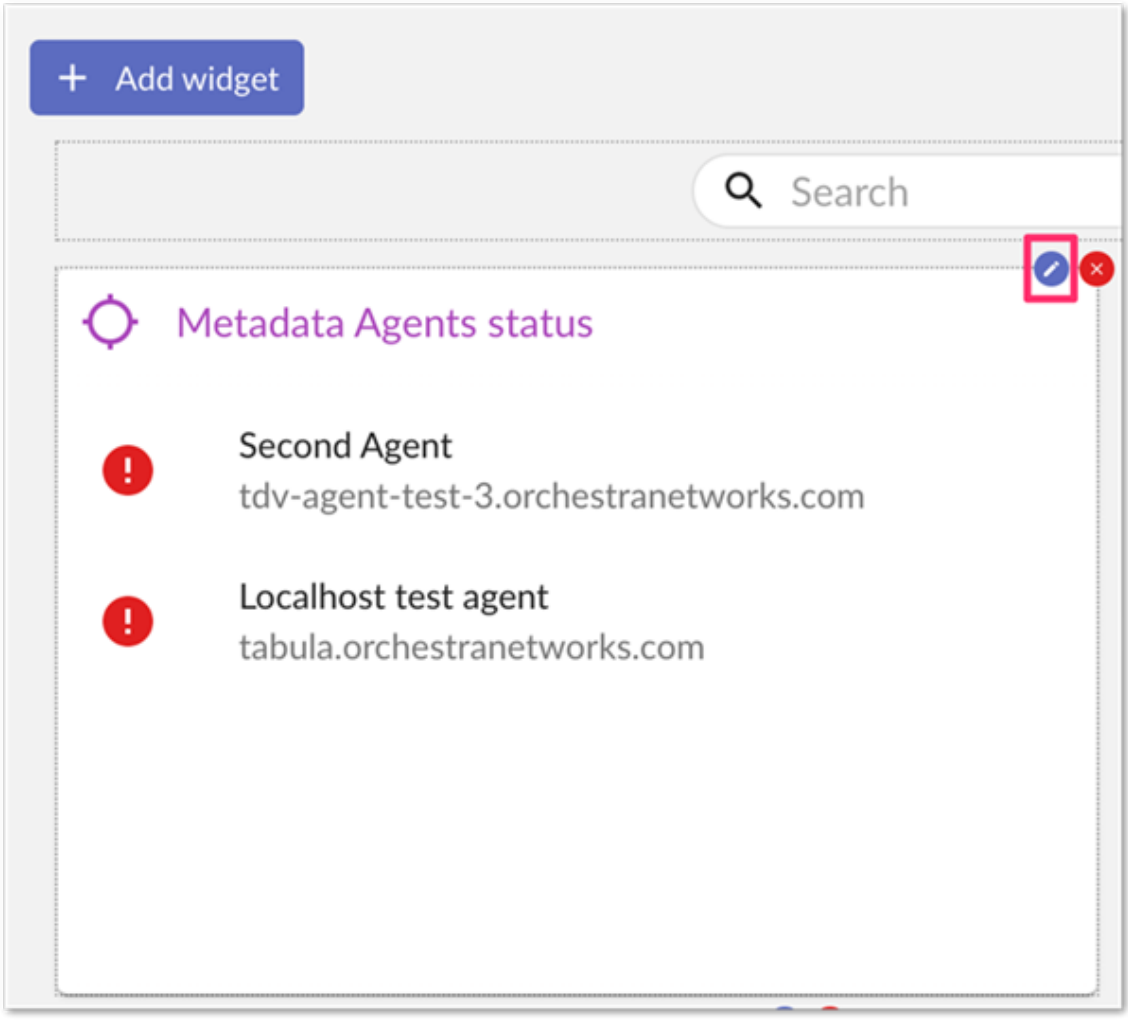
Modify a widget

When you modify a widget, a simplified version of the widget configuration wizard allows you to modify: type attributes, the sharing configuration, and verify and apply the changes.

To modify widget:

If the widget you want to edit is on your **Home page**:

1. Select the gear icon to edit the **Home page**.
2. Select the edit icon on the widget you want to modify.
3. Complete the steps in the wizard.



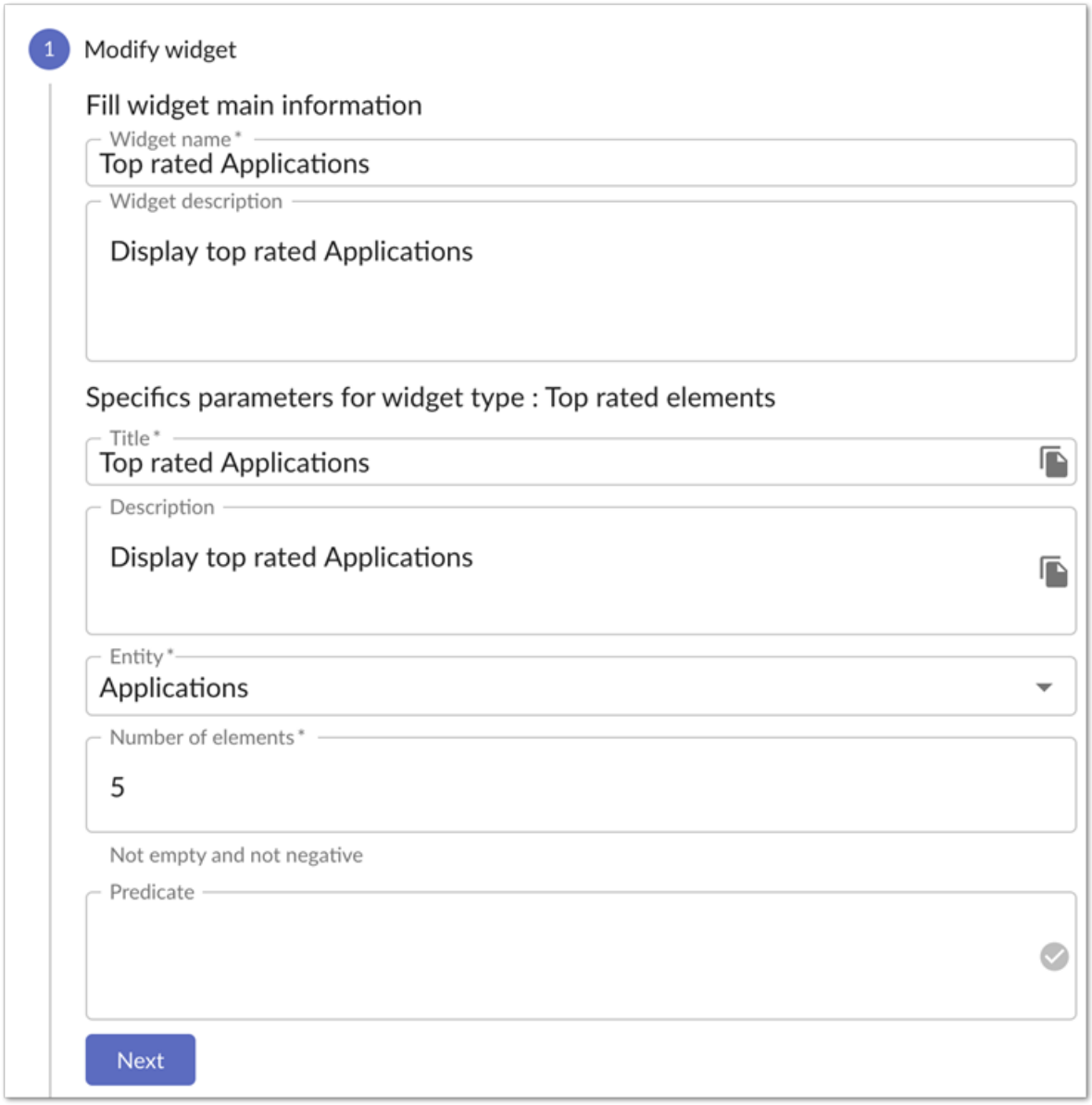
If the widget you want to edit is not on your **Home page**:

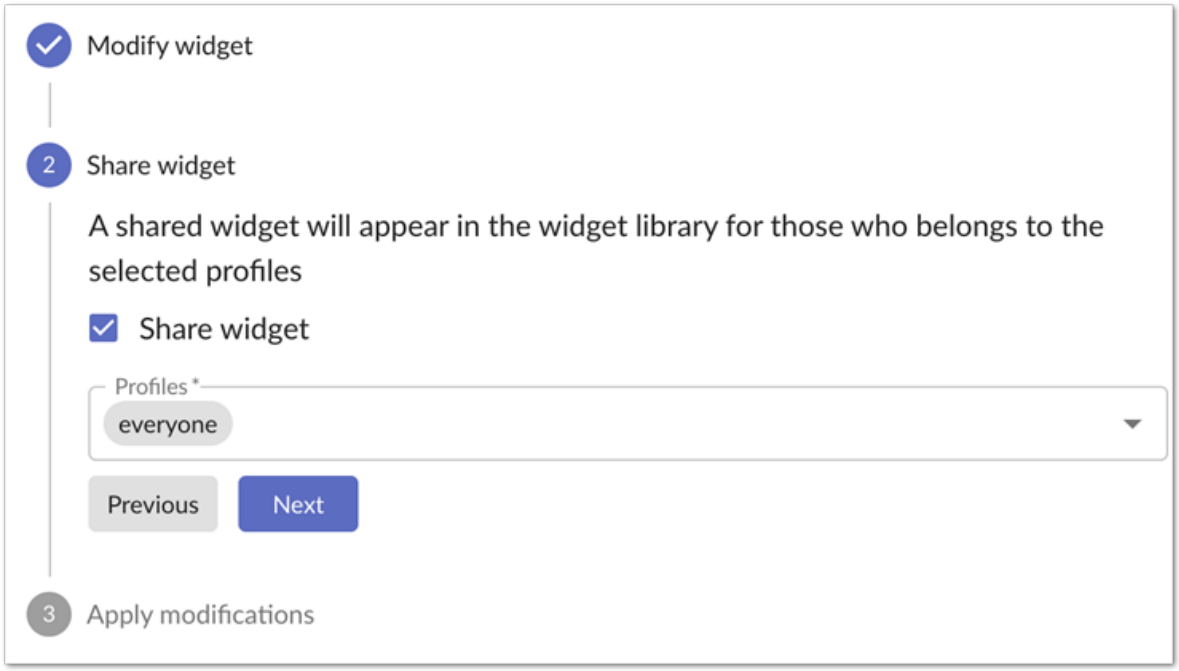
1. Select the gear icon to edit the **Home page**.
2. Select **Add widget** and in the dialog that displays, select the **My widget** tab.
3. Select the edit icon on the widget you want to modify.

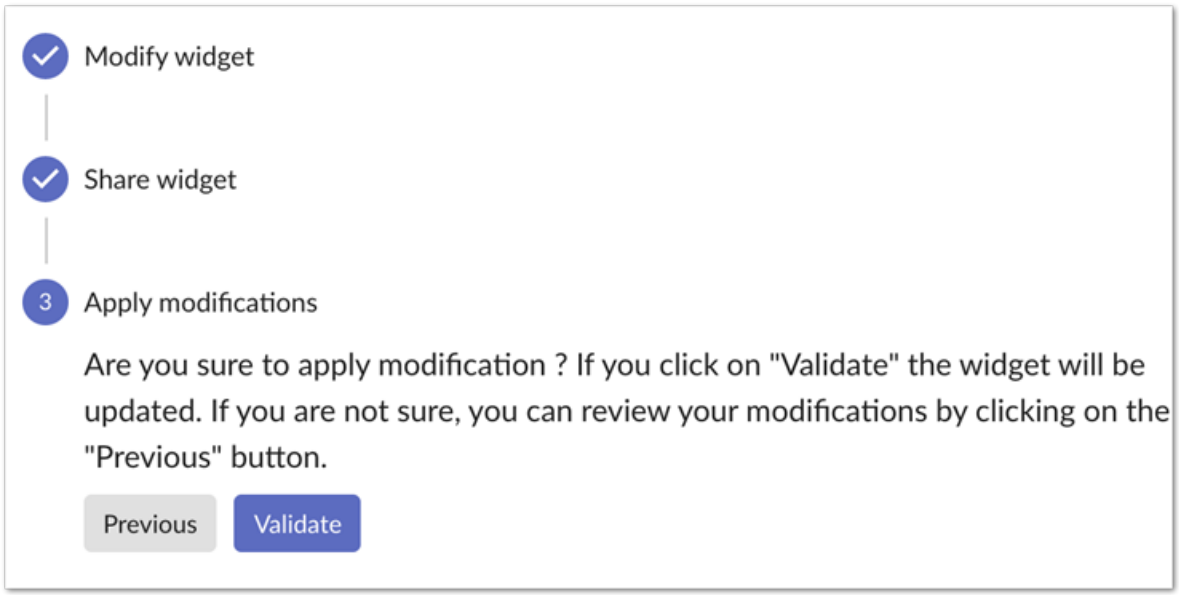
4. Complete the steps in the wizard.



The following sequence of images demonstrates the widget modification process.





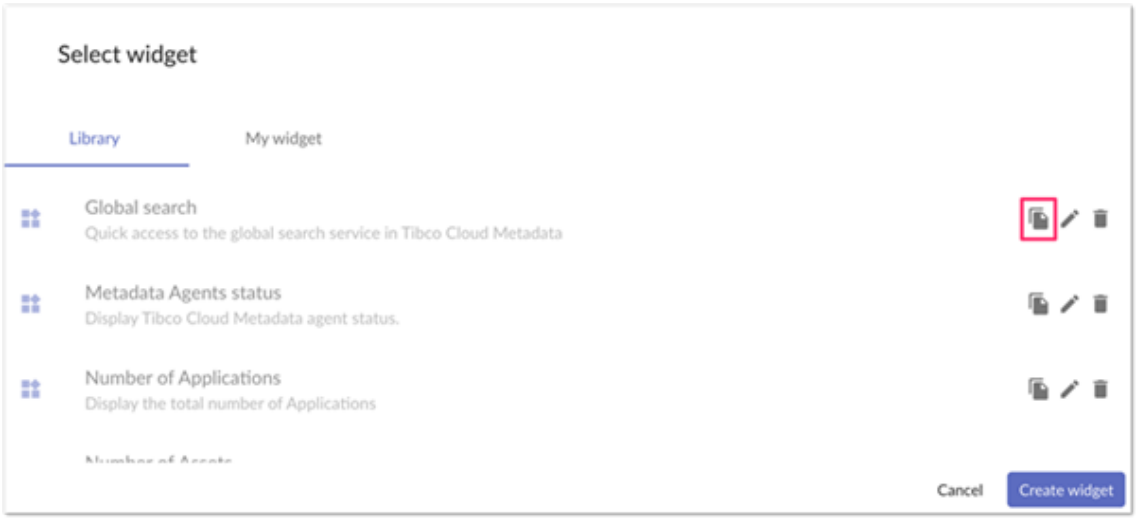


Duplicate a widget

To duplicate an existing wizard:

1. Select the gear icon to edit the **Home page**.
2. Select **Add widget**.
3. Choose the desired widget from the library or your collection and select its duplicate icon. The configuration wizard displays.

4. Complete the steps in the wizard to duplicate the widget and add it to your current home page.



Delete a widget

You can only delete widgets that you have created. To delete a widget:

1. Select the gear icon to edit the **Home page**.
2. Select **Add widget**.
3. Select the **My widget** tab.
4. Select the delete icon on the desired widget. The widget is deleted after you confirm the action.



Note: if your widget was shared or was in use in your home page, this action will remove it for you and all other users.

68.3 Customize your home page

If you prefer to display a specific widget instead of another you can do it thanks to home page customization.

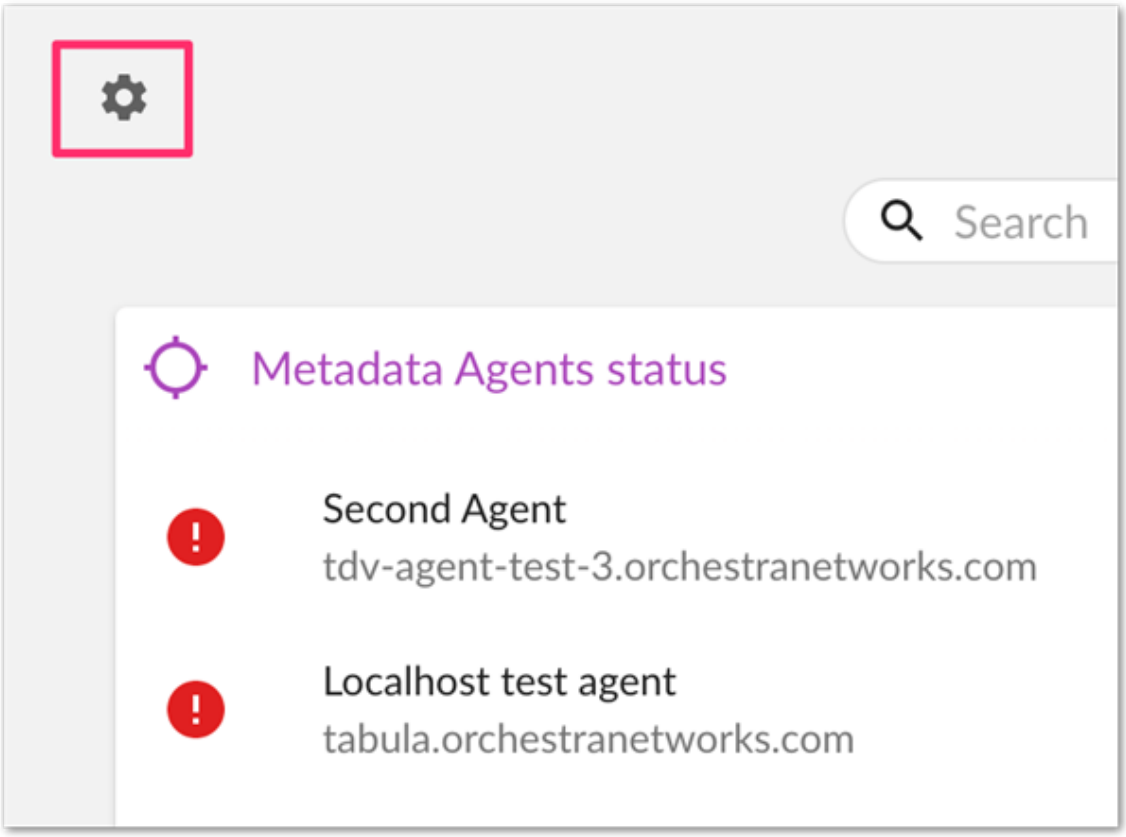
If you start from the default home page or from a blank one, you can:

- Modify widget position [p 351]
- Add/Remove pre-configured widget [p 352]
- Add/Update/Remove a widget created by you [p 355]

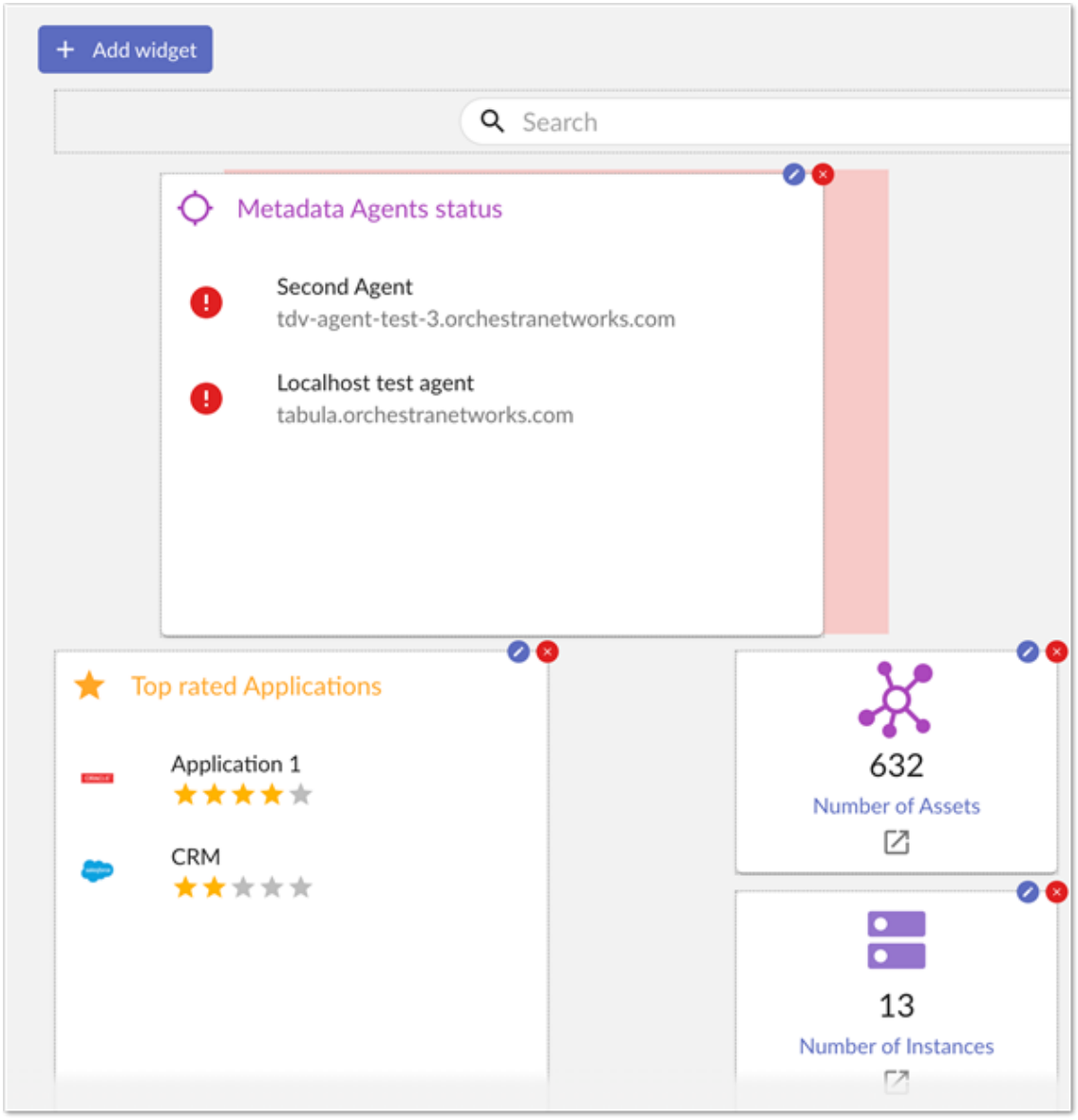
Modify widget position

To move a widget:

1. Select the settings icon.



2. Select and drag a widget to the desired location.

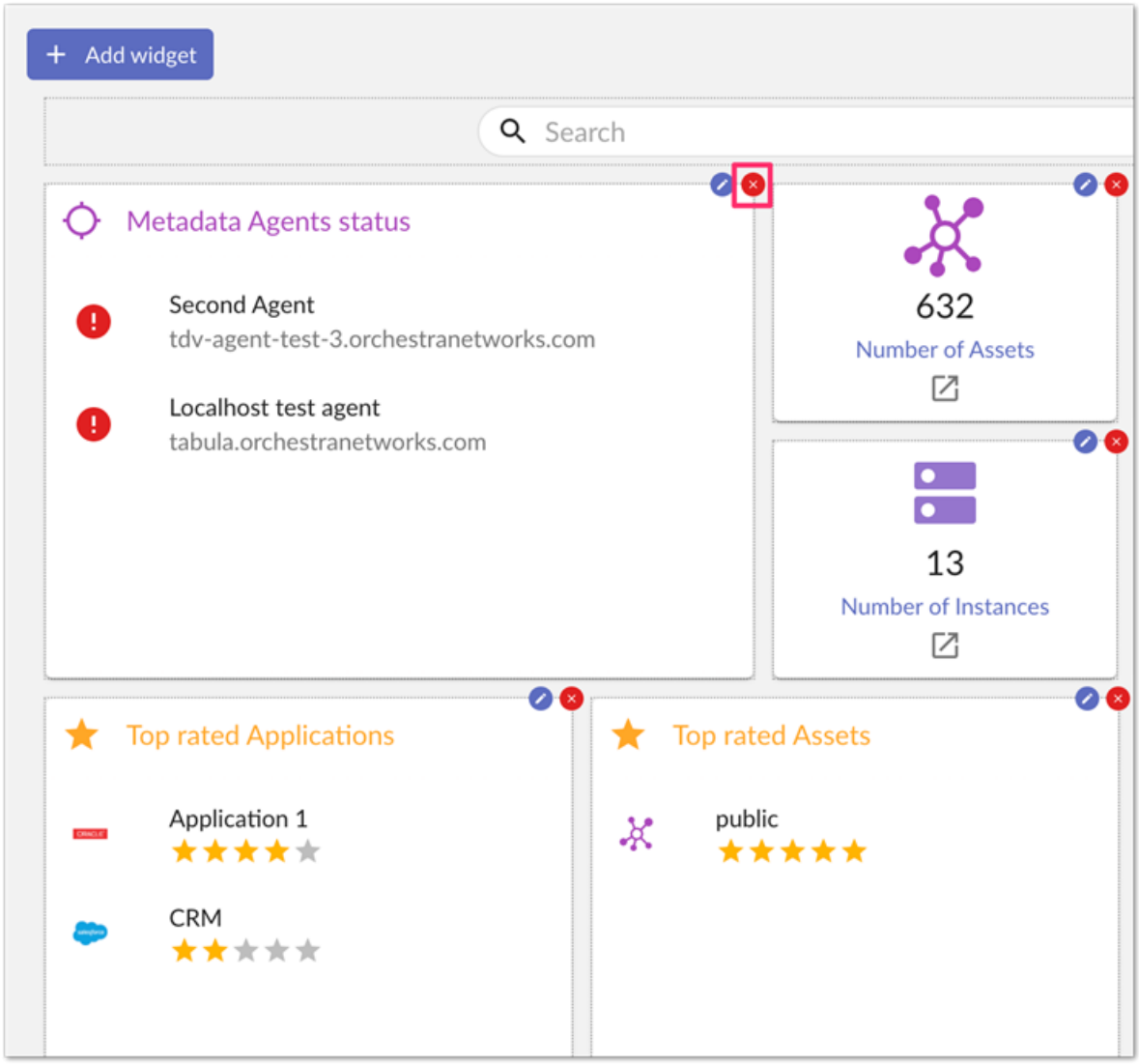


3. Save your modifications.

Add/Remove pre-configured widget

To add and remove widgets, enter **Home page** edit mode by selecting the settings icon.

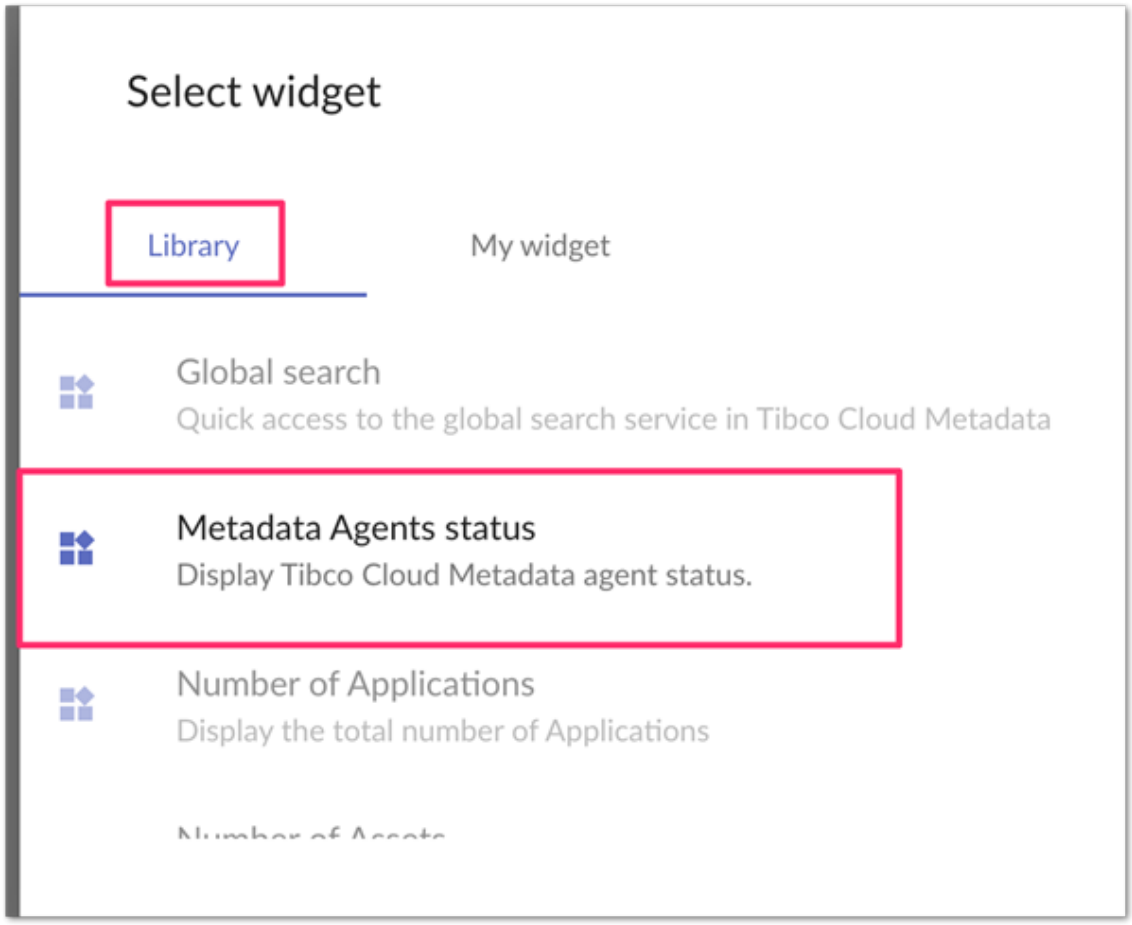
To remove a widget, select its close icon.



To add a widget:

1. Select **Add widget**.

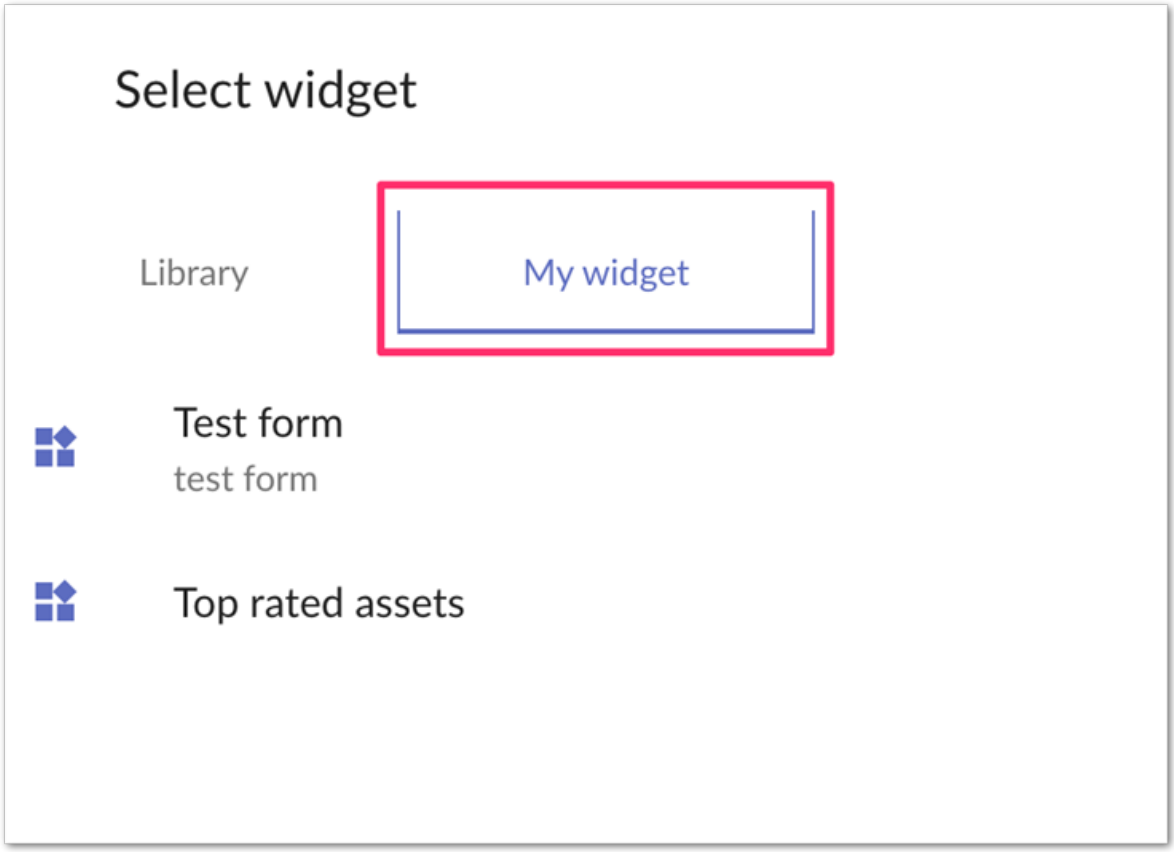
2. From the **Library** tab, select the widget you want to add.



> **Note**
>
> Widgets are added to the end of your home page. Once it is added, you can move to the desired location. You can only add a widget that is not already on your home page.

Add/Update/Remove a widget created by you

Use the steps in the previous section, but select the **My widgets** tab in the **Add widget** dialog box.



CHAPTER 69

Custom views

This chapter contains the following topics:

1. Overview
2. Tabular views
3. Hierarchical views

69.1 Overview

You can customize views in the EBX® Metadata Management application based on the requirements of target users. You can create the following types of views:

- Tabular: A table view to sort and filter the displayed records.
- Hierarchical view: A tree view that links data in different tables based on their relationships.

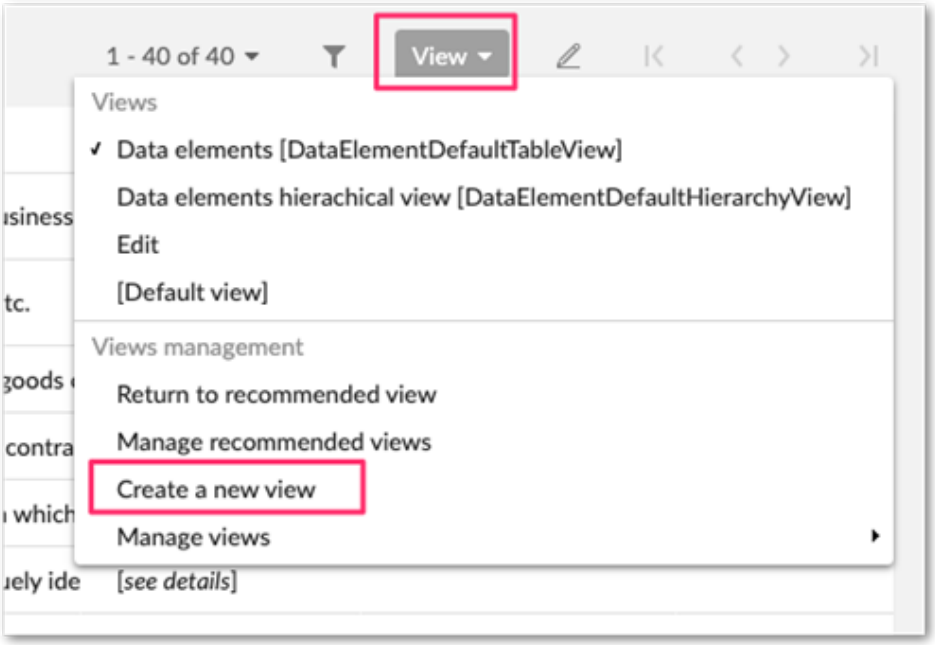
69.2 Tabular views

In the table view mode, select the **View** button on the top right corner. The menu that displays allows you to:

- Apply an existing view by clicking on the view's name.
- Manage existing views, if you have sufficient permission.
- Create a new view.

Creating a new view

To create a new view, select **View**, then **Create a new view**.



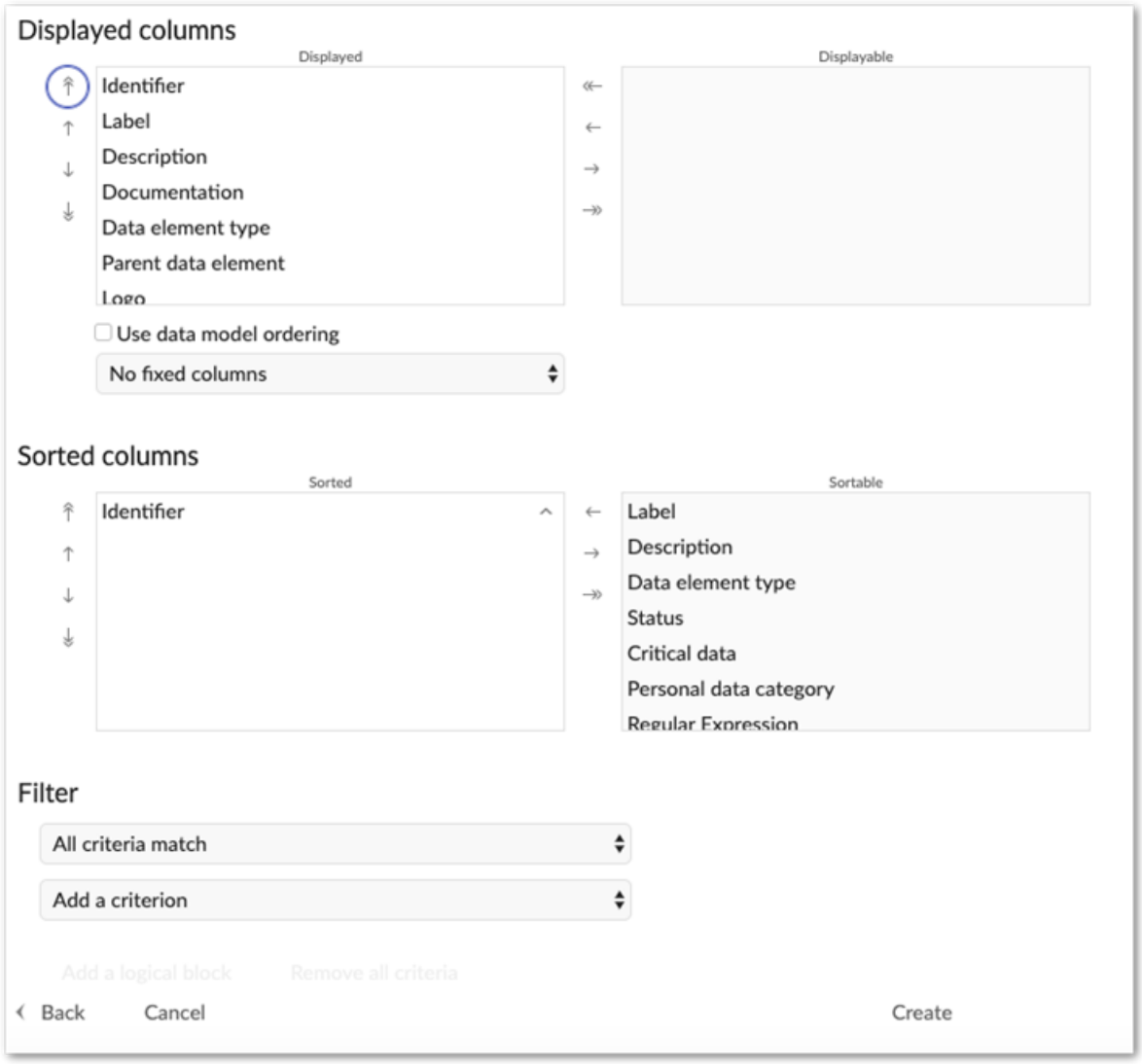
In the dialog box that displays, you can:

- Set the name
- Set the owner
- Set profile/user to share the view with
- Set the view mode

To create a new tabular view, set the view mode to **Simple tabular view**. If you want to share your view with other users you can:

- Set the profile (identified by **[profile-name]**): the view is shared with all users included in the profile.
- Set the user: the view is shared with only the specified user.

Click on the **Next** button to continue the creation process. You are presented with the options shown in the following image and described below.



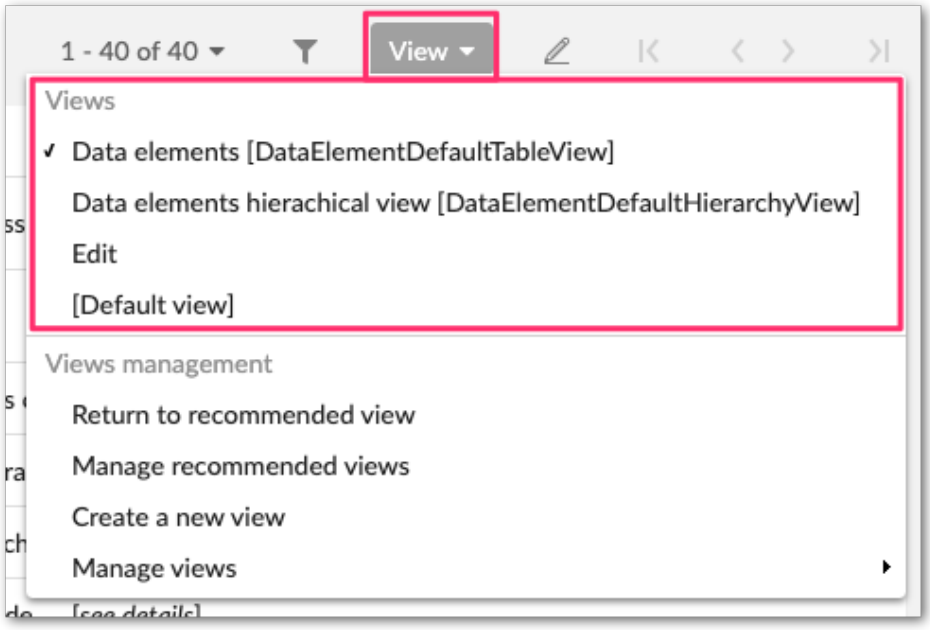
In the next view you can:

- Select columns you want to display.
- Set columns order.
- Add a filter.
- Set the pagination (number of records per page).
- Enable grid edit and set a toolbar. A toolbar corresponds to a list of available actions on the top of the table. You can set:
  - Light generic toolbar for the table view.
  - Generic toolbar for the table view.

Finally, save your new view or select **Save and apply** to save it and return to the table view with your new view applied.

Applying an existing view

To apply an existing view, open the **View** menu and select the view you want to see.



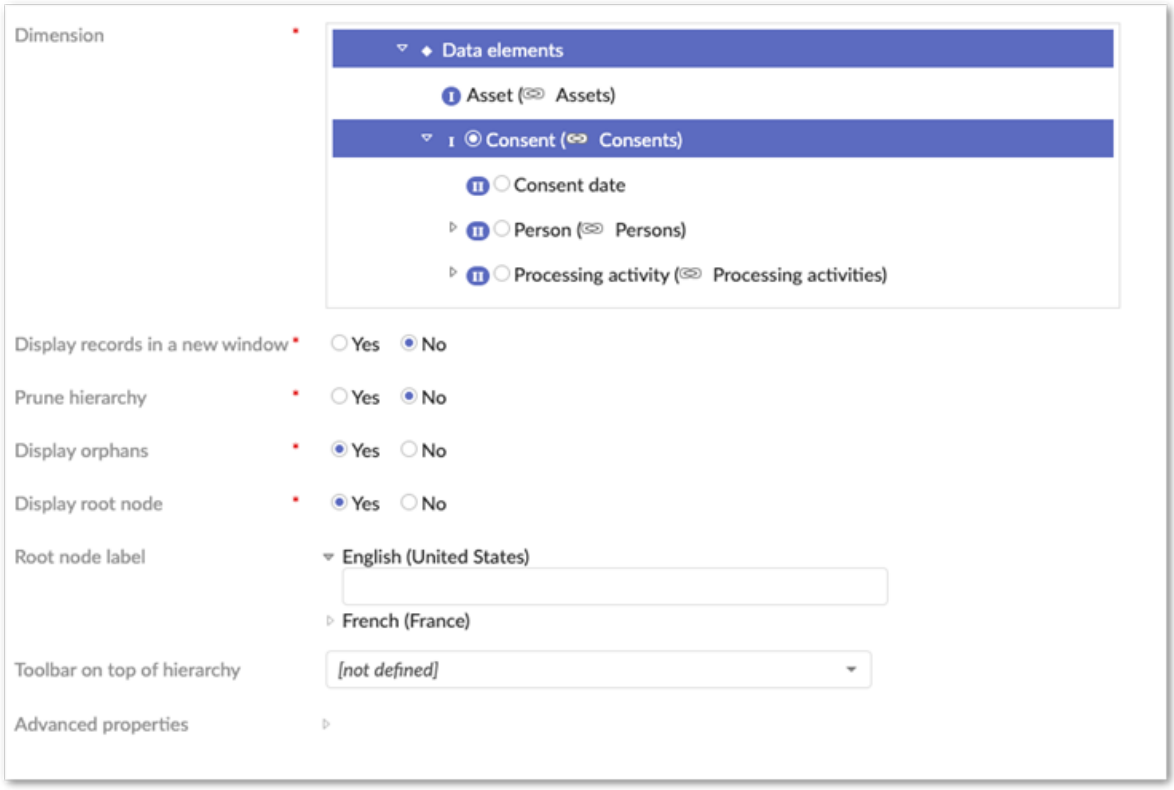
69.3 **Hierarchical views**

Creating new hierarchical view

To create a new hierarchical view, follow the steps above to create a tabular view. However, in the step to select the view mode, select **Hierarchical view**.

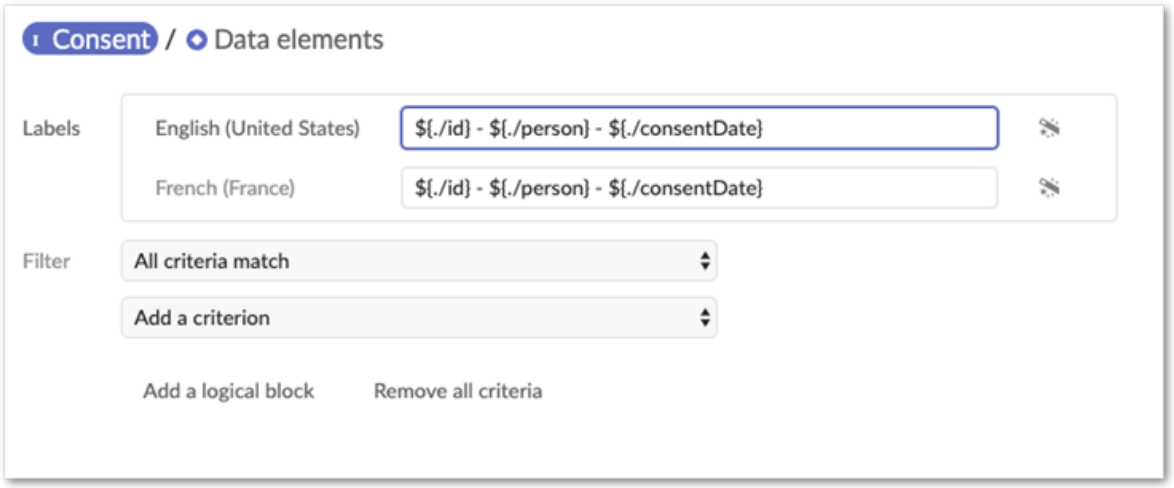


Select **Next** to define the hierarchy's dimensions and other options. The following image and list highlight some options:



- Display records in a new window: *If Yes, a new window will be opened with the record. Otherwise, it will be displayed in a new page of the same window.*
- Prune hierarchy: *If Yes, hierarchy nodes that have no children and do not belong to the target table will not be displayed.*
- Display orphans: *If Yes, hierarchy nodes without a parent will be displayed.*
- Display root node: *If No, the root node of the hierarchy will not be displayed in the view.*
- Root node label: *Localized label of the hierarchy root node.*
- Toolbar on top of hierarchy: *Allows you to set the toolbar on top of the hierarchy.*
- Display non-matching children: *In a recursive case, when a search filter is applied, allows the display of non-matching children of a matching node during a search.*
- Remove recursive root leaves: *In a recursive case, when a search filter is applied or if the mode is pruned, removes from the display the root leaves.*
- Detect cycle: *Allow cycle detection and display in a recursive case; the oldest node record will be chosen as the cycle root. Note that this does not work in search or pruned mode.*

Select **Next** to define how you want to display each node of your hierarchy.



In this view you can:

- Switch to node by clicking on the node name.
- Define label.
- Add a filter.

Finally, select **Create** to create your view or **Create and apply** to apply it after creation.

CHAPTER 70

Advanced data services overview

Advanced data services allow you to connect to the TIBCO® Data Virtualization (TDV) tool to harvest metadata from the data sources that already exist in your system or to create aggregated views in TDV using EBX® metadata assets. To use this functionality you must have a TIBCO® Metadata agent installed and configured on your system and registered.

If you have administrative access, see Installing the TIBCO® Metadata Agent [p 815] for more details.

CHAPTER 71

Metadata harvesting

This chapter contains the following topics:

1. Overview
2. Start harvesting service
3. Harvesting main view
4. Create a new configuration
5. Run configuration
6. Use an existing configuration
7. Schedule a harvesting
8. Create data source from instance
9. Specific Cases

71.1 Overview

Metadata harvesting is the process of gathering metadata from a data source that exists in your system and is supported by TDV and store it as an EBX® Metadata Management application asset.

You can run harvesting for data sources that are already configured in the TDV or for the new sources defined in the EBX® Metadata Management application.

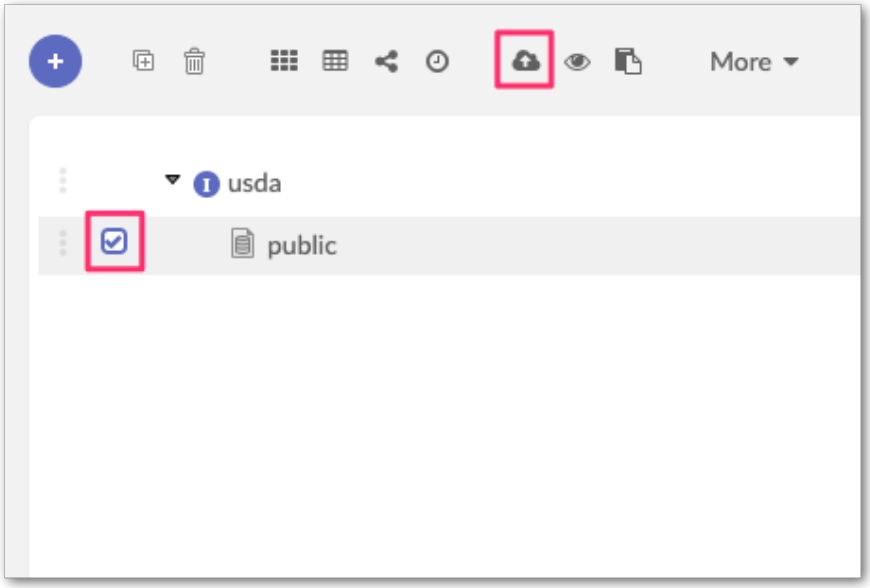
If you start the **Harvesting** service from an asset or a selection of assets, please, make sure that all assets were previously harvested by the Metadata agent.

If you start the **Harvesting** service from an instance or a selection of instances, you can create a data source in TDV for any that do not yet exist. See Create data source from instance [p 375] for more information.

The **Harvesting** service cannot be run on disabled assets or instances.

71.2 **Start harvesting service**

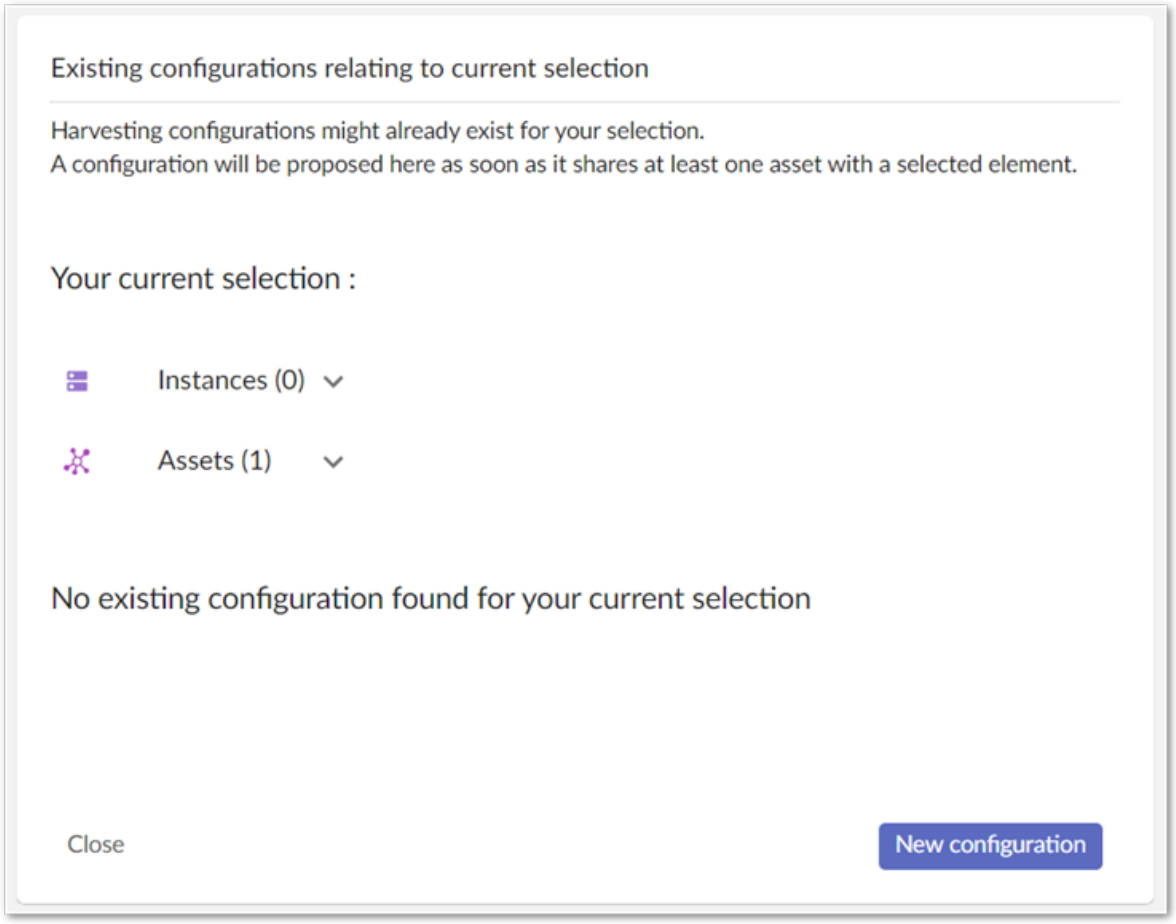
On the asset hierarchical view or instance view, select one or more assets or instances for which you want to harvest metadata. Select the cloud icon to begin harvesting. The harvesting main view displays.



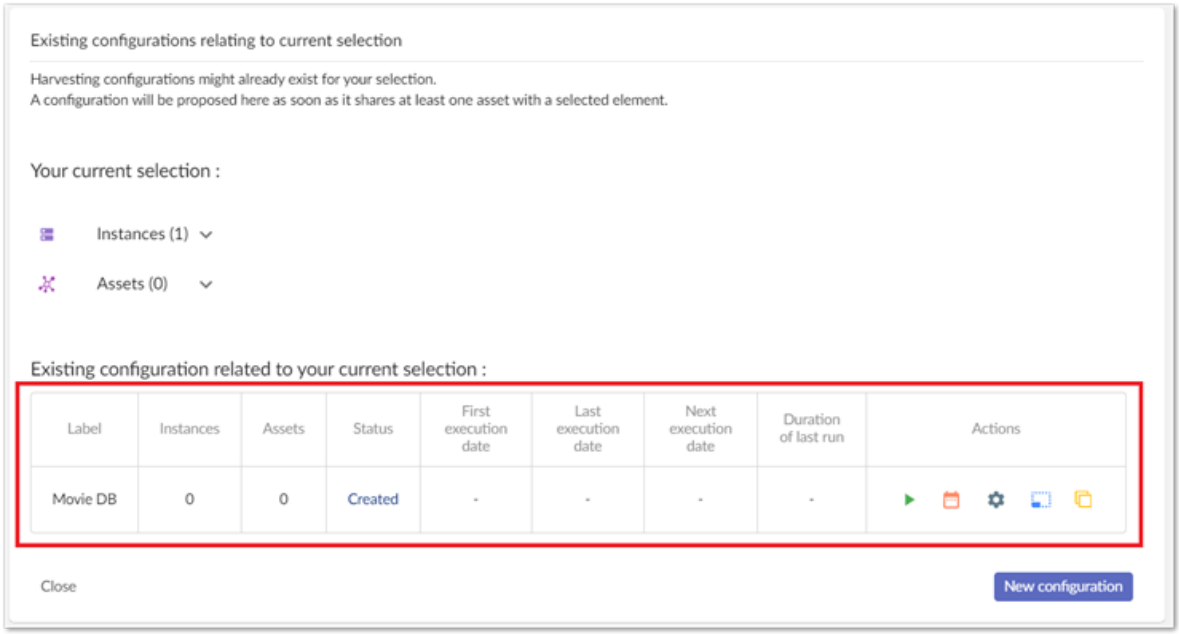
71.3 **Harvesting main view**

The harvesting main view displays your current selection and existing configurations.

An existing configuration is a harvesting that was already executed for one or several elements related to your current selection. If no existing configuration is available you will see a main view similar to the one in the screenshot below:



If you have an existing configuration, you will see something like the screenshot below:



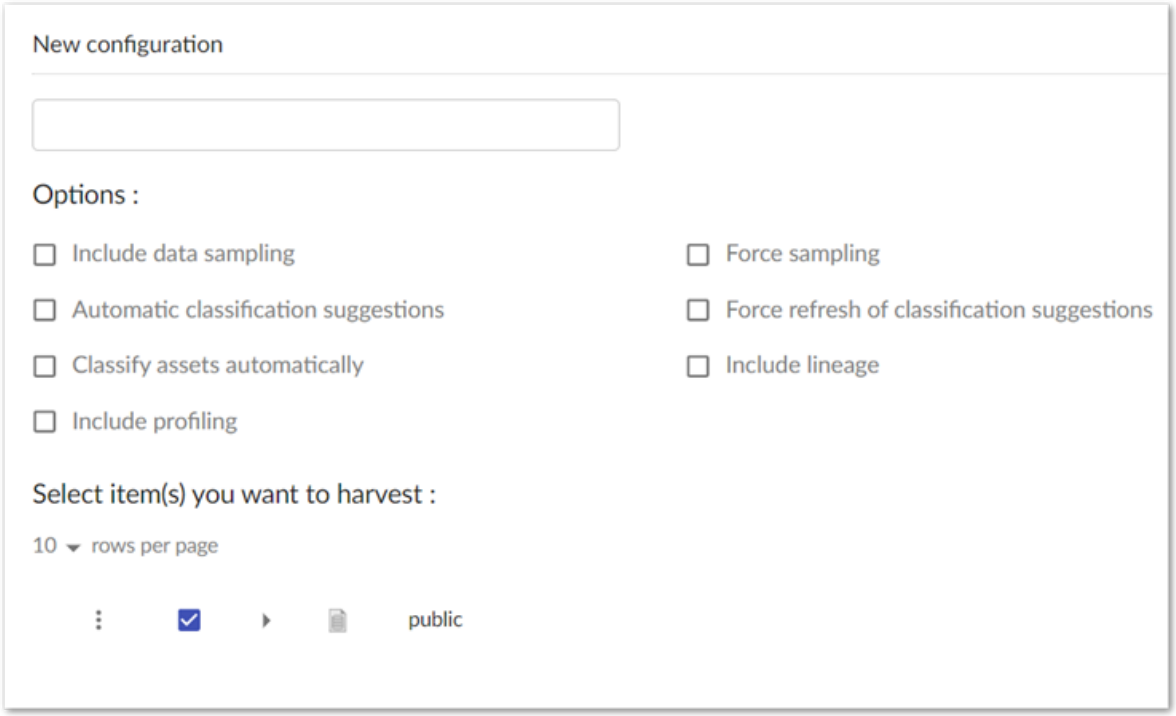
From an existing configuration, you can perform the following actions:

- Run
- Schedule
- Configure
- Duplicate
- Extend with current selection

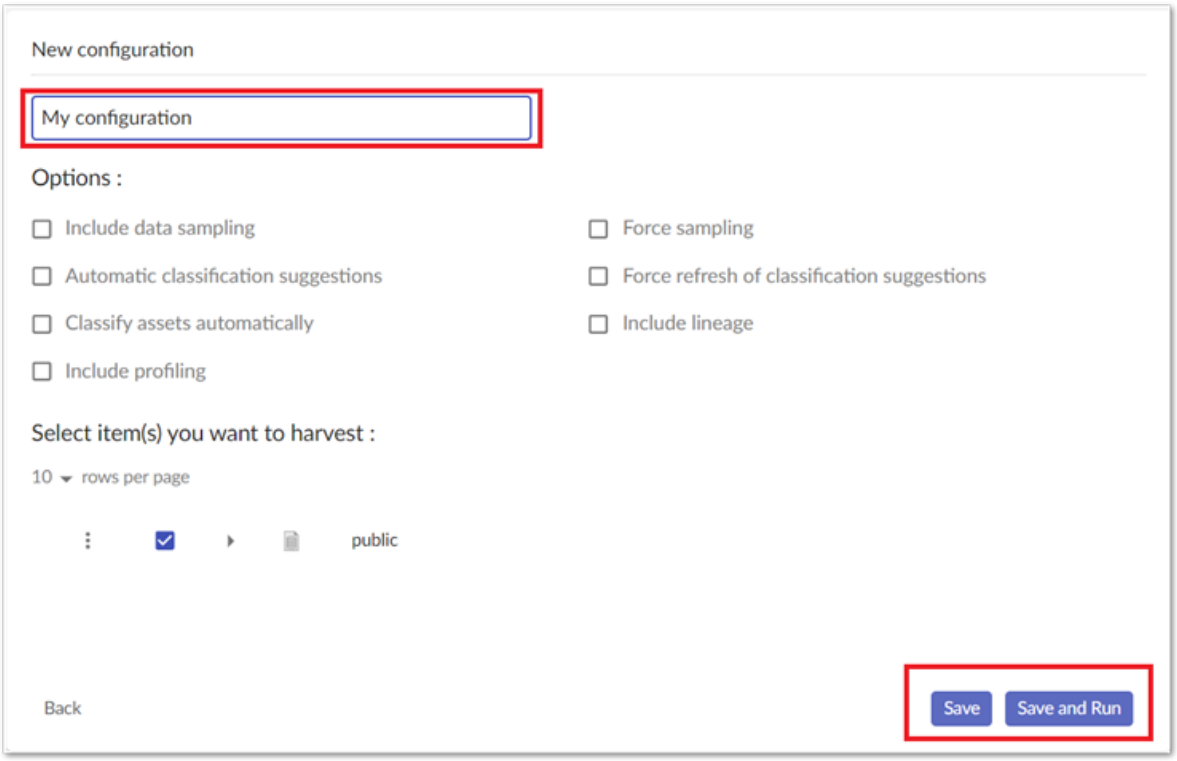
See Use an existing configuration [p 372] for more information.

71.4 Create a new configuration

From the main view, click on the bottom right **New Configuration** button. The new configuration screen displays.



In this screen, start by adding a configuration name. This is necessary for displaying more actions.



On the top of this screen, you can see options:

- Include data sampling: for the assets eligible for sampling harvesting will retrieve data samples from the source system, if available. See Asset sampling [p 379] for more details regarding supported types.
- Force sampling (activates **Include data sampling** automatically): force data sampling on your assets (overwrite existing samples).
- Automatic classification suggestion: the harvesting will also start the classification of your assets.
- Force refresh of classification suggestion (activates *Automatic classification suggestion*): if your asset already has a classification suggestion, this option will force the harvesting to update it.
- Classify assets automatically (activate *Automatic classification suggestion*): Apply the classification suggestion to your asset instead of giving you a list of potential matches.
- Include lineage: if your harvesting contains assets of type TDV Published View, the lineage will be extracted and stored for them. As lineage can be quite complicated and concern a large number of resources, it is recommended to activate this option only for scheduled runs, not for immediate harvesting executions.
- Include profiling: for eligible assets (DB tables, CSV files, Excel worksheets) cardinality statistics and other profiling information will be included in the harvesting results. As queries for statistics can be time consuming on large data sources, it is recommended to activate this option only for scheduled runs, not for immediate harvesting executions.

In the hierarchical view, you can perform multiple actions:



1)	This action allows you to unselect all children and keep the parent node selected. This is only available if the node has children.
2)	Select or unselect the node for harvesting. If the node has children, this action will also select/unselect all children.
3)	Expand more or expand fewer children. Clicking on this icon will display children if they exist, if not you will see *No data* message. If you have connection issues with TDV, you may see an error here.



This is not an action but the name of the instance/asset with the logo. The subtitle contains the following storage information: - New: this is a new instance/asset - Stored: instance/asset is already in the EBX® Metadata Management application.

Once you give your configuration a name and if at least one element is selected for harvesting, you will see **Save** and **Save and Run** buttons on the bottom:

- Save: Only saves your configuration. After saving it, you can schedule it if needed. See Schedule a harvesting [p 373] for more information.
- Save and Run: Save your configuration and run the harvesting. This action will display the *Run* screen of the harvesting

Data profiling

Data profiling is available for following asset types:

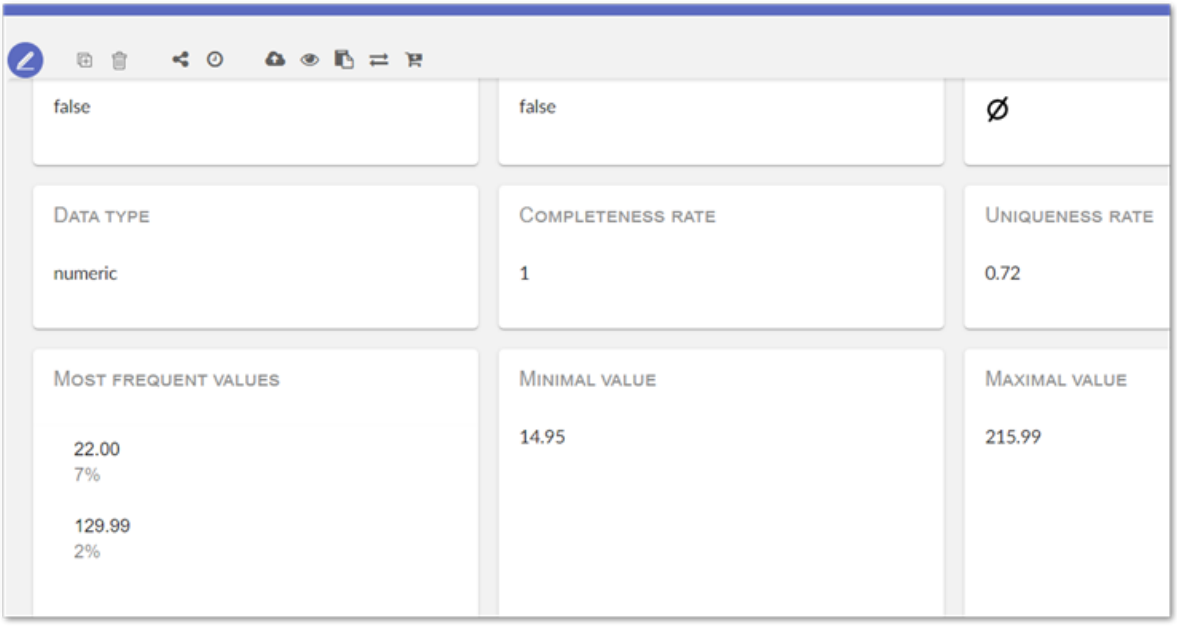
- DB table and its columns
- CSV file and its columns
- Excel Worksheet and its columns

If one of such assets is present in the scope of your harvesting (as a root asset or as a descendant), you can activate data profiling for it.

Please, note that data profiling can take time to be gathered from the physical data source, so we recommend to only activate this option for scheduled configurations and do not use it with direct running.

Following information can be retrieved from the data source:

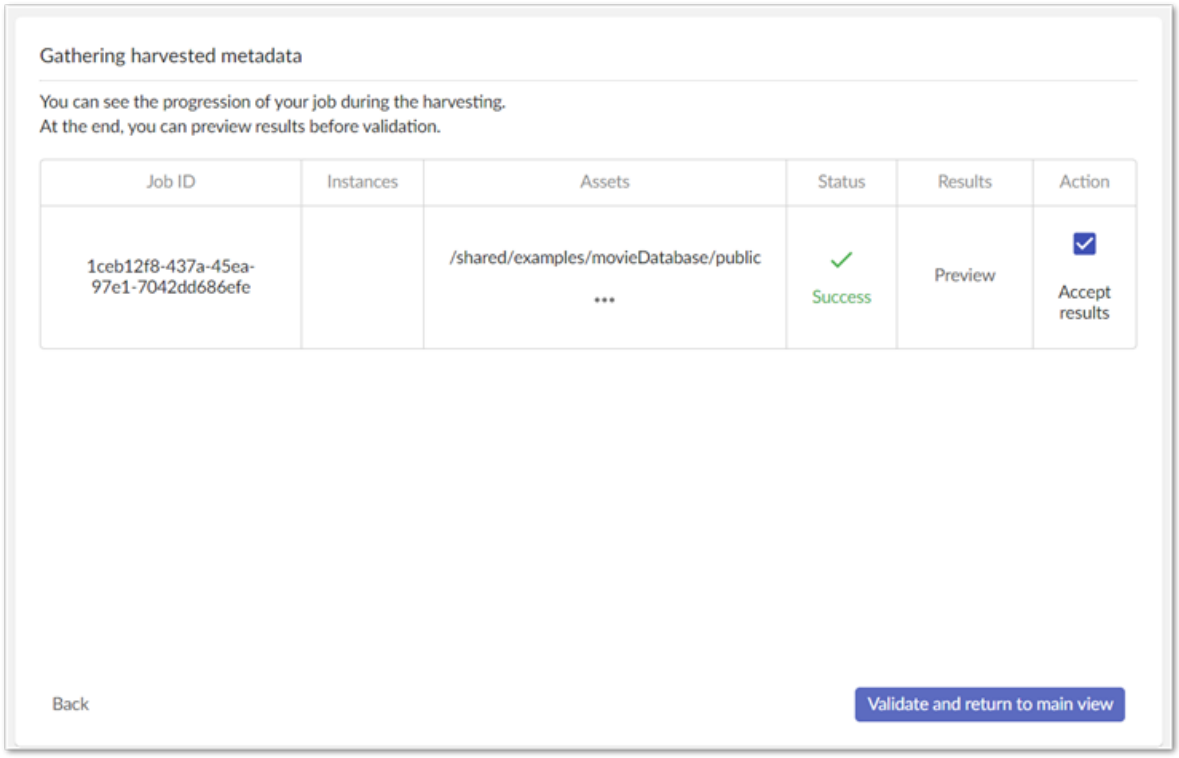
- Completeness rate: shows the part of non-null values in total values.
- Uniqueness rate: shows the part of unique values in total values.
- Most frequent values: lists top 5 values that are most common in this column.
- Minimal value: shows the minimum value in this column.
- Maximal value: shows the maximum value in this column.



71.5 Run configuration

If you select **Save and Run** from the configuration view, you will start the harvesting. First the application needs to communicate with TDV, that's why you will see the progress view.

After that, the progress table displays.



This table is auto-updated to display the status of the harvesting. You have multiple available actions:

- In the **Instances** or **Assets** column, you can select the … to display the full list of instances/assets. The list displays in a pop-up. Click outside the popup to close it.
- In the **Results** column, once the status is **Success**, you can select **Preview** to display harvesting results. If the status is **Failed** or **In progress**, this button is not available.
- If results are correct for you, you can check the **Accept results** checkbox to approve them. This checkbox is checked by default.
- Select **Validate and return to main view** to:
  - Accepts results and add them in the asset table if **Accept results** is checked. You will be redirect to the main view.
  - Reject results if **Accept results** is not checked. You will be redirected to the main view.
- Click on the **Back** button to go back to the main view. From the main view, you will be able to return to this progress view if your harvesting is not validate yet.

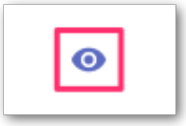
Once you validate results, you will see on the main view that your configuration status is updated.

71.6 Use an existing configuration

From the main harvesting view, if you have existing configurations, the following actions are available:



1)	Run the configuration. You will see the progress view, then the progress table.
2)	Schedule the configuration.
3)	Configure the current configuration. You will see the configuration screen.
4)	Extend the configuration with current selection. The configuration screen displays and the harvesting scope is merged with your current selection.
5)	Duplicate the current configuration. A copy of this configuration displays on the configurations list.



See the progress table view. This option is only available if the configuration status is **Pending for approval**.

Selecting **Configure** displays the configuration screen. This screen is similar to the one evoked in the Create a new configuration [p 368] section but the instance/asset hierarchy, configuration name, configuration options are pre-populated. You will be able to change them and save it.

71.7 Schedule a harvesting

You can schedule one or several executions of an existing harvesting configuration. There are two ways to do so:

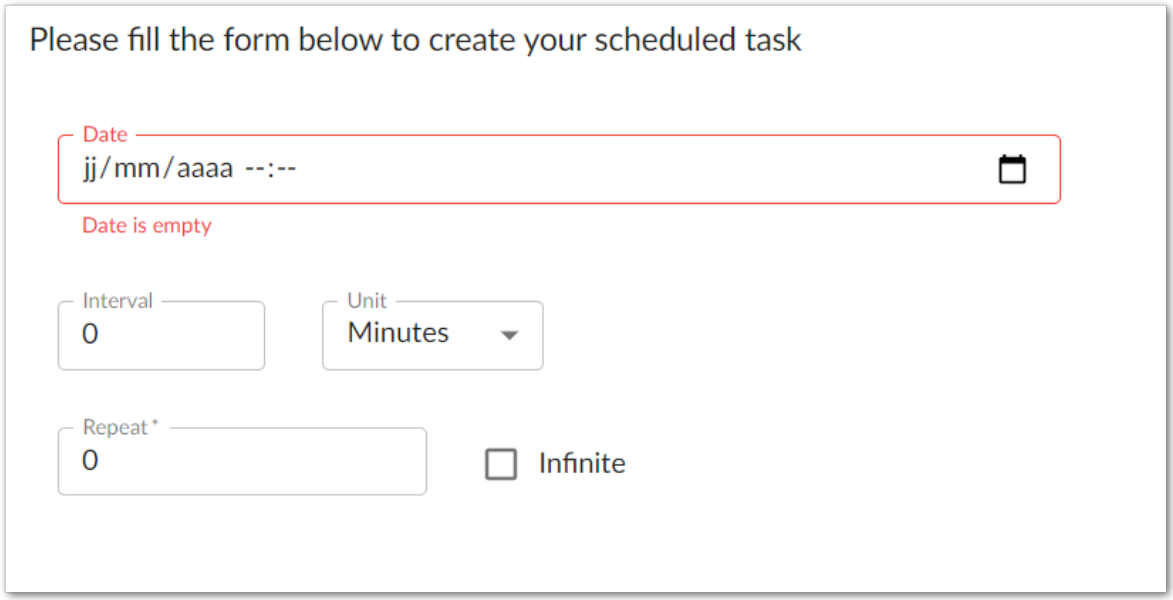
- From the configuration view, after saving the configuration, by selecting **Schedule** on the bottom.



- From the main view by selecting **Calendar**.



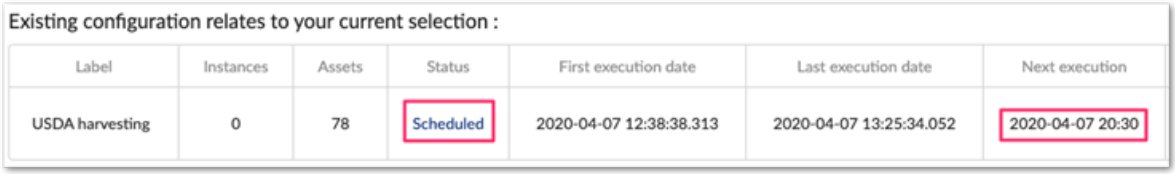
Clicking on the button displays a dialog box where you can configure the scheduling.



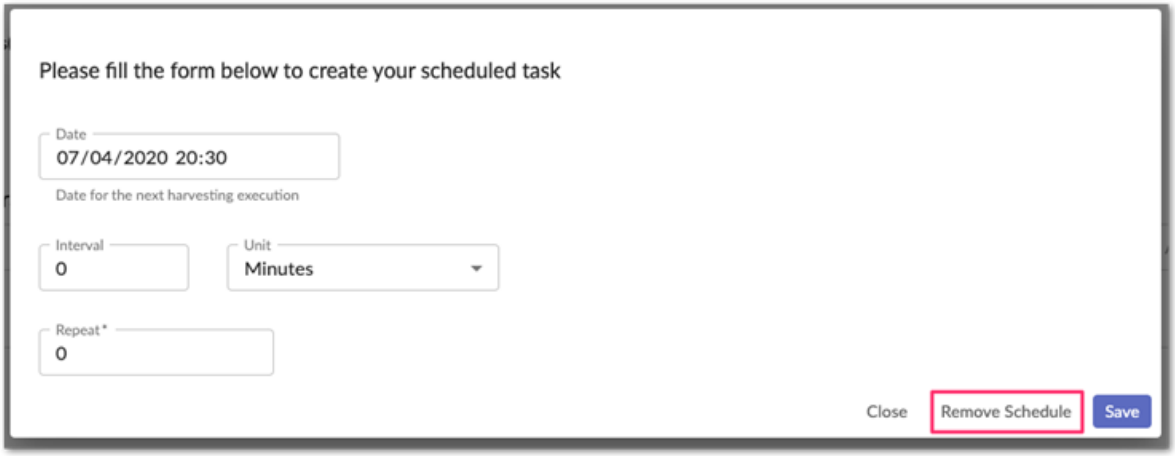
To save the scheduling you have to insert the date. Date This field is mandatory Insert the date and time here. You can use the arrow on the right to display the calendar. Interval between multiple scheduling if repeat is set. Unit of the interval. By default, it is in minutes but you can change it to hours or days. Repeat Number of repeats for the scheduling. To repeat this harvesting to infinite, check the **Infinite** checkbox.

Example: You want to schedule the harvesting every month. The date is 01/01/current year, interval 28 days and repeat is 12.

If you want to cancel harvesting, you have to be sure that the configuration is already scheduled. You can see it from the main view, the configuration will have the status **Scheduled** and the next execution date.



By selecting the calendar icon near the configuration, the schedule dialog box will display a **Remove schedule** button. Click on it to cancel scheduling.



71.8 Create data source from instance

If you wish to harvest metadata for a data source that is not yet configured in TDV, you can configure it directly from the EBX® Metadata Management application.

Prerequisites

To do so, first you need to check that all necessary information is provided in the instance's attributes and that the instance type is supported for creation by the EBX® Metadata Management application.

Currently, you can create:

- relational data sources;
- file-based data sources, such as Excel, XML, WSDL or CSV;
- SOAP data sources;
- REST data sources (for these data sources, manual creation of the operations is required in TDV after creating the data source).

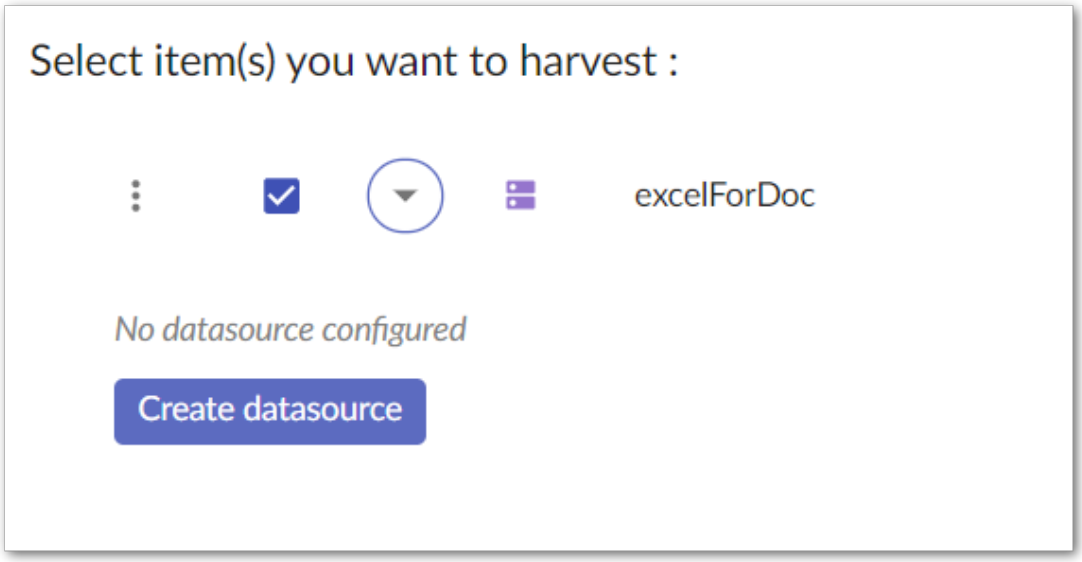
A complete list of supported adapters is available in the **Administration - Datasource Types** table with attribute **Creatable?** = **Yes**.

An instance that you want to use to create a data source in TDV must provide necessary properties to connect to the physical data source from TDV. Below is the list of mandatory attributes depending on the type of the data source.

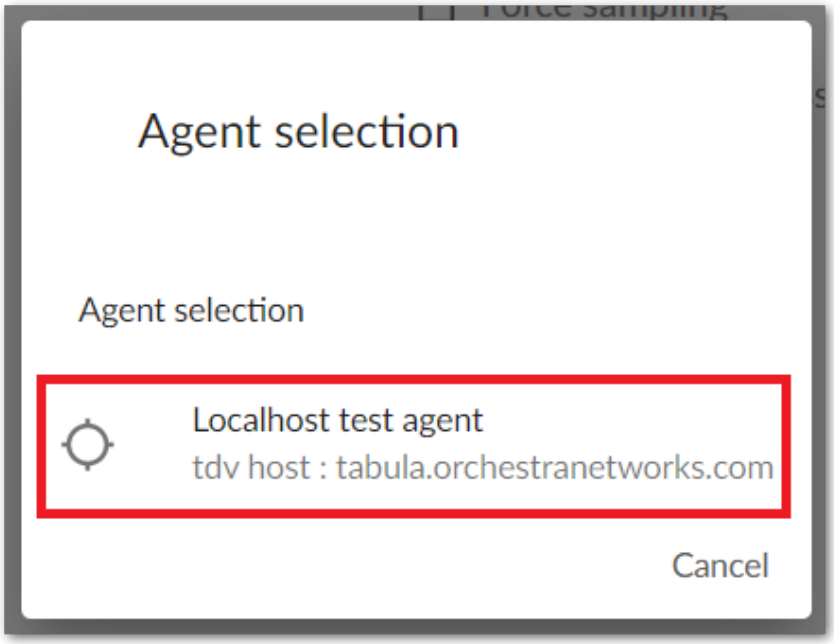
Create and harvest

Start creation of the new harvesting configuration for your instance as described in Create a new configuration [p 368] section.

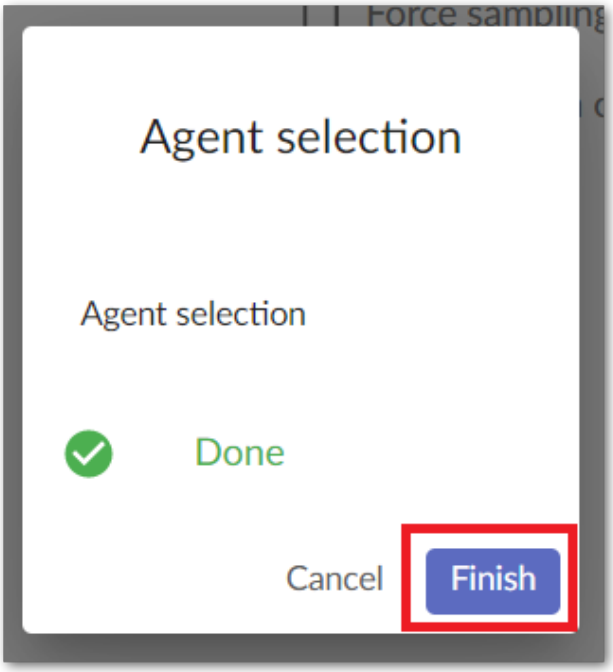
Click to expand children of the instance and the **Create datasource** button displays.



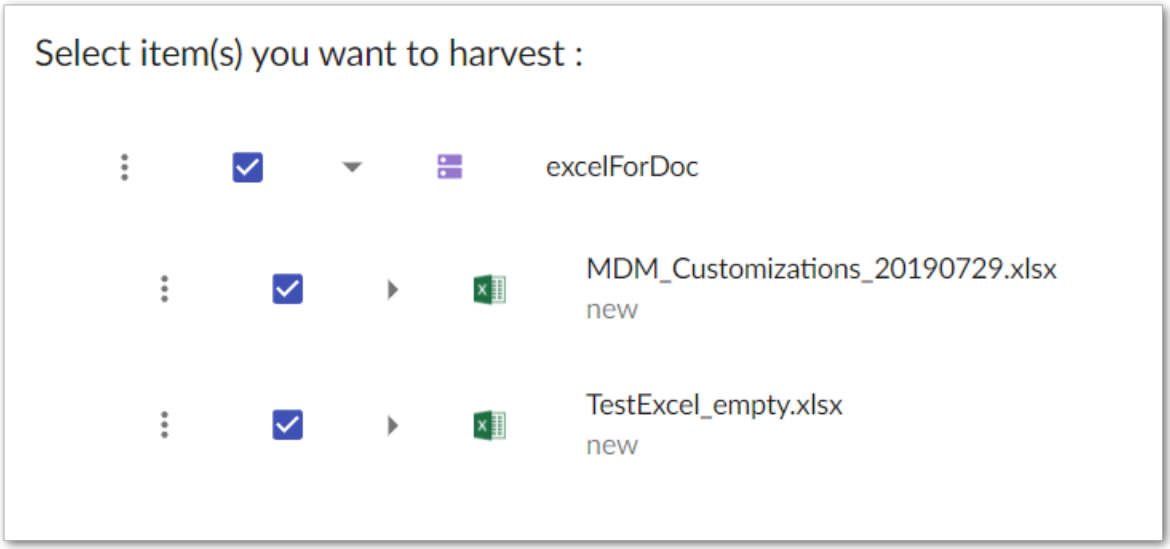
Click on it to show the list of available Metadata Agents. Click on the agent that you wish to use for data source creation to launch the process.



Wait for the process to complete. In case the creation was successful you will see a message *Done* in green. Finalize the process by selecting **Finish**.



This should reload the harvesting screen and allow you to see the direct children of the instance.



You can now continue with harvesting configuration as described in previous sections.

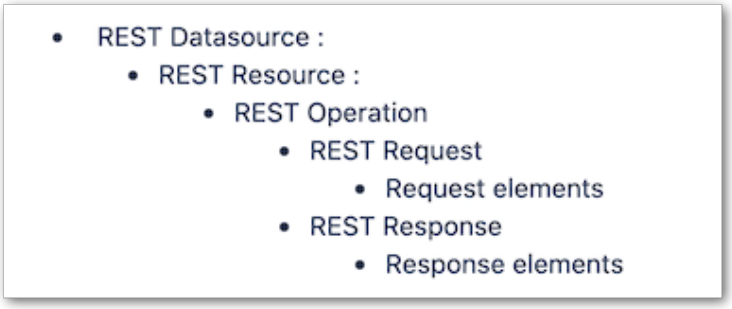
71.9 Specific Cases

Harvesting REST Resources

The EBX® Metadata Management application supports harvesting of REST resources documented manually in TDV. This comes with a specific behavior due to the way resources are documented in

TDV. Indeed, the EBX® Metadata Management application represents REST resources according to the RESTful architecture which is not the case in TDV.

Here is the way REST Resources are displayed:



Harvesting configurations on Assets under *Operation* in the hierarchy are not possible since these resources are created by the agent and do not exist in TDV.

During the configuration of a harvesting from a REST Datasource, you will see all the operations registered in TDV, but no REST Resource asset as they are created by the EBX® Metadata Management application.

CHAPTER 72

Asset sampling

This chapter contains the following topics:

1. Overview
2. Asset type available for sampling
3. How to request a sample
4. Wrong asset selection for sampling
5. XML Element sampling

72.1 Overview

Asset sampling allows you to request a sample data in a file to see the content of the asset. By default, the sample contains the 100 top rows from the target source.

From the asset view, you can request a sample file if the asset is available for sampling.

72.2 Asset type available for sampling

You can request a sample on the following asset types:

- DB Table (database table)
- EXCEL worksheet
- CSV file
- TDV published view
- XML Element (which is marked as object root)

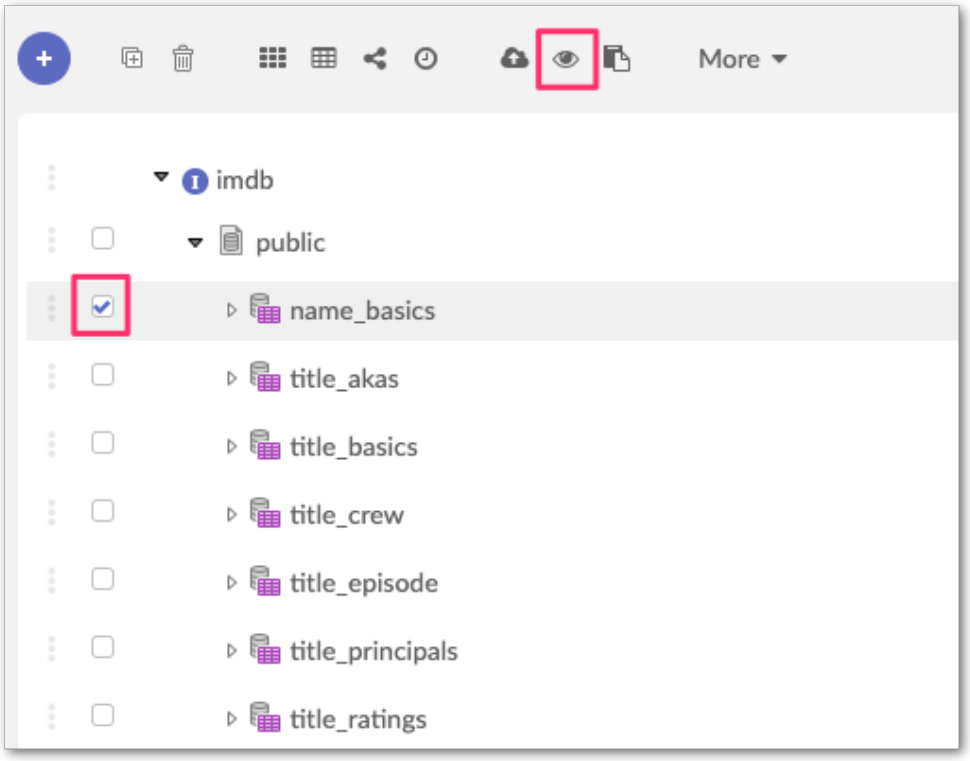
Sampling cannot be run on disabled assets.

Sampling can only be run for the assets that were created using the Metadata harvesting [p 365] service.

72.3 How to request a sample

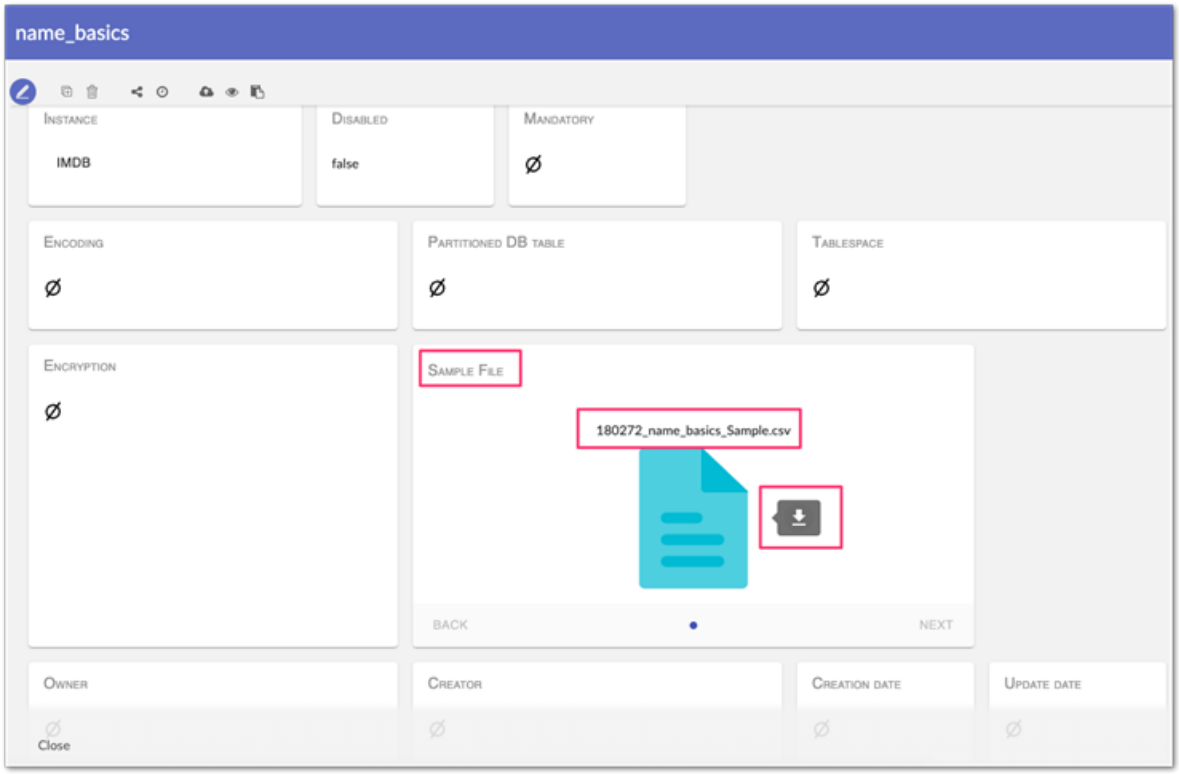
Start by displaying the asset's view by selecting **Assets** in the perspective menu.

On the hierarchical view, select one or more assets for which you want a sample file. Once it's done, select the eye icon on the top to request sample.



You will see a progress message and a success message when finished.

You can close the service and go back to your asset to download the sample file. On the asset view, locate the **Sample file** field, hover your mouse cursor on the file. A tooltip displays a download button.



You can finally open the file.

180272_name_basics_Sample

birthYear	deathYear	knownForTitles	nconst	ns_soundex	primaryName	primaryProfession	s_soundex	sn_soundex
1899	1987	0050419,0072308,0053137,0043044	1	F6323	Fred Astaire	soundtrack,actor,miscellaneous	A236	A2361
1924	2014	0038355,0117057,0071877,0037382	2	L6512	Lauren Bacall	actress,soundtrack	B24	B2465
1934	null	0057345,0054452,0059956,0049189	3	B6231	Brigitte Bardot	actress,soundtrack,producer	B63	B6316
1949	1982	0077975,0080455,0072562,0078723	4	J5142	John Belushi	actor,writer,soundtrack	B42	B425
1918	2007	0050986,0050976,0069467,0083922	5	I5256	Ingmar Bergman	writer,director,actor	B625	B6252
1915	1982	0038787,0077711,0036855,0038109	6	I5263	Ingrid Bergman	actress,soundtrack,producer	B625	B6252
1899	1957	0043265,0033870,0034583,0037382	7	H5161	Humphrey Bogart	actor,soundtrack,producer	B263	B2635
1924	2004	0068646,0044081,0078788,0070849	8	M6451	Marlon Brando	actor,soundtrack,director	B653	B6535
1925	1984	0059749,0061184,0087803,0057877	9	R2631	Richard Burton	actor,producer,soundtrack	B635	B6356
1899	1986	0031867,0029870,0042041,0035575	10	J525	James Cagney	actor,soundtrack,director	C25	C2525
1901	1961	0027996,0044706,0035896,0034167	11	G6216	Gary Cooper	actor,soundtrack,producer	C16	C1626
1908	1989	0042192,0035140,0056687,0031210	12	B312	Bette Davis	actress,soundtrack,make_up_department	D12	D1213
1922	2019	0053172,0048317,0060463,0055100	13	D623	Doris Day	soundtrack,actress,producer	D	D362
1916	null	0031381,0041452,0029843,0040806	14	O4131	Olivia de Havilland	actress,soundtrack	null	D1453
1931	1955	0048028,0043228,0049261,0048545	15	J5235	James Dean	actor,miscellaneous	D5	D5252
1925	1992	0069946,0091763,0057345,0096320	16	G6234	Georges Delerue	composer,soundtrack,music_department	D46	D4626
1901	1992	0052311,0051201,0021156,0055031	17	M6453	Marlene Dietrich	soundtrack,actress,music_department	D362	D3625
1916	null	0043338,0052365,0080736,0054331	18	K6232	Kirk Douglas	actor,producer,soundtrack	D242	D2426
1920	1993	0056801,0050783,0053779,0071129	19	F3621	Federico Fellini	writer,director,assistant_director	F45	F4513
1905	1982	0050083,0082846,0064116,0032551	20	H5615	Henry Fonda	actor,producer,soundtrack	F53	F5356
1917	2013	0032976,0039504,0040536,0034248	21	J5153	Joan Fontaine	actress,soundtrack,producer	F535	F5352
1901	1960	0031381,0026752,0025316,0052278	22	C4621	Clark Gable	actor,soundtrack,producer	G14	G1424
1922	1969	0037059,0055031,0032138,0047522	23	J3264	Judy Garland	soundtrack,actress	null	G6453
1904	2000	0082031,0117631,0045943,0071877	24	J5242	John Gielgud	actor,writer,director	G423	G4232

72.4 Wrong asset selection for sampling

If you have selected an asset which is not available for sampling, you will get an error message.

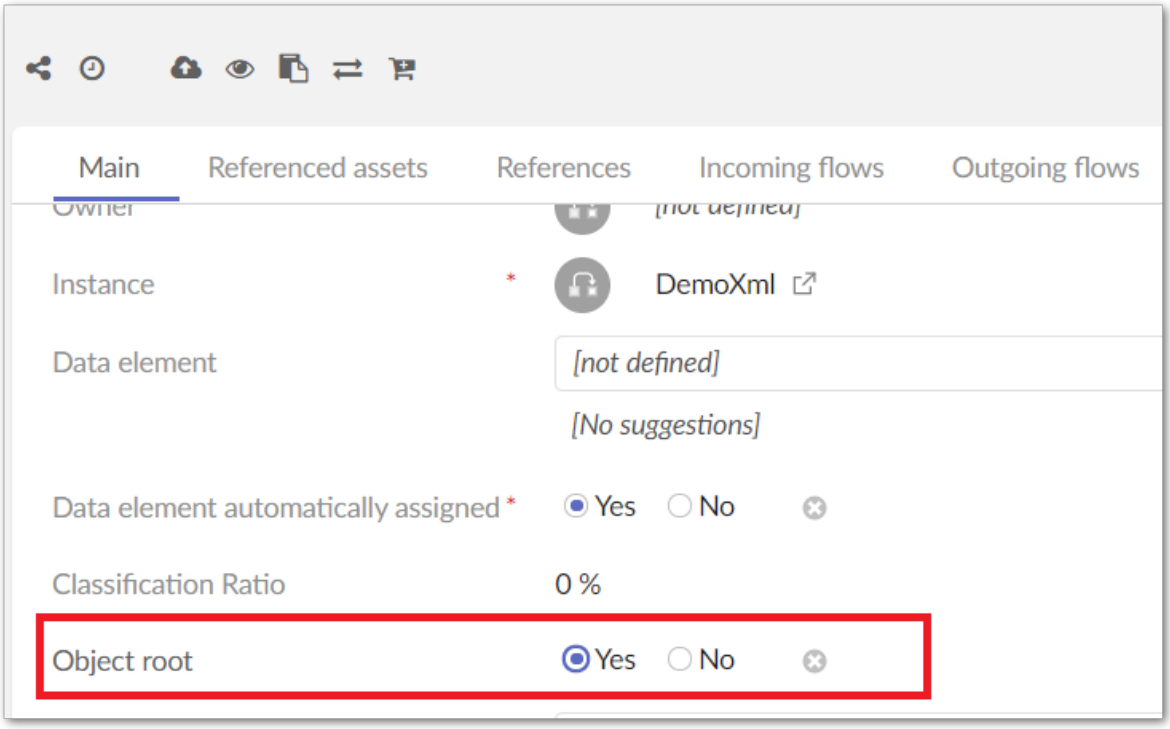


72.5 XML Element sampling

For XML structures it is possible to obtain a sample in XML format based on an asset of type XML ELEMENT.

As XML document structure is open and can be analyzed under different angles, you can define for which object in your XML you wish to sample.

To accomplish this, open the corresponding asset for editing, scroll down to the **Object root** property and set it to **Yes**:



Save the asset to apply changes.

Now you can request sampling for this asset in the harvesting process [p 365].

CHAPTER 73

Asset classification

This chapter contains the following topics:

1. Overview
2. Manual classification
3. Using classification console

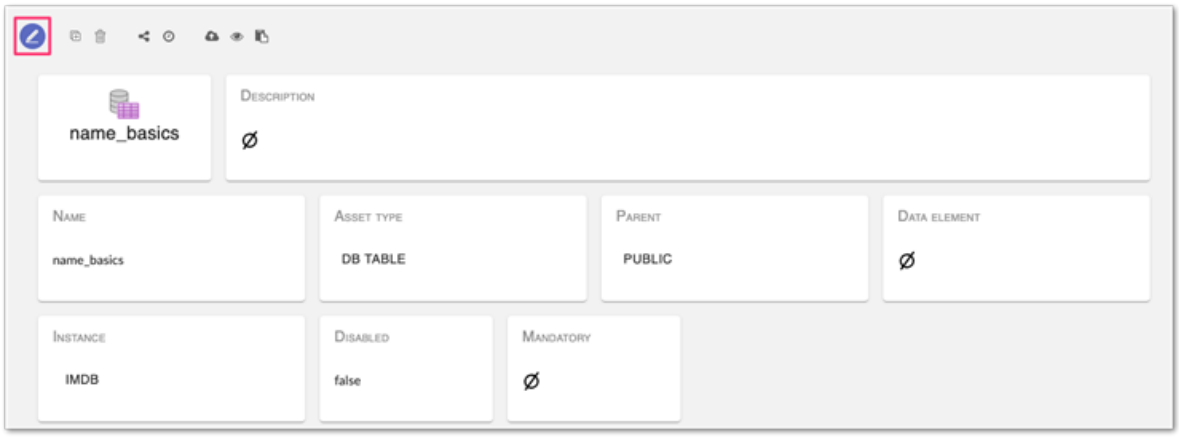
73.1 Overview

You can match assets with data elements. It's the main feature that explains how a business object is linked to a technical object.

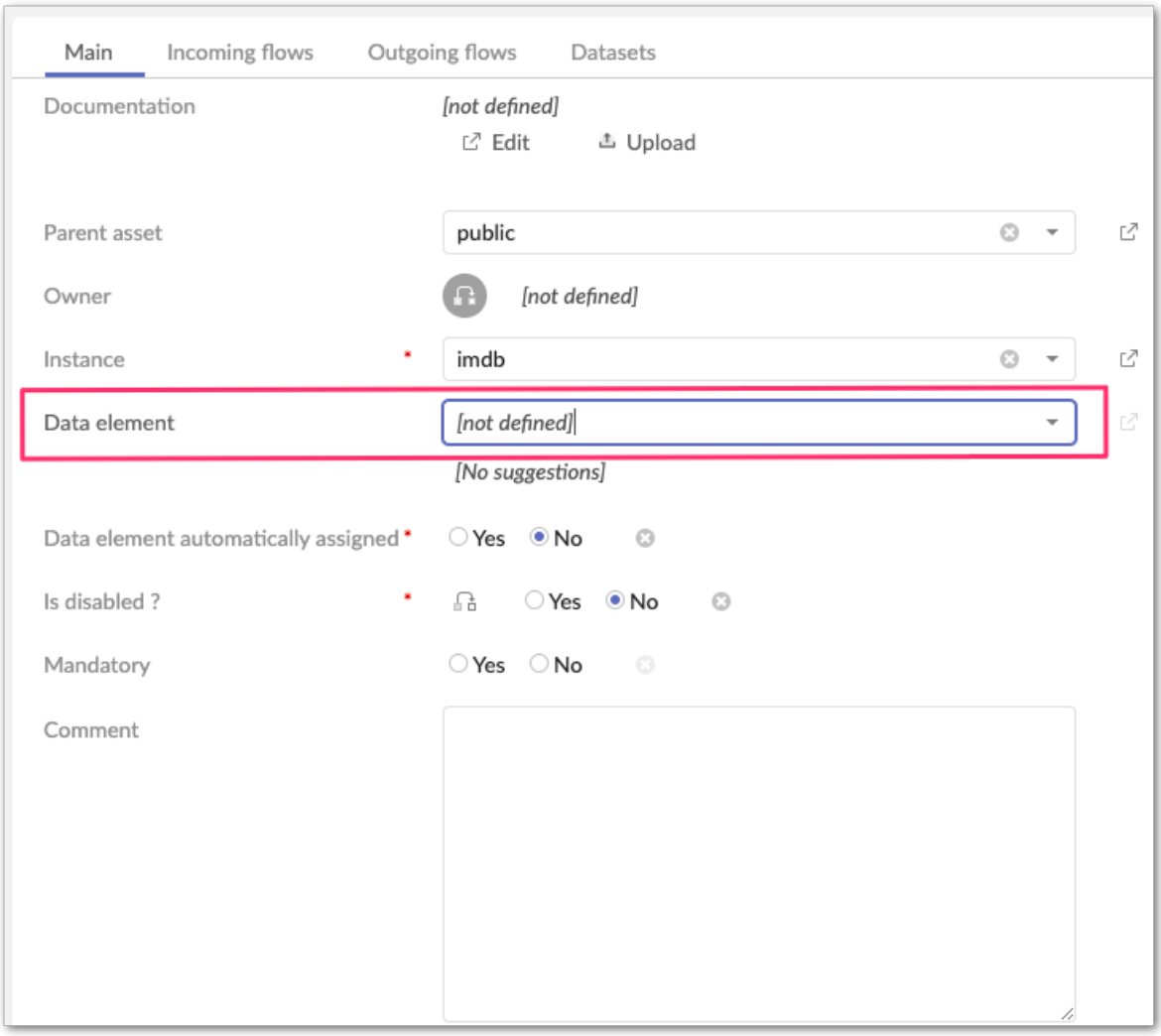
It's possible to define this link on your own but it could take time on many assets. The classification feature helps you to find the match between an asset (technical) and a data element (business).

73.2 Manual classification

On the asset view, click on the edit icon on the top left corner.



Then select the data element in data element list.



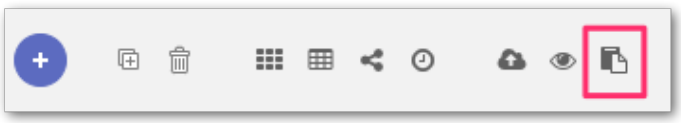
You can search by entering the data element name. Do not forget to save your update.

73.3 Using classification console

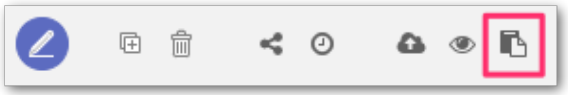
Instead of searching and selecting the data element by yourself, you can use the **Classification Console** to give you some suggestions on the right data element to link to the current asset.

This option is using a predefined algorithm and needs data to get more efficiency.

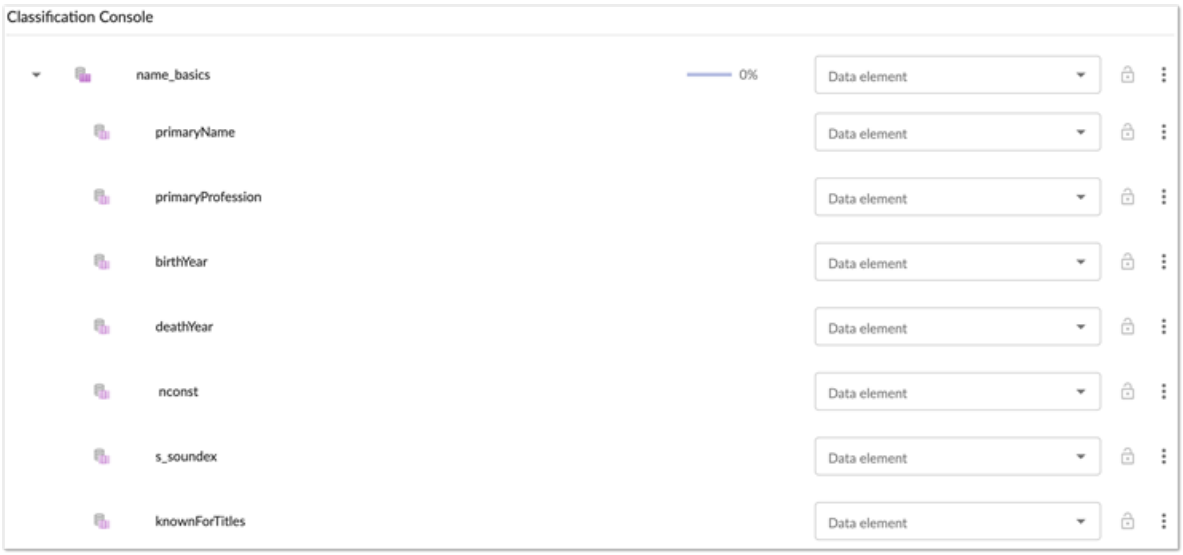
- From the hierarchical view, select one or more assets that you want to classify then select the **Classify** icon:



- From a single asset view, select the classify icon:



A new screen displays and shows the current selected asset and its children if they exist.

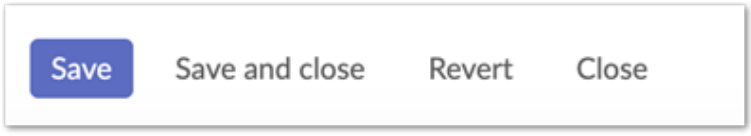


On this screen you can:

1. Expand or collapse the hierarchy (only available for asset with children).
2. Click on the asset name to see details.
3. See asset classification ratio (if asset and its children are classified).
4. Search and select the data element to link with the asset.
5. Lock or unlock the data element for automatic classification.
6. Access extra options like running automatic classification.

On the bottom bar you can:

- Save your current modification.
- Save and close the classification console.
- Revert the current modification.
- Close the classification console (which will display a confirmation pop-up if you have current unsaved modifications).



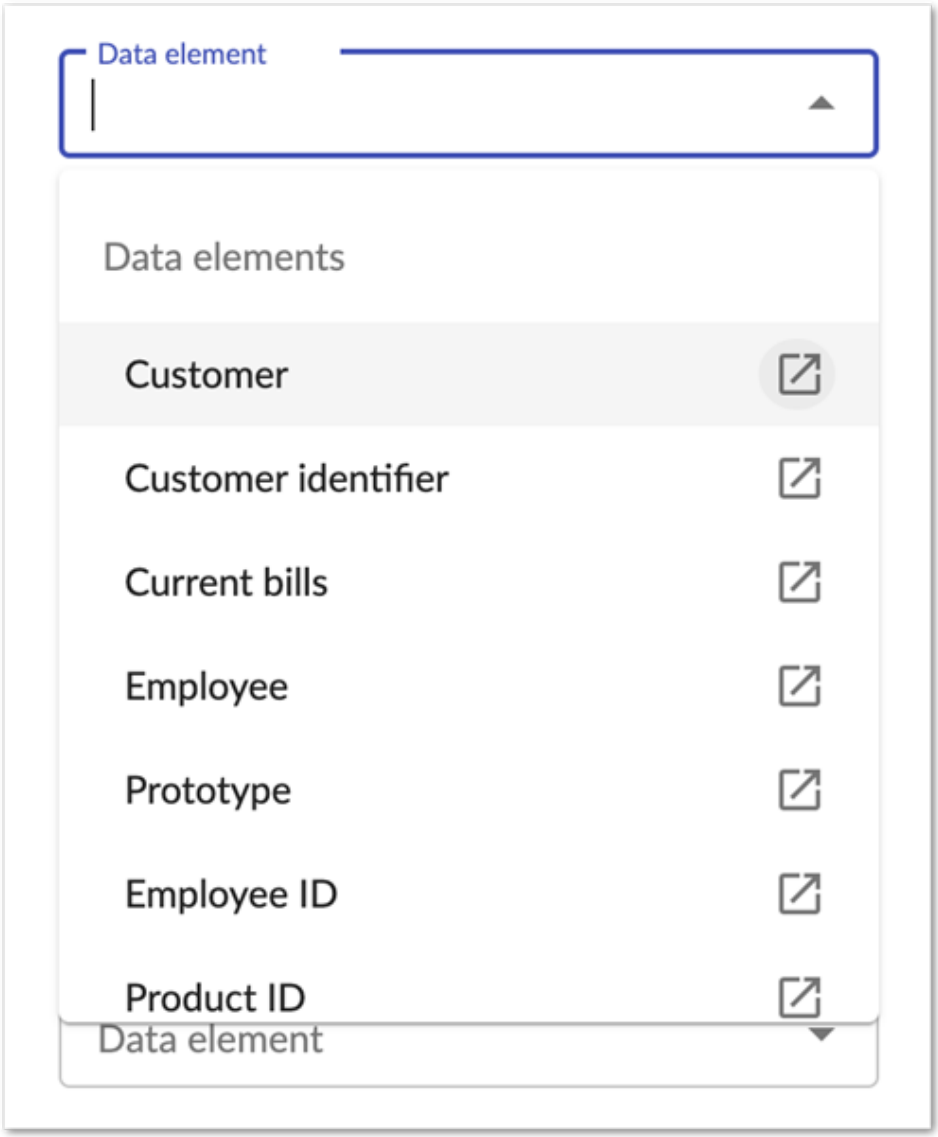
Search and select a data Element

Select the data element box to view:

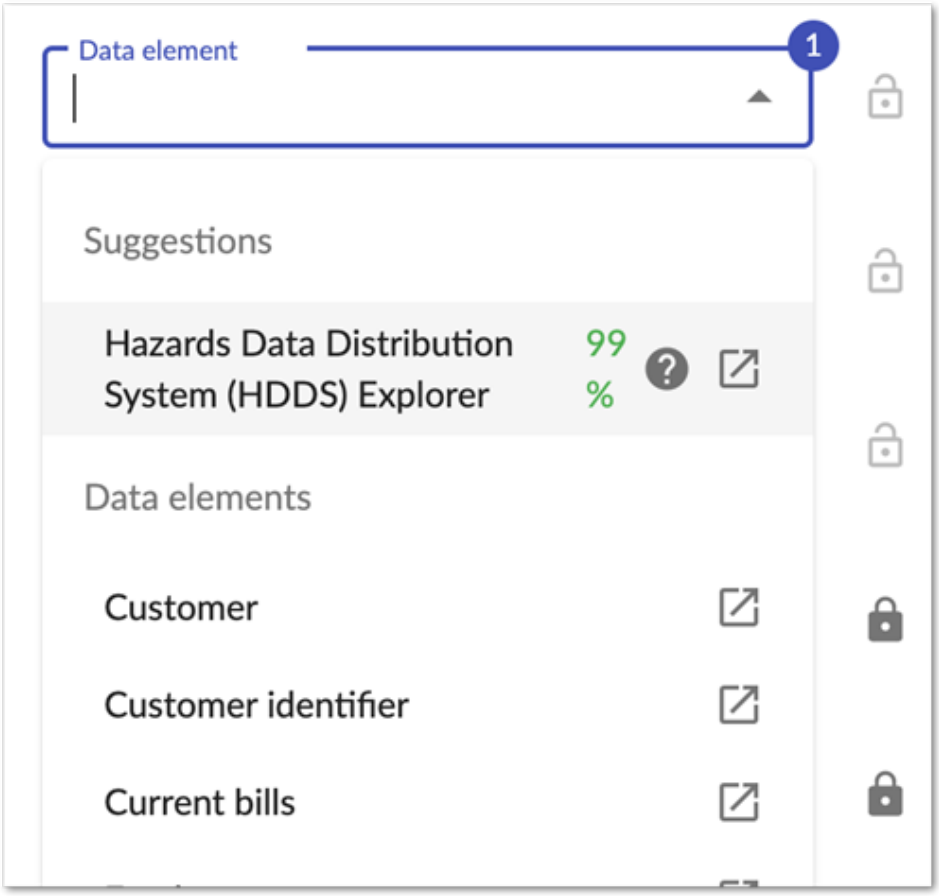
- Data element suggestion list if one exists.

- Data element list.

You can search a data element by entering text. You can see data element details by clicking on the preview icon.



If a data element select box contains suggestions you will see a notification near it. By clicking on the box, you will see the suggestions and all data element.



A suggestion displays the matched result in percentage and information about the classification configuration used.

If you have done modifications, you will see a blue exclamation point on the left side of the data element select box.

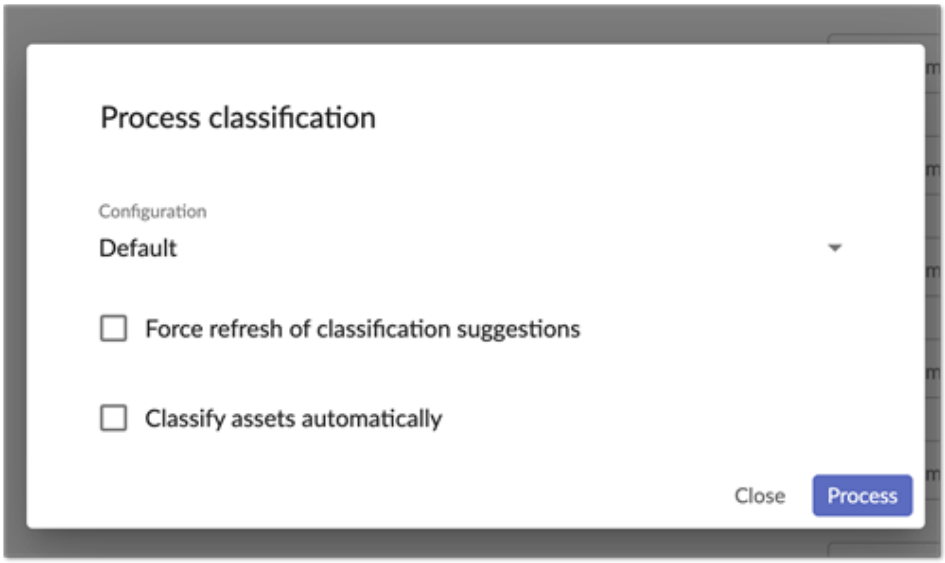
By hovering your mouse over it, you can see current modification details (data element change, asset locked or unlocked).

Run automatic classification

To access extra options, click on the vertical menu button on the right side of lock option. Select **Run classification**. A pop-up displays and allows you to:

- Select the classification configuration.
- Force a refresh of classification suggestions.

- Classify assets automatically (this will select the best matching data element for you. If you have unsaved changes, a warning message will inform you that you may lose modifications if you don’t save them before processing).



Click on *process* icon to start classification. While the classification is running, you cannot run a new one on the current asset and its children.

Do not forget to save modifications after leaving the classification console.

Lock an asset

If there is an existing data element linked to an asset, you can lock it by clicking on the lock button on the right side of the data element select box. A locked asset is ignored if you run the classification process with the **Classify assets automatically** option. An unlocked asset without data element can’t be locked.

CHAPTER 74

Lineage

This chapter contains the following topics:

1. Overview
2. Display the lineage
3. Browse the lineage

74.1 Overview

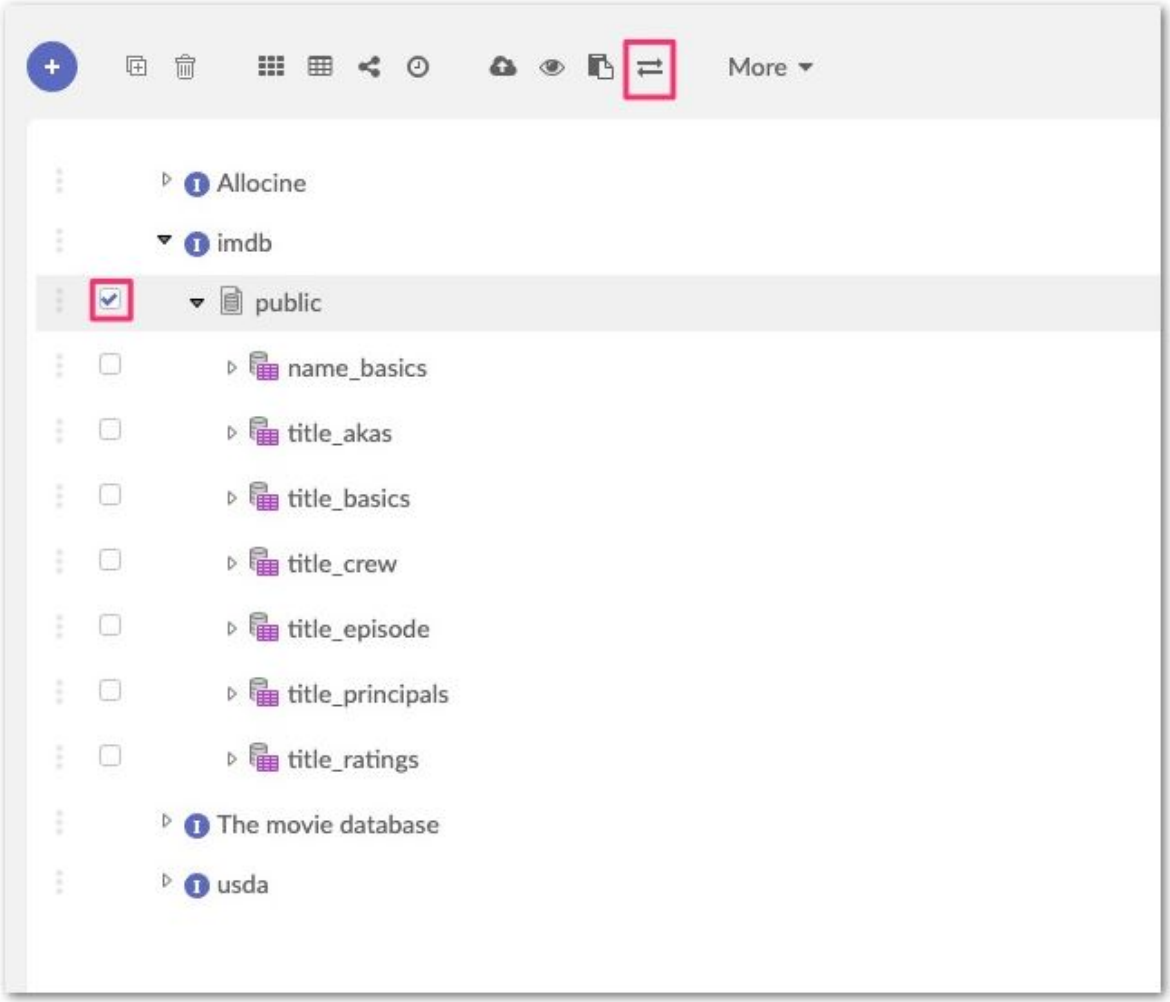
You can display the asset or instance lineage. It’s a main feature that explains how an asset or instance is composed and linked to other assets and instances.

74.2 Display the lineage

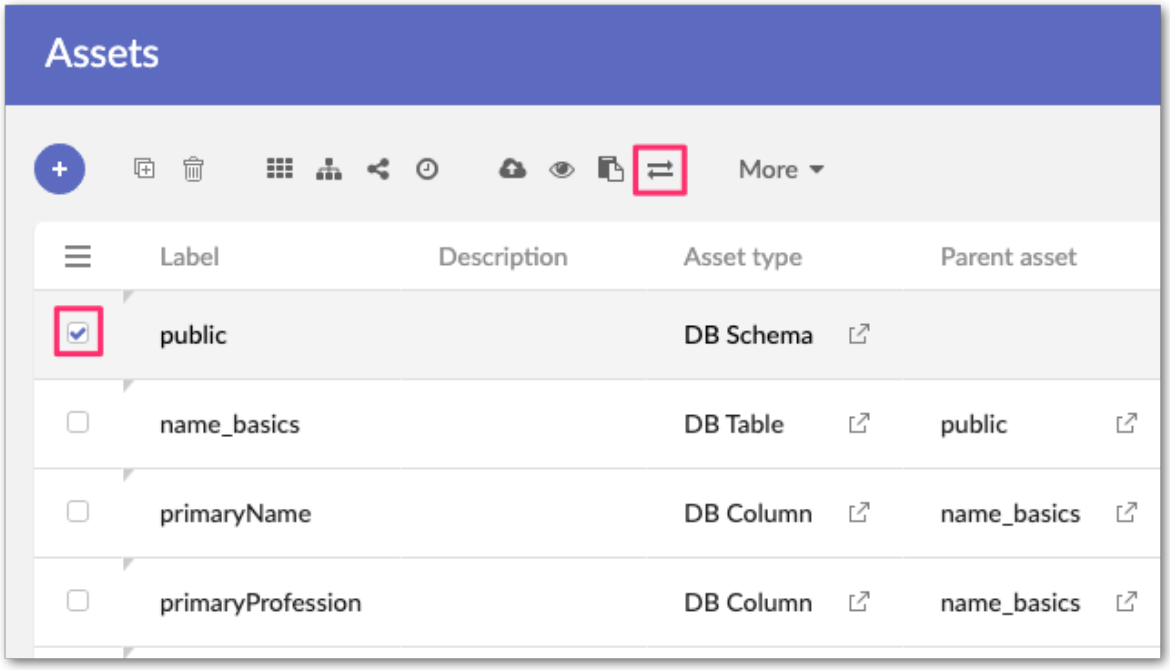
From the asset view

You can start the lineage by selecting an asset in the hierarchical view or the table view. You just have to click on the *lineage* icon on the top of the view.

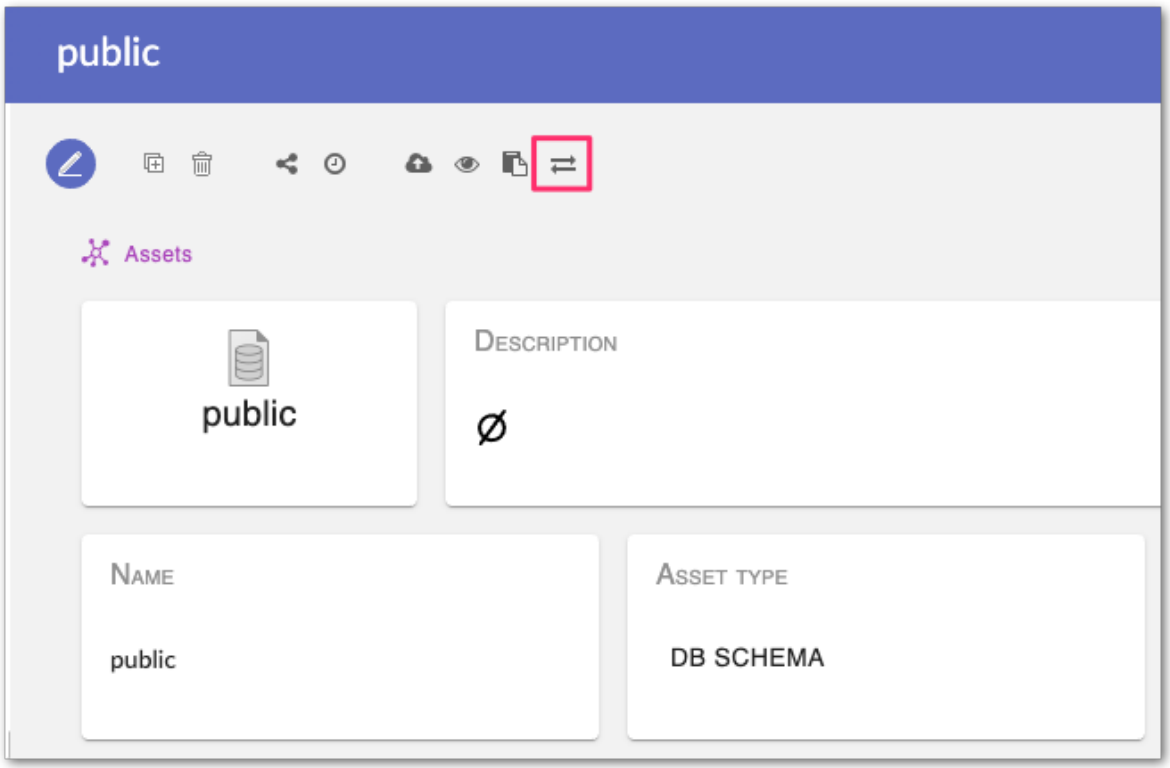
From the hierarchical view:



From the table view:



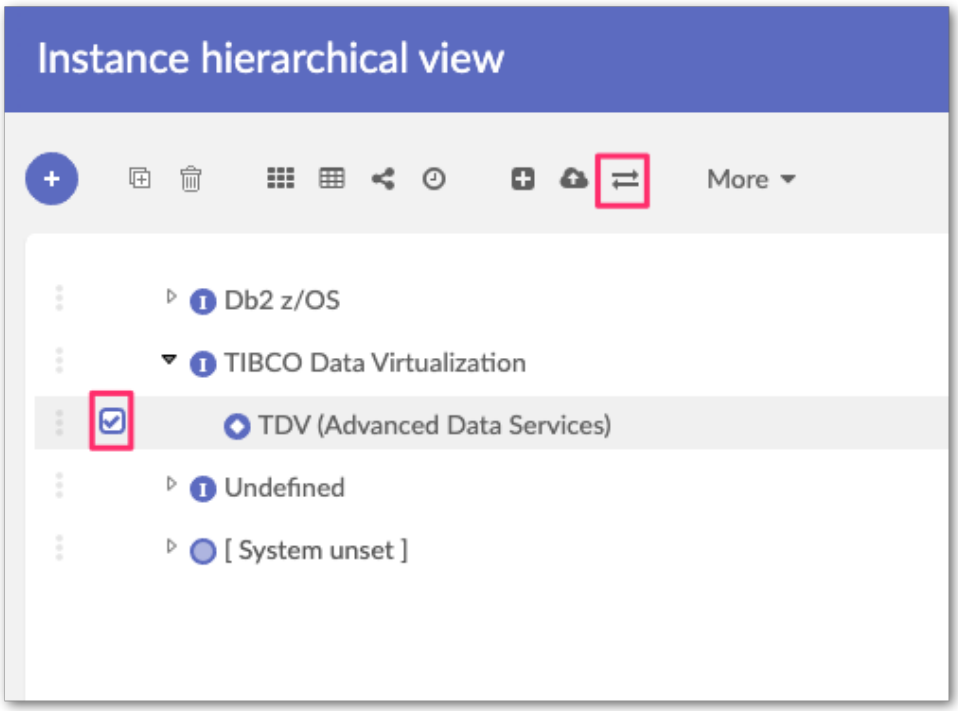
You can also start the lineage while you are reading the asset view form. Click on the *lineage* icon on the top of the asset view form:



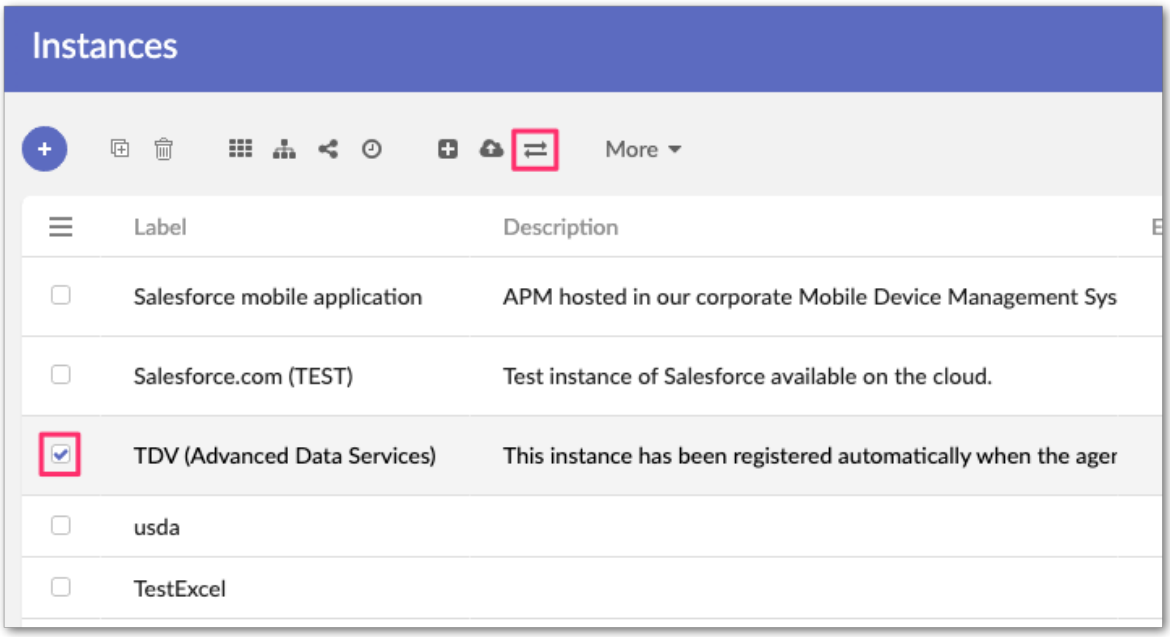
From the instance view

You can start the lineage by selecting an instance in the hierarchical view or the table view. You just have to click on the *lineage* icon on the top of the view.

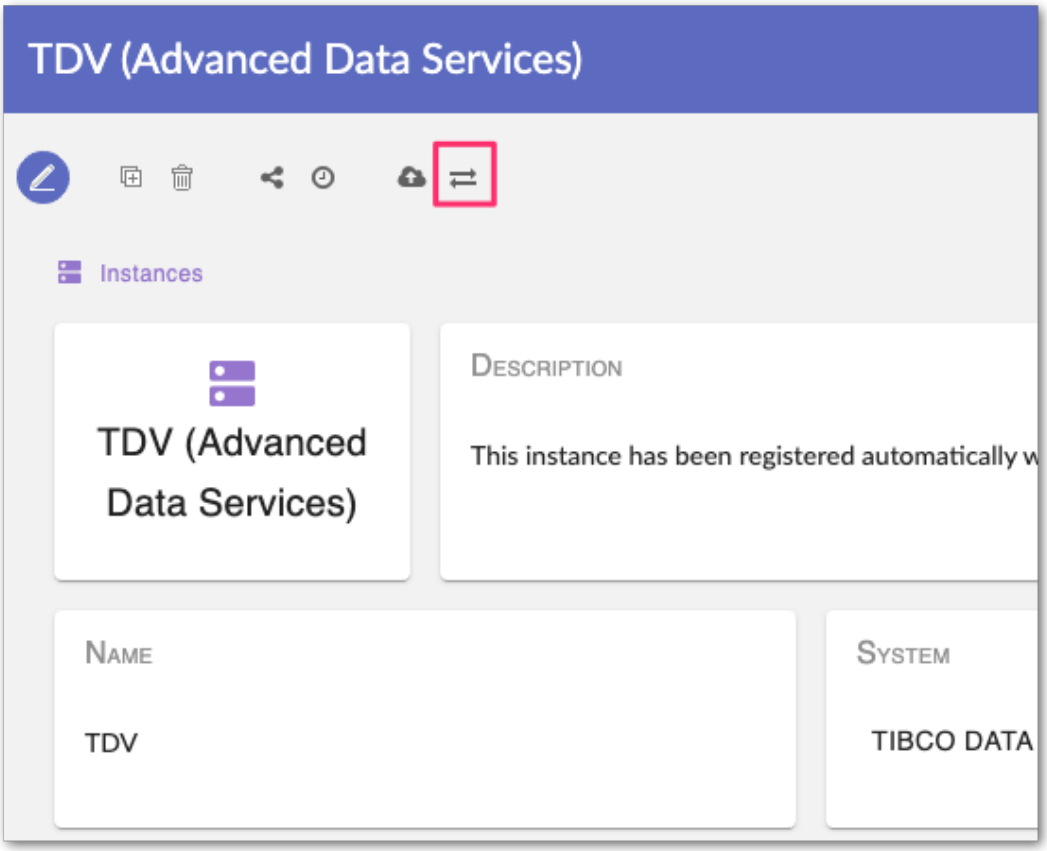
From the hierarchical view:



From the table view:



You can also start the lineage while you are reading the instance view form. Click on the *lineage* icon on the top of the instance view form:

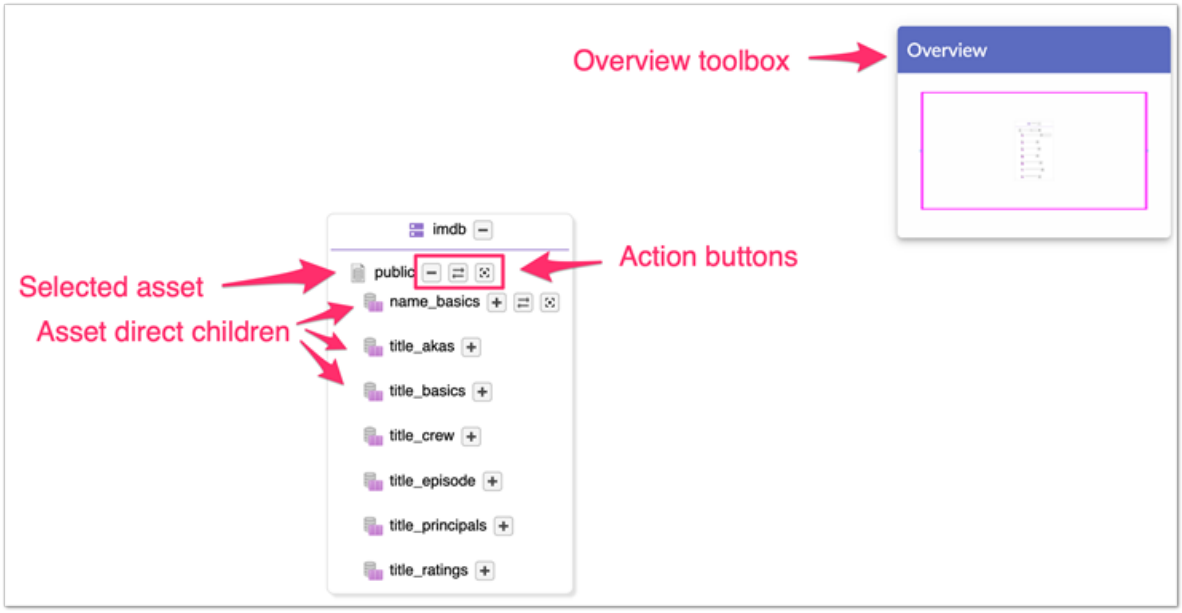


74.3 **Browse the lineage**

Once you have started the lineage service you will see:

- The asset or instance you previously selected: if your selection is an asset, you will see it with its instance.
- Direct children of the instance or asset.
- Action buttons.

- The overview toolbox for helping you browsing the lineage graph.

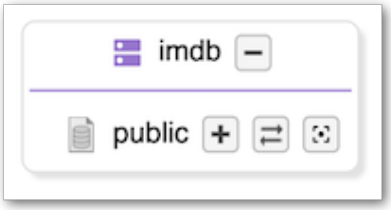


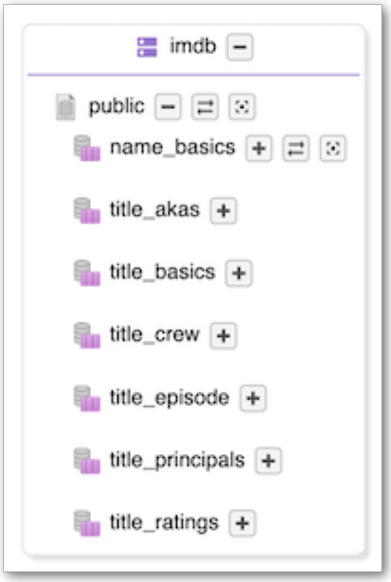
Overview toolbox

The overview toolbox helps you to scroll the lineage graph when this one is big because of multiple nodes and links. You can drag the overview box to any location in the lineage view.

Expand/Collapse a node

By clicking on the + button near the node name, you will expand it to display its children. The + button is only available if children exist.





When a node is expanded, you can collapse it by clicking on the - button near the node name.

Display node links

If a node contains links to display, you will see the *link* icon near the node name (icon with two arrows). By clicking on this button, you will display:

- node incoming links
- node outgoing links

If all links of the node are displayed, the *link* icon is highlighted.



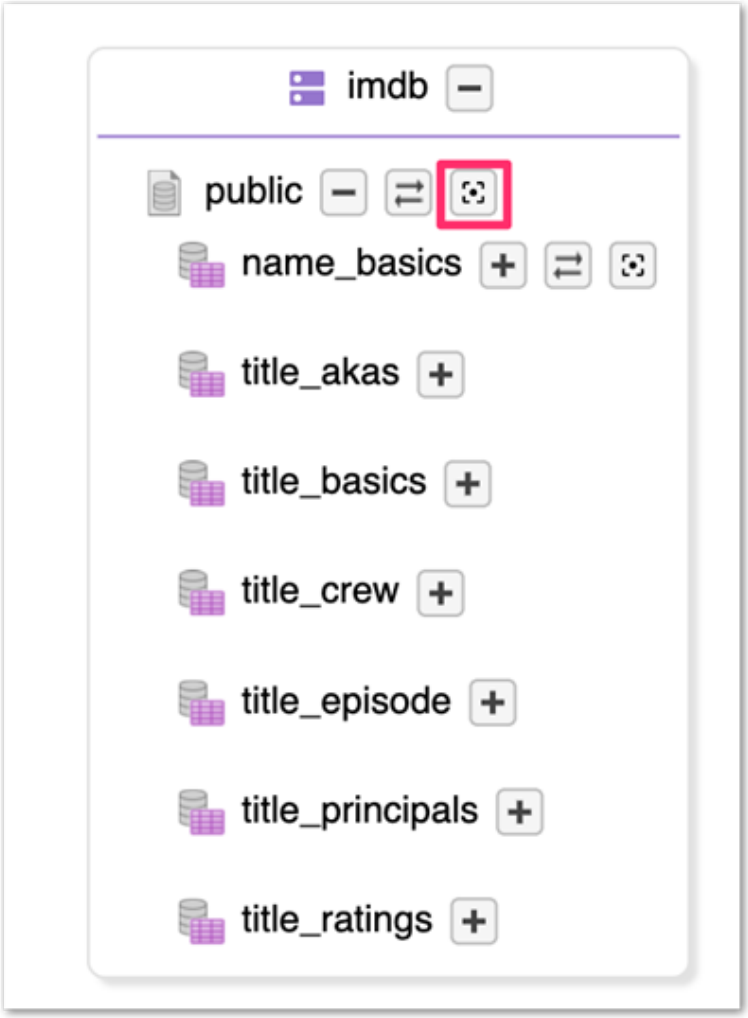
If the lineage is too complex with lot of links, you can click on a node to highlight its links.



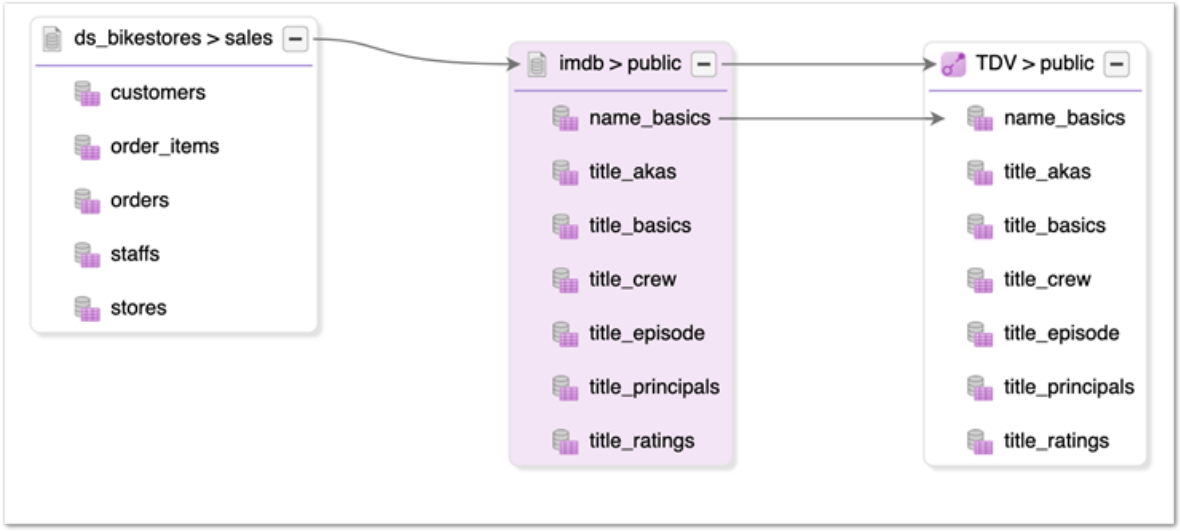
Click anywhere else on the lineage graph to remove link highlighting.

Display full lineage for a node

The first lineage view helps you to browse the lineage step by step (by clicking manually on each link button). This graph is called: *Browsable lineage* If you want to display the full lineage of the selected node, click on the *focus* icon near the node. This button is only available if the node contains links.



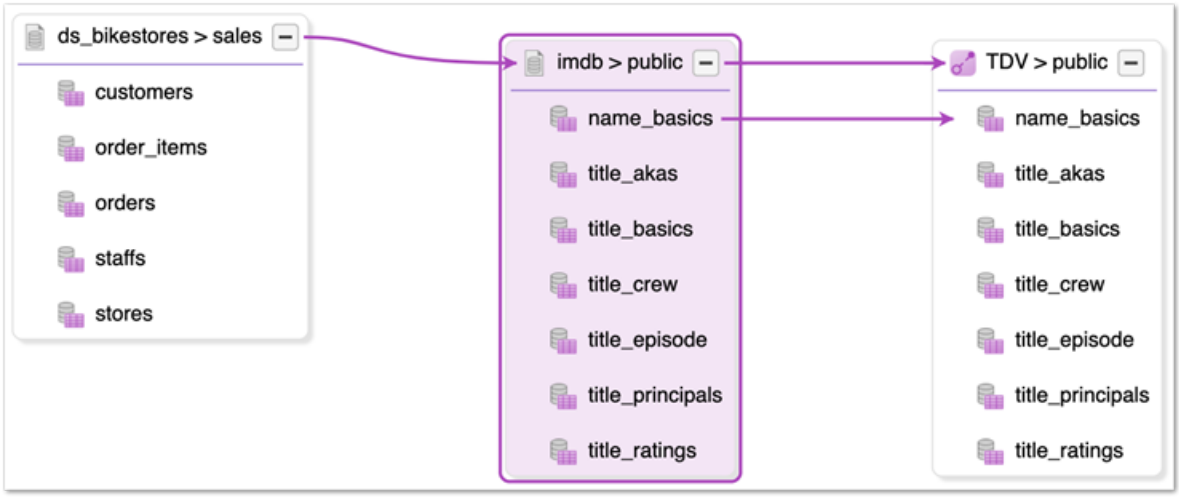
By clicking on this icon, a new lineage graph view will be displayed. This graph is called: *Asset lineage* This view contains the currently selected node and its children. It also displays all links for the node, children nodes, and other nodes recursively.



The node from where you have clicked on the *focus* icon is displayed with a purple background.

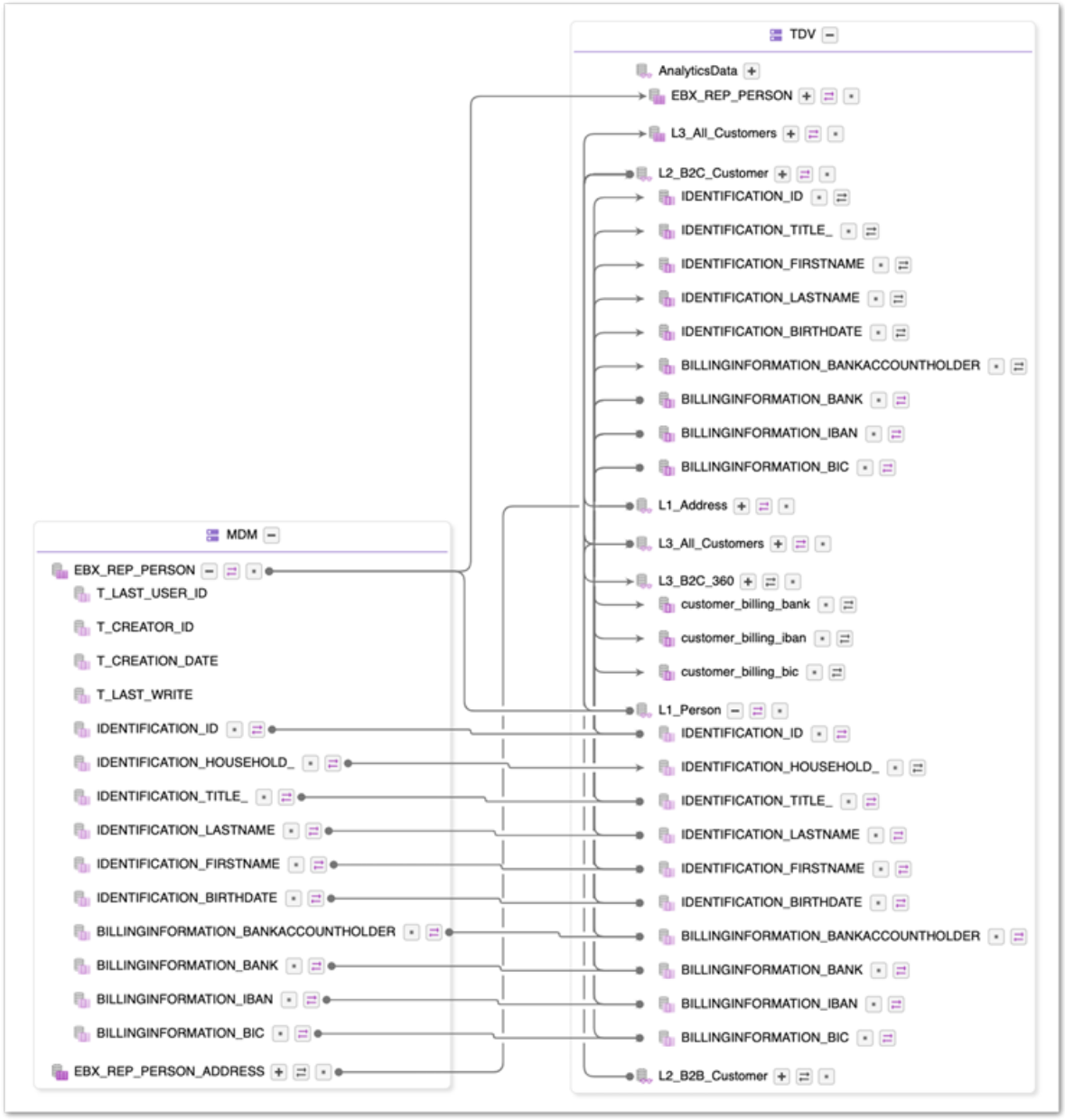
In this graph, every node head title contains the instance name. The template is *Instance Name* > *Node name*.

Click on a node to highlight links if they exist.

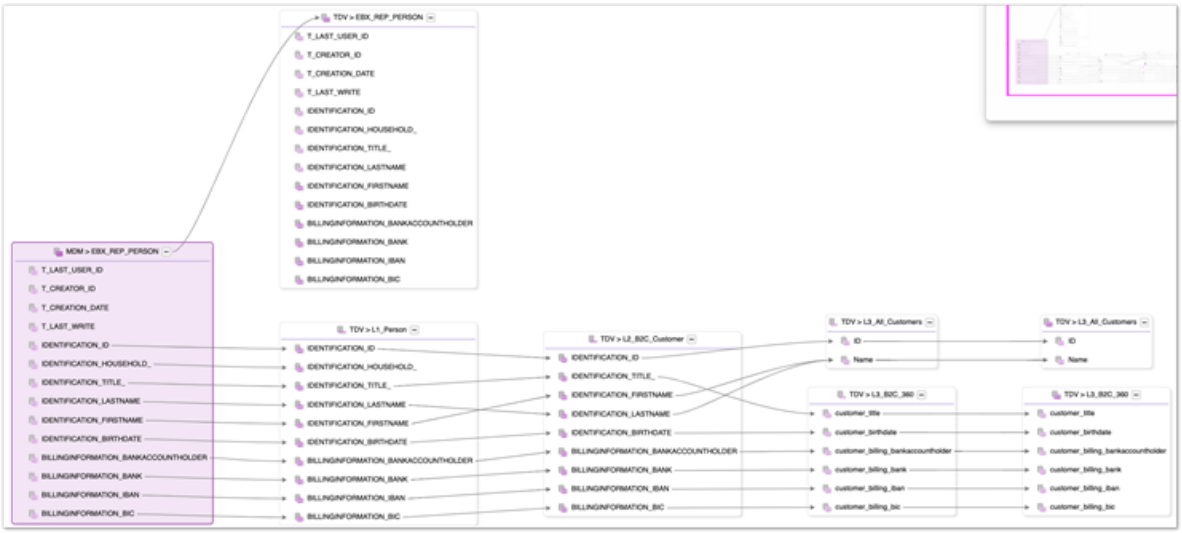


This graph view is different from the previous one because it will not display the default asset hierarchy view but an exploded version (to be more readable).

Below is an example of a complex lineage view with *Browsable lineage* graph and *Asset lineage* graph. In this example, all links from EBX_REP_PERSON node and its children links are displayed recursively.

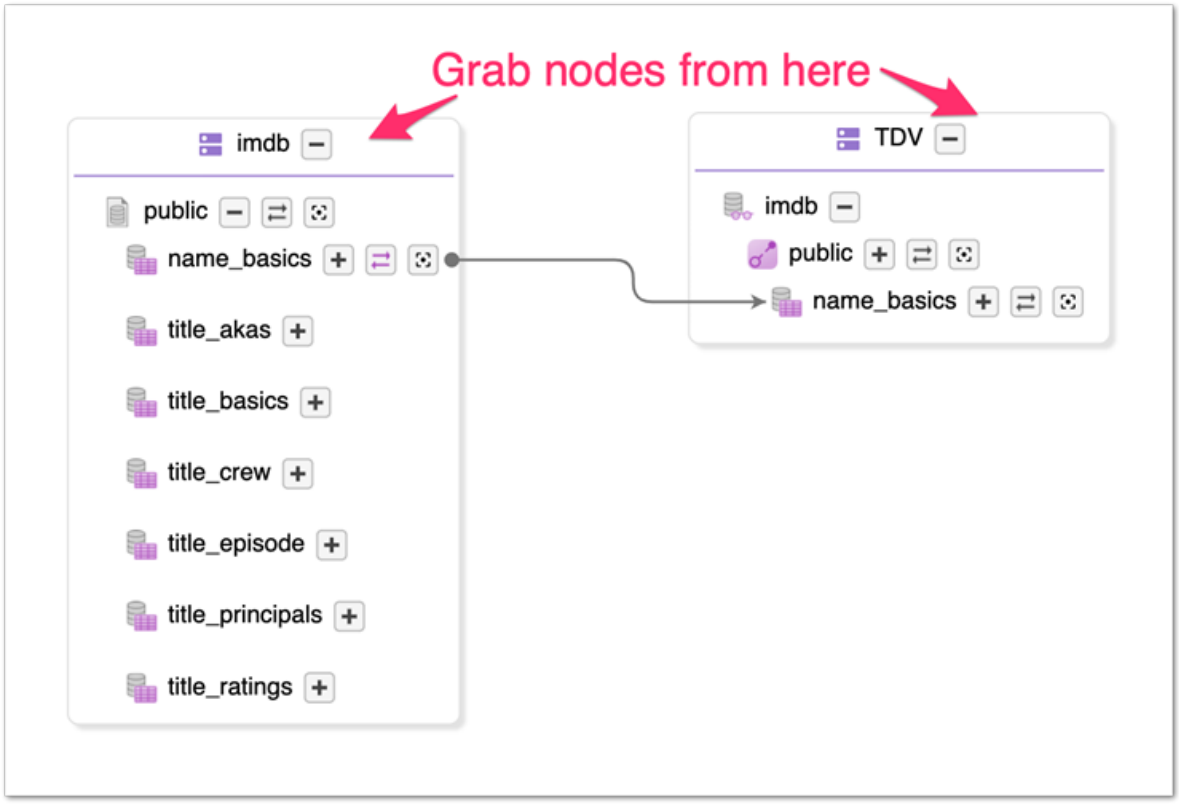


As you can see, the full lineage is difficult to read because each asset node is grouped into the instance node. On the other hand, in the screenshot below, each asset node is separated and the instance name is displayed in the node group title.



Move a node

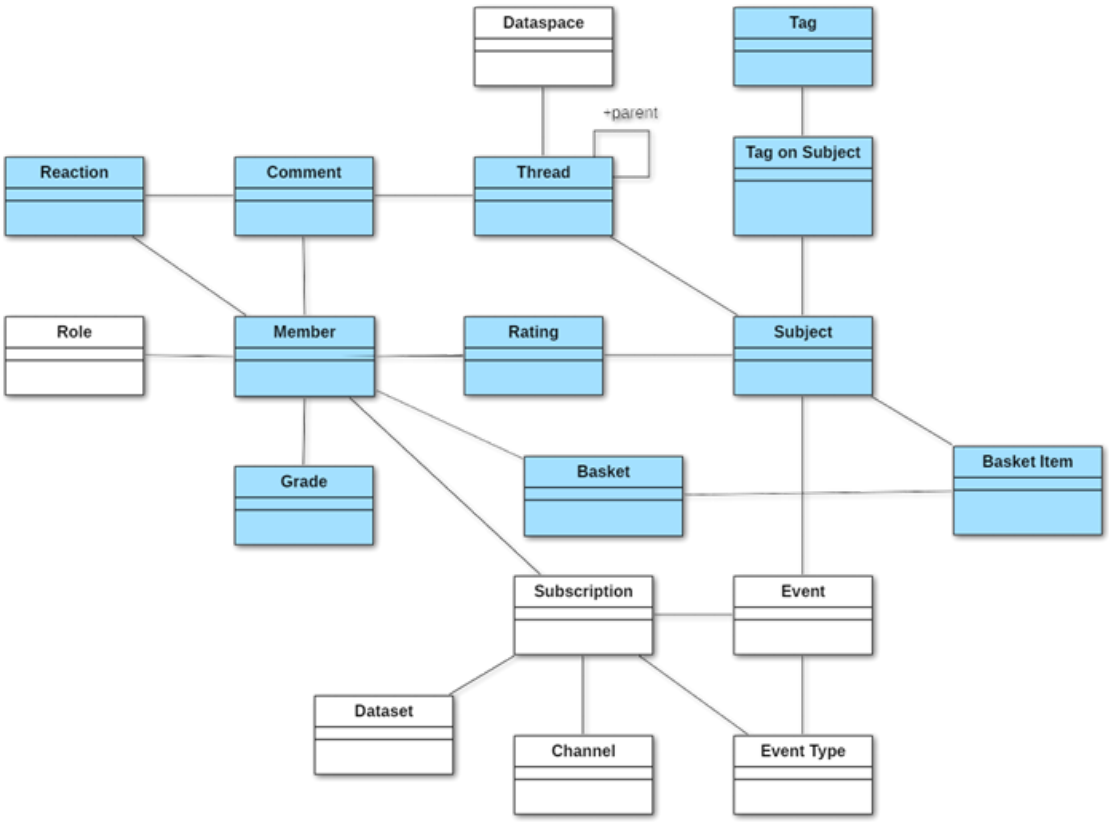
You can move a node by grabbing it with your mouse. You can only move an *Instance* node.



CHAPTER 75

Collaboration overview

This section presents metadata collaboration features, including tags, comments, ratings and the workflows associated with these features.

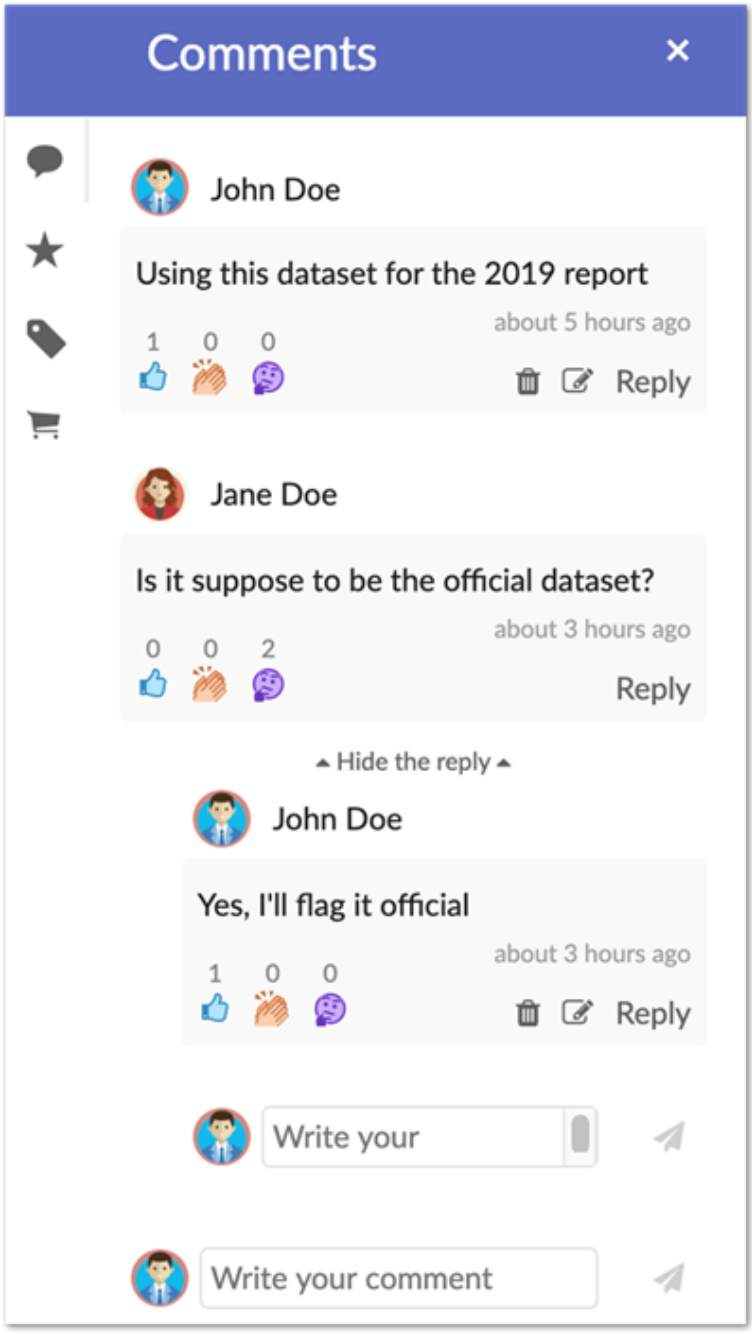


CHAPTER 76

Metadata comments

Users can comment on records and these comments can turn into discussions that help users better understand the metadata. They can also use comments to request specific information.

The comment feature is accessed via the sidebar located to the right of the screen when viewing a record.



Comments are displayed in chronological order, oldest comment first. The following actions are available on comments:

- Read comments posted on a given record.
- Write comments on a given record. The maximum length of the comment is currently 255 characters. You can also edit or delete your own comments.
- Reply to any comments, even your own. There is only one level of reply available. It is not possible to have a sub-reply thread.

- React to a comment. A reaction is represented by an emoticon and a label: Agree, Congrats, Interesting. Reactions are displayed below the comment with a counter of the number of users applying a particular reaction. Currently, only one reaction per user is allowed, if a user applies a different reaction, the previous one will be removed. The user can remove their reaction at any time.

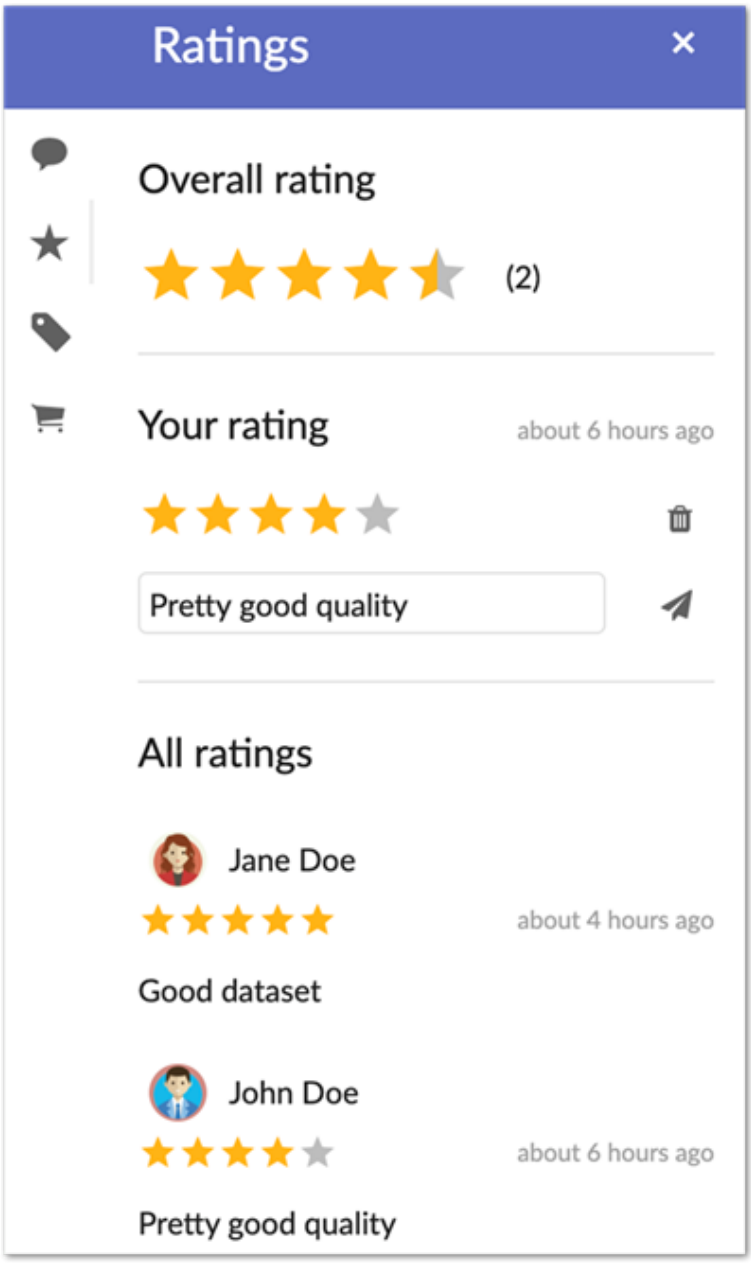
CHAPTER 77

Metadata ratings

Users can leave ratings on records to express their evaluation of the data. The overall rating is the average of all ratings for a given record. It helps users searching for metadata to better understand whether it is trustworthy.

For instance, a user can say a dataset is of good quality, or a particular attribute is not useful because it is usually empty.

The ratings feature can be accessed via the sidebar located to the right of the screen on any record.



The list of ratings from all users is displayed in chronological order, latest updated rating first. The following actions are available on ratings:

- View the overall rating of a given record.
- See a list of ratings from all users for a given record.
- Apply, modify or delete their own record ratings. The user can rate with a value between 1 and 5. Alongside the rating value, he can leave a justification comment. The overall rating of the record is updated accordingly and the list of all ratings is refreshed.

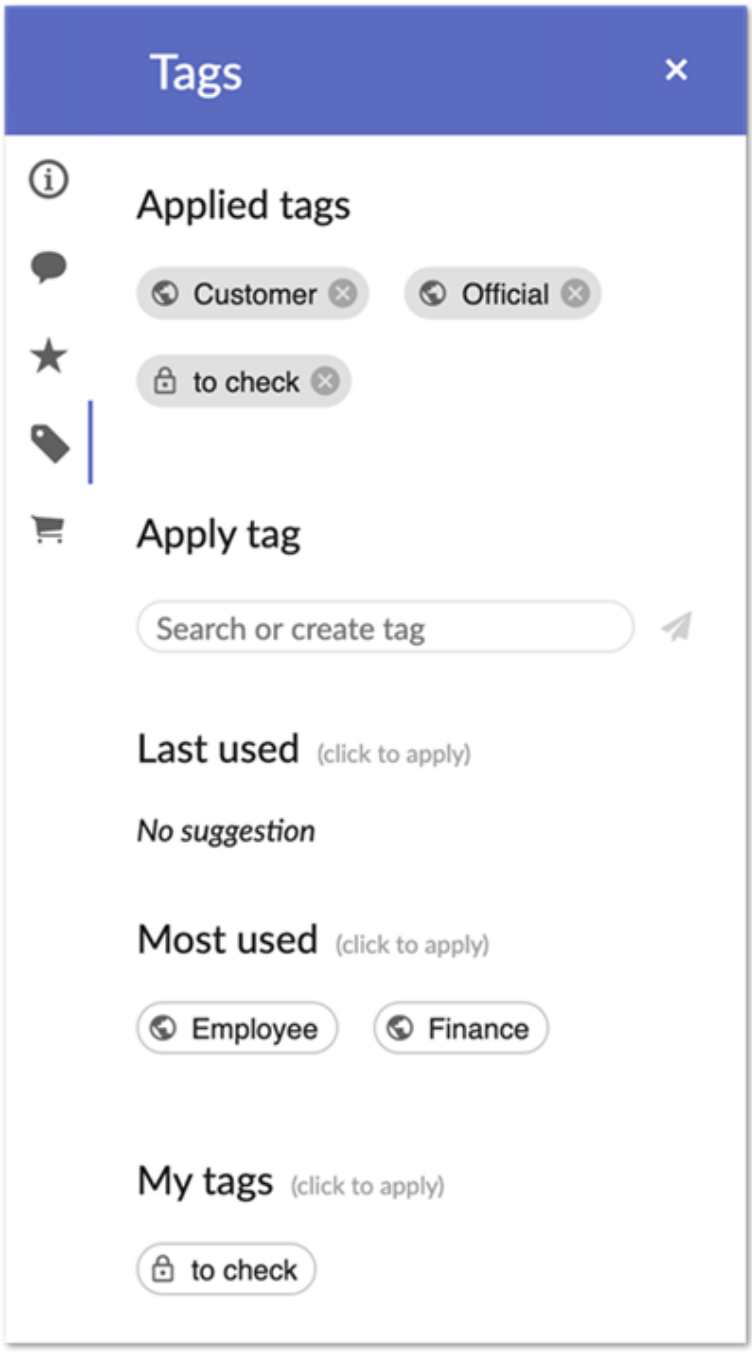
CHAPTER 78

Tag metadata

Users can tag records and one record can have multiple tags. Tags are useful for users to provide additional contextual information that defines the metadata. Additionally, you can filter metadata lists using tags.

Note: tags can be words or phrases (groups of words), they can also include special characters.

The Tags feature can be accessed via the sidebar located to the right of the screen on any record.



There are two types of tag *Public* and *Private*. Each type corresponds to the visibility of the tag to the users. While *Public* tags are visible and usable by any users, the *Private* tags are only visible to the user that created it. The icon on the left side indicates the type of visibility: the world for *Public* and the lock for *Private*.

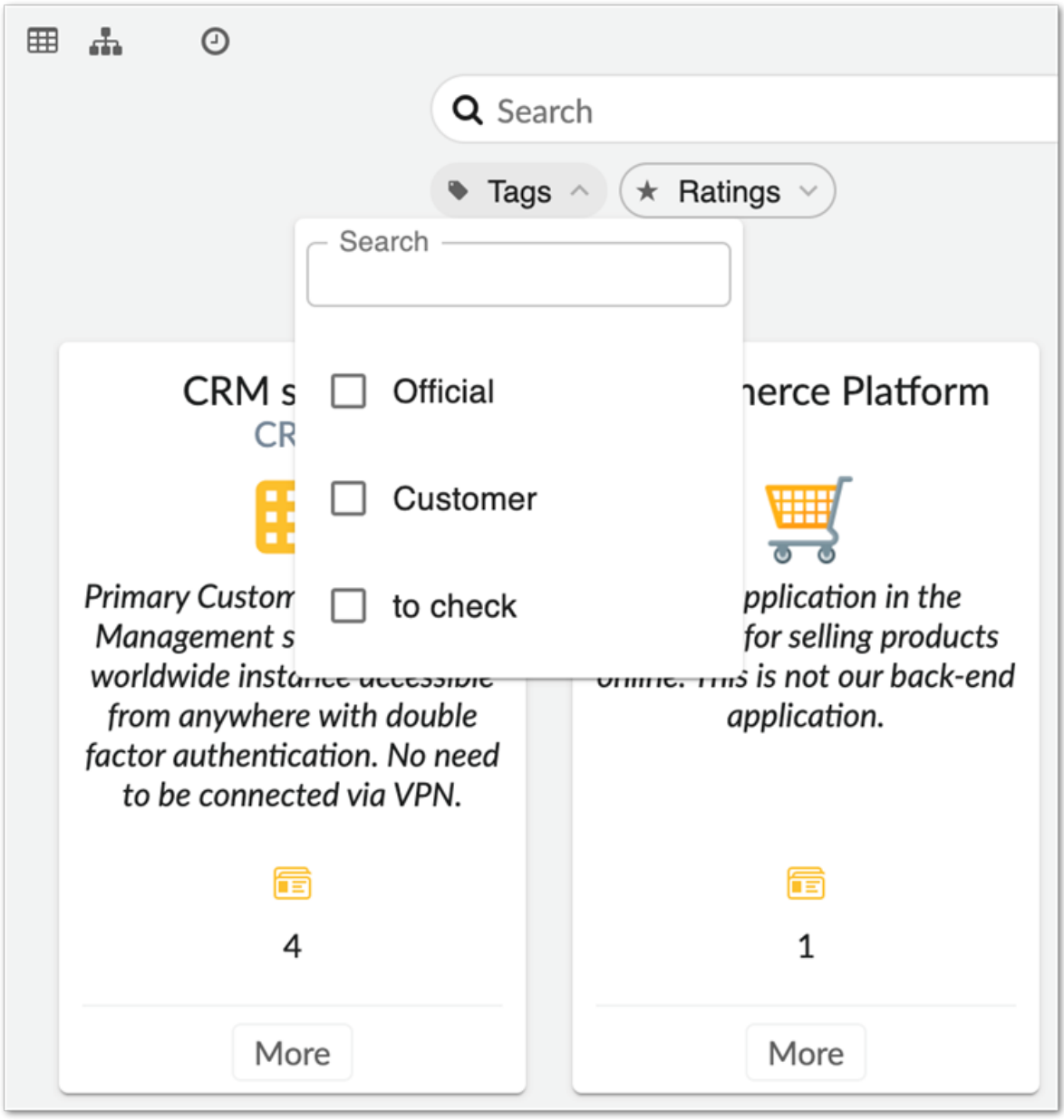
The top section **Applied tags** displays the tags applied to the current record. You can remove an applied tag by clicking on the cross button on the right of the tag.

The **Apply tag** section allows users to create a new tag or search for an existing one and apply it to the record. Note that creating *Public* tags is controlled by permissions configured by the administrator.

All users might not have the ability to create them. For those allowed to create *Public* tags, they can choose the visibility after clicking the submit button.

The **Last used** section displays the last ten tags that have been used by the current user. The **Most used** section displays the ten tags the most used in the whole repository. Clicking on a tag applies it to the current record. This action is performed real-time.

Finally, the **My tags** section displays the list of all the tags the connected user has created.



The card view integrates the tags allowing users to filter records based on the tags that have been applied to them.

CHAPTER 79

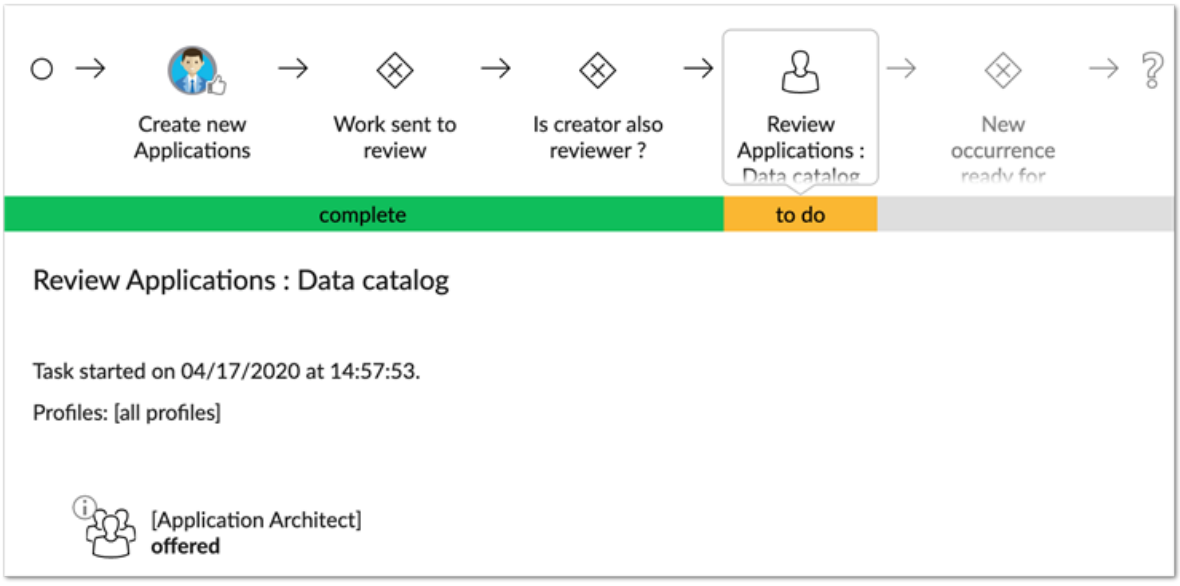
Managing metadata with collaborative workflows

Your organization's administrators can define the permissions for users to create and update records. They can require the use of collaborative workflows to perform such operations.

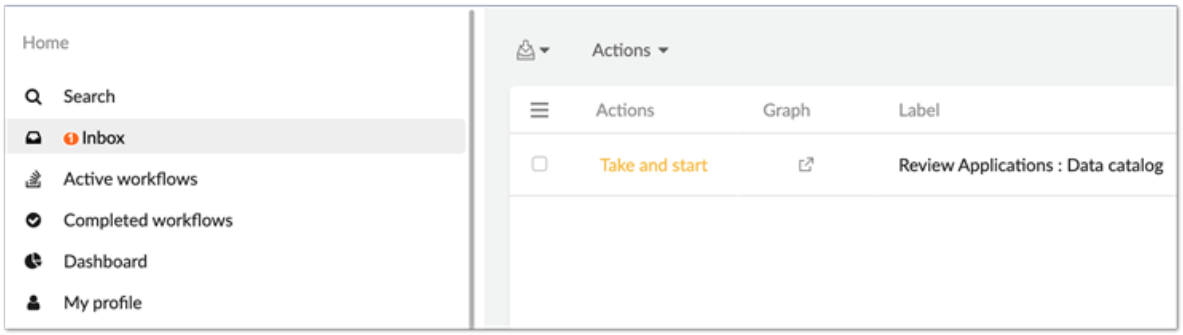
Creation and Update workflows have several steps until completion, namely:

- Creation/Update
- Review
- Approval (possibly multiple in parallel)
- Finalize

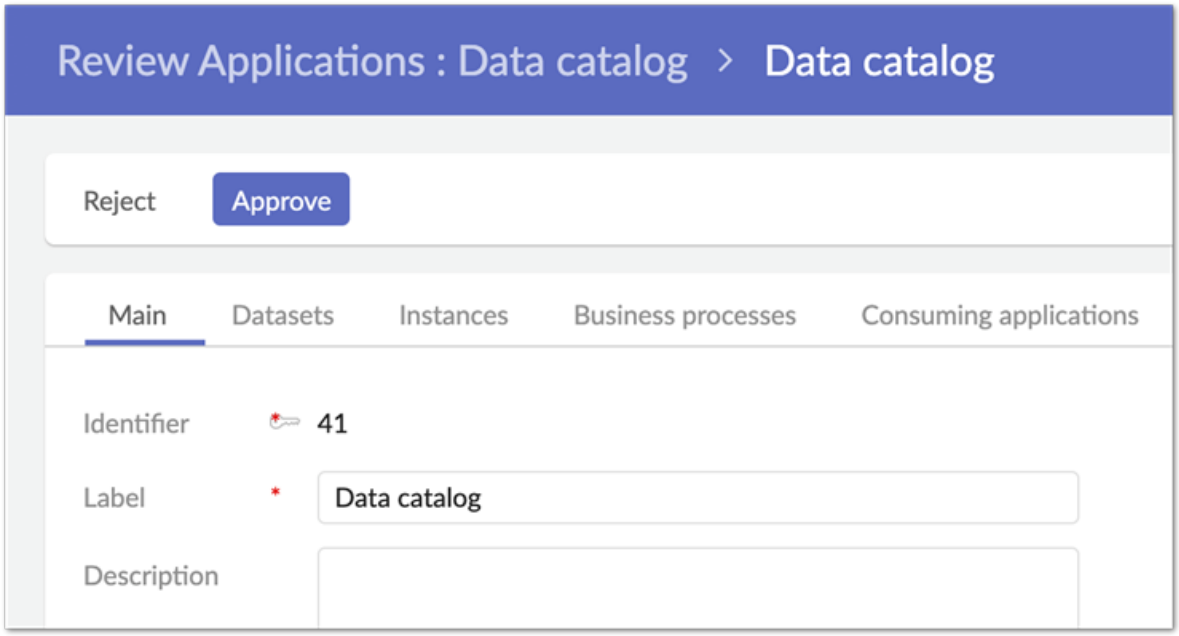
Those steps might or might not be activated for your organization and for a particular entity. Contact your administrator for more information.



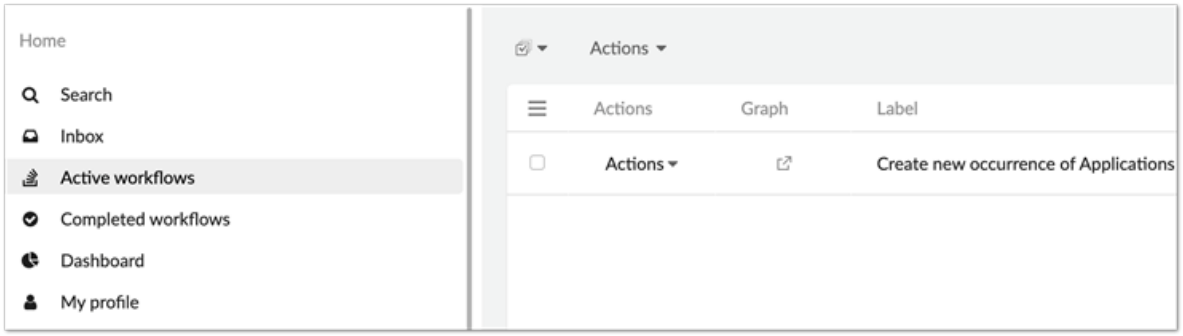
When you are involved in a workflow, a task will be offered to you and other users with the same role, you will see it in your **Inbox**. You can select **Take and Start** to get access to it. Once you take a task, it is assigned to you privately.



Within the task, the actions displayed at the top depend on the type of task. For instance, Reject/ Approve.



You can access the active workflows from the left navigation menu. Only the workflows you have access to are listed. From this list of active workflows, you can display the graph of the workflow to check which step is being performed.



Reference Manual

Integration

CHAPTER 80

Overview of integration and extension

Several service and component APIs allow you to develop custom extensions for TIBCO EBX® and integrate it with other systems.

This chapter contains the following topics:

1. User interface customization and integration
2. Data services
3. XML and CSV import/export services
4. Programmatic services

80.1 User interface customization and integration

The EBX® graphical interface can be customized through various EBX® APIs.

It can also be integrated into any application that is accessible through a Supported web browsers [p 524].

See Interface customization [p 1024] for more information.

80.2 Data services

The data services provides a means for external systems to interact with EBX® using one of following:

- Web Services Description Language (WSDL/SOAP) standard
- Representational state transfer (REST)

See also

WSDL/SOAP data services [p 1060]

REST data services [p 1114]

80.3 XML and CSV import/export services

EBX® includes built-in services for importing data from and export data to XML and CSV formats. Imports and exports for XML and CSV can be performed using:

- the user interface
- the data services
- the Java API

See also

XML import and export *[p 153]*

CSV import and export *[p 159]*

80.4 Programmatic services

Programmatic services allow executing procedures in a well-defined context, for example in a scheduled task or in a batch.

Some examples of programmatic services include:

- Importing data from an external source,
- Exporting data to multiple systems,
- Data historization, launched by a supervisory system,
- Optimizing and refactoring data.

  If EBX® **built-in optimization services** `AdaptationTreeOptimizerSpec`[API] are not sufficient.

See also *`ProgrammaticService`*[API]

CHAPTER 81

Using TIBCO EBX® as a Web Component

This chapter contains the following topics:

1. Overview
2. Integrating EBX® Web Components into applications
3. Repository element and scope selection
4. Combined selection
5. Request specifications
6. Example calls to an EBX® Web Component

81.1 Overview

EBX® can be used as a user interface Web Component, called through the HTTP protocol. An EBX® Web Component can be integrated into any application that is accessible through a supported web browser. This method of access offers the major benefits of EBX®, such as user authentication, data validation, and automatic user interface generation, while additionally providing the ability to focus user navigation on specific elements of the repository.

Typical uses of EBX® Web Components include integrating them into the intranet frameworks of organizations or into applications that manage the assignment of specific tasks to users.

See also *Supported web browsers* *[p 524]*

81.2 Integrating EBX® Web Components into applications

A web application that calls an EBX® Web Component can be:

1. A non-Java application, the most basic being a static HTML page.

   In this case, the application must send an HTTP request that follows the EBX® Web Component request specifications [p 421].

2. A Java application, for example:
   - A Java web application running on the same application server instance as the EBX® repository it targets or on a different application server instance.

- An EBX® User service [p 1024] or a Custom widget [p 1025], in which case, the new session will automatically inherit from the parent EBX® session.

> **Note**
>
> In Java, the recommended method for building HTTP requests that call EBX® web components is to use the class `UIHttpManagerComponent`[API] in the API.

81.3 Repository element and scope selection

When an EBX® Web Component is called, the user must first be authenticated in the newly instantiated HTTP session. The Web Component then selects a repository element and displays it according to the `scope` layout parameter defined in the request.

The parameter `firstCallDisplay` may change this automatic display according to its value.

The repository elements that can be selected are as follows:

- Dataspace or snapshot
- Dataset
- Node
- Table or a published view
- Table record

The scope determines how much of the user interface is displayed to the user, thus defining where the user is able to navigate in the session. The default scope that the Web component uses is the smallest possible depending on the entity or service being selected or invoked by the request.

See also *Scope [p 424]*

See also *firstCallDisplay [p 424]*

It is also possible to select a specific perspective as well as a perspective action.

By default, the selection of the element is done in the context of the perspective of the user if the scope is "full".

See also *Perspective [p 21]*

81.4 Combined selection

A URL of a Web component can specify a perspective and an action or an entity (dataspace, dataset, etc). Thus, for a Web component that has specified in its URL a perspective and an entity (but no action), if an action of the perspective matches this entity, then this action will be automatically selected.

Otherwise, if no action matches this entity, no action will be selected but the entity is opened regardless.

If an action is specified at the same time than an entity, this last is ignored and the action will be selected.

Specific case

If the target entity is a record and if an action is on the table that contains this record, then this action will be selected and the record will be opened inside the action.

In the same way, if a workflow work item is targeted by the web component, and if an action on « inbox » exists in the perspective, then this action will be selected and the work item will be opened inside it.

Known limitations

If the Web component specifies a predicate to filter a table, the perspective action must specify the exact same predicate to be selected.

In the same way, if the perspective action specifies a predicate to filter a table, the Web component must specify the exact same predicate to establish the match.

81.5 Request specifications

Base URL

In a default deployment, the base URL must be of the following form:

`http://<host>[:<port>]/ebx/`

> **Note**
>
> The base URL must refer to the servlet `FrontServlet`, defined in the deployment descriptor `/WEB-INF/web.xml` of the web application `ebx.war`.

User authentication and session information parameters

Parameter	Description	Required
`login` and `password`, or a *user directory-specific token*	Specifies user authentication properties. If neither a login and password pair nor a user directory-specific token is provided, user will be required to authenticate through the repository login page. See `Directory`[API] for more information.	No
`trackingInfo`	Specifies the tracking information of the new session. Tracking information is logged in history tables. Additionally, it can be used to programmatically restrict access permissions. See `AccessRule`[API] for more information.	No
`redirect`	The URL to which the user will be redirected at the end of the component session, when they click on the button 'Close'. The close button is always displayed for record selections, but whether or not it is displayed for all other cases must be specified using the parameter `closeButton`. For more information, see Exit policy (deprecated) [p 675].	No
`locale`	Specifies the locale to use. Value is either `en-US` or `fr-FR`.	No, default is the locale registered for the user.

Entity and service selection parameters

Parameter	Description	Required
`branch`	Selects the specified dataspace.	No
`version`	Selects the specified snapshot.	No
`instance`	Selects the specified dataset. The value must be the reference of a dataset that exists in the selected dataspace or snapshot.	Only if `xpath` or `viewPublication` is specified.
`viewPublication`	Specifies the publication name of the tabular, hierarchical, or tile view to apply to the selected content. This publication name is the one declared during the publication of the view. It can be found in the 'Administration' area under *Views configuration > Views*. All settings of the view, that is, its filters, sort order, and displayed columns, are applied to the result. A dataspace and a dataset must be selected in order for this view to be applied. The target table selection is not necessary, as it can be automatically determined based on the definition of the view. This parameter can be combined with the predicate specified in the `xpath` parameter as a logical 'AND' operation.	No
`xpath`	Specifies a node selection in the dataset. Value may be a valid absolute path located in the selected dataset. The notation must conform to a simplified XPath, with abbreviated syntax. It can also be a predicate surrounded by "[" and "]" (to be encoded using %5B and %5F respectively) if a table can be automatically selected using other Web Component parameters (for example, `viewPublication` or `workflowView`). For XPath syntax, see XPath supported syntax [p 441] See `UIHttpManagerComponent.setPredicate`[API] for more information.	No
`service`	Specifies the service to access. For more information on built-in User services, see Built-in services [p 427]. In the Java API, see `ServiceKey`[API] for more information.	No
`workflowView`	Specifies the workflow section to be selected. See `WorkflowView`[API] for more information.	No.
`perspectiveName`	Specifies the name of the perspective to be selected. If this parameter is specified, the `scope` parameter can have only two values: full and data.	Only if `perspectiveActionId` or `perspectiveActionName` is specified.
`perspectiveActionId`	**Deprecated.** Please consider using `perspectiveActionName` instead. Specifies the identifier of the perspective action to be selected.	No.

Parameter	Description	Required
`perspectiveActionName`	Specifies the unique name of the perspective action to be selected.	No.

Layout parameters

Parameter	Description	Required
`scope`	Specifies the scope to be used by the web component. Value can be `full`, `data`, `dataspace`, `dataset` or `node`. See `UIHttpManagerComponent.Scope`[API] for more information.	No, default will be computed to be the smallest possible according to the target selection.

`firstCallDisplay`	Specifies which display must be used instead of the one determined by the combination of selection and `scope` parameter.
	Possible values are:
	• `auto`: The display is automatically set according to the selection.
	• `view`: Forces the display of the tabular, the hierarchical or the tile view.
	• `record`: If the predicate has at least one record, forces the display of the first record in the list.
	For example,
	`firstCallDisplay=view`
	`firstCallDisplay=view:hierarchyExpanded`
	`firstCallDisplay=view:hierarchyExpandedToSearch`
	`firstCallDisplay=record`
	`firstCallDisplay=record:{predicate}`
	See `UIHttpManagerComponent.setFirstCallDisplay`[API] for more information.
	See `UIHttpManagerComponent.setFirstCallDisplayHierarchyExpanded`[API] for more information.
	See `UIHttpManagerComponent.setFirstCallDisplayHierarchyExpandedToSearch`[API] for more information.
	See `UIHttpManagerComponent.setFirstCallDisplayRecord`[API] for more information.
		No, default will be computed according to the target selection.

`closeButton`	Specifies how to display the session close button. Value can be `logout` or `cross`. See `UIHttpManagerComponent.CloseButtonSpec`[API] for more information.	No. If scope is not `full`, no close button will be displayed by default.

`dataSetFeatures`	Specifies which features to display in a UI service at the dataset level or a form outside of a table.
	These options pertain only to features in the workspace. It is recommended to use this property with the smallest `scope` possible, namely `dataset` or `node`.
	Syntax:
	`<prefix> ":" <feature> [ "," <feature>]*`
	where
	• `<prefix>` is `hide` or `show`,
	• `<feature>` is `services`, `title`, `save`, or `revert`.
	For example,
	`hide:title`
	`show:save,revert`
	See `UIHttpManagerComponent.DataSetFeatures`[API] for more information.
		No.

Parameter	Description	Required

`viewFeatures`	Specifies which features to display in a tabular, a hierarchy or a tile view (at the table level).
	These options pertain only to features in the workspace. It is recommended to use this property with the smallest `scope` possible, namely `dataset` or `node`.
	Syntax:
	`<prefix> ":" <feature> [ "," <feature>]*`
	where
	• `<prefix>` is `hide` or `show`,
	• `<feature>` is `create`, `views`, `selection`, `filters`, `services`, `refresh`, `title`, or `breadcrumb`.
	For example,
	`hide:title,selection`
	`show:service,title,breadcrumb`
	See `UIHttpManagerComponent.ViewFeatures`[API] for more information.
		No.

`recordFeatures`	Specifies which features must be displayed in a form at the record level.
	These options pertain only to features in the workspace. It is recommended to use this property with the smallest `scope` possible, namely `dataset` or `node`.
	Syntax:
	`<prefix> ":" <feature> [ "," <feature>]*`
	where
	• `<prefix>` is `hide` or `show`,
	• `<feature>` is `services`, `title`, `breadcrumb`, `save`, `saveAndClose`, `close`, or `revert`.
	For example,
	`hide:title`
	`show:save,saveAndClose,revert`
	See `UIHttpManagerComponent.RecordFeatures`[API] for more information.
		No.

`pageSize`	Specifies the number of records that will be displayed per page in a table view (either tabular, hierarchical or tile).	No.
`startWorkItem`	Specifies a work item must be automatically taken and started. Value can be `true` or `false`. See `ServiceKey.WORKFLOW`[API] for more information.	No. Default value is `false`, where the target work item state remains unchanged.

81.6 Example calls to an EBX® Web Component

Minimal URI:

```
http://localhost:8080/ebx/
```

Logs in as the user 'admin' and selects the 'Reference' dataspace:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=Reference
```

Selects the 'Reference' dataspace and accesses the built-in validation service:

```
http://localhost:8080/ebx/?
login=admin&password=admin&branch=Reference&service=@validation
```

Selects the roles table in the default directory:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/roles
```

Selects the record 'admin' in the default directory:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/users[./login="admin"]
```

> **Note**
>
> For clarity purposes, the above URLs are not encoded and this can make them incompatible with some application servers.

Accesses the interface for creating a new user in the default directory:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/users&service=@creation
```

Compares the record 'admin' in the default directory with the record 'jSmith':

Compares the record 'R1' in the dataset 'instanceId' in the dataspace 'Reference' with the record 'R0':

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/users[./
login="admin"]&service=@compare&compare.branch=ebx-directory&compare.instance=ebx-
directory&compare.xpath=/directory/users[./login="jSmith"]
```

> **Note**
>
> For clarity purposes, the above URLs are not encoded and this can make them incompatible with some application servers.

CHAPTER 82

Built-in user services

EBX® includes a number of built-in user services. Built-in user services can be used:

- when defining workflow model tasks [p 200]
- when defining perspective action menu items [p 22]
- as extended user services when used with service extensions [p 1051]
- when using EBX® as a Web Component [p 419]

This reference page describes the built-in user services and their parameters.

This chapter contains the following topics:

1. Access data (default service)
2. Create a new record
3. Duplicate a record
4. Export data to an XML file
5. Export data to a CSV file
6. Import data from an XML file
7. Import data from a CSV file
8. Access a dataspace
9. Validate a dataspace, a snapshot or a dataset
10. Merge a dataspace
11. Access the dataspace merge view
12. Compare contents
13. Data workflows

82.1 Access data (default service)

By default, workflows automatically consider this service as complete. That is, the 'Accept' button is always available.

This is the default service used if no service is specified.

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`disableAutoComplete`	Disable Accept at start	By default, the interaction associated with this service is directly considered as complete. Therefore, the Accept button is automatically displayed at the opening of the work item. This parameter is useful to disable this behavior. If the value is 'true', the developer will be in charge of completing the interaction by using SessionInteraction in a user service or a trigger, for example. The default value is 'false'. Perspectives do not use this parameter.
`firstCallDisplay`	First call display mode	Defines the display mode that must be used when displaying a filtered table or a record upon first call. Default (value = 'auto'): the display is automatically set according to the selection. View (value = 'view'): forces the display of the tabular view or of the hierarchical view. Record (value = 'record'): if the predicate has at least one record, forces the display of the record form.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.
`viewPublication`	View	The publication name of the view to display. The view must be configured for the selected table.
`xpath`	Dataset node (XPath)	The value must be a valid absolute location path in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax.

82.2 Create a new record

For a workflow, the creation service is considered complete when the first successful submit is performed (record has been created). If this service is called whereas it is already complete, the created record is displayed in update or read-only mode (depending on the user rights).

Service name parameter: `service=@creation`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - This field is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Dataset table (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Output parameters

Parameter	Label	Description
`created`	Created record	Contains the XPath of the created record.

82.3 Duplicate a record

For a workflow, the duplicate service is considered complete when the first successful submit is performed (record has been created). If this service is called whereas it is already complete, the created record is displayed in update or read-only mode (depending on the user rights).

Service name parameter: `service=@duplicate`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - This field is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Record to duplicate (XPath)	The value must be a valid absolute location path of an existing record. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Output parameters

Parameter	Label	Description
`created`	Created record	Contains the XPath of the created record.

82.4 **Export data to an XML file**

The exportToXML service is considered complete when export is done and file downloaded.

Service name parameter: `service=@exportToXML`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.
`xpath`	Dataset table to export (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

82.5 **Export data to a CSV file**

Workflows consider the exportToCSV service as complete when export is done and file downloaded.

Service name parameter: `service=@exportToCSV`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.
`xpath`	Dataset table to export (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

82.6 **Import data from an XML file**

Workflows consider the importFromXML service as complete when import is performed.

Service name parameter: `service=@importFromXML`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Dataset table to import (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

82.7 **Import data from a CSV file**

Workflows consider the importFromCSV service as complete when import is performed.

Service name parameter: `service=@importFromCSV`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Dataset table to import (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

82.8 **Access a dataspace**

A workflow automatically considers that the dataspace selection service is complete.

Service name parameter: `service=@selectDataSpace`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.

82.9 **Validate a dataspace, a snapshot or a dataset**

Workflows automatically consider the validation service as complete.

Service name parameter: `service=@validation`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace or snapshot is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace or snapshot is required for this service.

Output parameters

Parameter	Label	Description
`hasError`	Found errors	Contains 'true' if validation has produced errors.
`hasFatal`	Found fatal errors	Contains 'true' if validation has produced fatal errors.
`hasInfo`	Found informations	Contains 'true' if validation has produced informations.
`hasWarning`	Found warnings	Contains 'true' if validation has produced warnings.

82.10 **Merge a dataspace**

Workflows consider the merge service as complete when merger is performed and dataspace is closed.

Service name parameter: `service=@merge`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.

Output parameters

Parameter	Label	Description
`mergeResult`	Merge success	Contains 'true' if merge succeeded, otherwise 'false'.
`mergeState`	Merge state	Contains the return code of the merge. It is strongly recommended to parse this value by using the InteractionMergeState UIHttpManagerComponentReturnCode.

82.11 Access the dataspace merge view

The merge.view service is automatically considered complete.

Service name parameter: `service=@merge.view`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.

82.12 Compare contents

Workflows automatically consider the compare service as complete.

Service name parameter: `service=@compare`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`compare.branch`	Dataspace to compare	The identifier of the dataspace to compare - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`compare.filter`	Comparison filter	To ignore inheritance and function fields in the comparison (disable resolved mode), the filter "persistedValuesOnly" must be specified. By default, when no filter is defined, the comparison uses resolved mode.
`compare.instance`	Dataset to compare	The value must be the reference of a dataset that exists in the selected dataspace to compare.
`compare.version`	Snapshot to compare	The identifier of the snapshot to compare - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`compare.xpath`	Table or record to compare (XPath)	The value must be a valid absolute location path of a table or a record in the selected dataset to compare. The notation must conform to a simplified XPath, in its abbreviated syntax.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`xpath`	Table or record (XPath)	The value must be a valid absolute location path of a table or a record in the selected dataset. The notation must

Parameter	Label	Description
		conform to a simplified XPath, in its abbreviated syntax.

82.13 Data workflows

This service provides access to the data workflows user interfaces.

Service name parameter: `service=@workflow`

> **Note**
>
> This service is for perspectives only.

Input parameters

Parameter	Label	Description
`scope`	Scope	Defines the scope of the user navigation for this service.
`viewPublication`	View publication	Defines the publication name of the view to apply for this service.
`workflowView`	Workflow view	Specifies the workflow view type. Value can be one of the following: "inbox", "launcher", "monitoringPublications", "monitoringWorkflows", "monitoringWorkItems" or "completedWorkflows".
`xpath`	Filter (XPath)	An optional filter. The syntax should conform to an XPath predicate surrounded by "[" and "]".

CHAPTER 83

Supported XPath syntax

This chapter contains the following topics:

1. Overview
2. Example expressions
3. Syntax specifications for XPath expressions
4. Java API

83.1 Overview

The XPath notation used in TIBCO EBX® must conform to the *abbreviated syntax* of the XML Path Language (XPath) Version 1.0 standard, with certain restrictions. This document details the abbreviated syntax that is supported.

83.2 Example expressions

The general XPath expression is:

```
path[predicate]
```

Absolute path

```
/library/books/
```

Relative paths

```
./Author
../Title
```

Root and descendant paths

```
//books
```

Table paths with predicates

```
../../books/[author_id = 0101 and (publisher = 'harmattan')]
/library/books/[not(publisher = 'dumesnil')]
```

Complex predicates

```
starts-with(col3,'xxx') and ends-with(col3,'yyy') and osd:is-not-null(./col3))
```

```
contains(col3,'xxx') and ( not(col1=100) and date-greater-than(col2,'2007-12-30') )
```

Predicates with parameters

```
author_id = $param1 and publisher = $param2 where the parameters $param1 and $param2 refer respectively to 0101 and 'harmattan'
```

```
col1 < $param1 and col4 = $param2  where the parameters $param1 and $param2 refer respectively to 100 and 'true'
```

```
contains(col3,$param1) and date-greater-than(col2,$param2) where the parameters $param1 and $param2 refer respectively to 'xxx' and '2007-12-30'
```

> **Note**
>
> The use of this notation is restricted to the Java API since the parameter values can only be set by the method `Request.setXPathParameter`[API] of the Java API.

Search predicate

- Syntax: `osd:search(fields, queryString[, templateKey])`
- Examples:
  - `osd:search('col1', 'xxx')`
  - `osd:search('col1,col2','xxx')`
  - `osd:search('','xxx')`
  - `osd:search(col1,'xxx', myTemplate@myModule)`

The `osd:search` function tries to match a term, or a list of terms, against the set of fields of the current table. This function is generic, handling every field datatype supported by EBX®. When no fields are specified, it searches against: all fields (in case of history table), or all optimized fields (in all other cases). See Limitations [p 512] for more information about optimized / unoptimized fields.

Optimized text search filters cannot be OR-combined with non-optimized filters or filters that involve a different table. See Technical limitation [p 133] for more information.

For any concerned field, if a label exists, the search targets the label, rather than the value; however, this is not yet supported in some cases. See Value-labeling [p 511] for more information.

For a more advanced usage, the query string supports specialized operators, see Special characters [p 129] for more information.

See also *Quick Search [p 128]*

The predicate `osd:search` is localized.

> **Note**
>
> The locale can be set by the methods of the Java API `Request.setLocale`[API] or `Request.setSession`[API].

> **Note**
>
> The identifier of a **search template** `SearchTemplate`[API] can be specified, to customize the behavior of the search.

Predicates for validation search

```
osd:has-validation-item()
```

```
osd:has-validation-item('error,info')
```

```
osd:contains-validation-message('xxx')
osd:contains-validation-message('xxx','info,warning')
```

- XPath functions for validation search cannot be used on XPath predicates defined on associations and foreign key filters.
- The predicates `osd:label`, `osd:contains-record-label` and `osd:contains-validation-message` are localized.

> **Note**
>
> The locale can be set by the methods of the Java API `Request.setLocale`[API] or `Request.setSession`[API].

Attention

To ensure that the search is performed on an up-to-date validation report, it is necessary to perform an explicit validation of the table just before using these predicates.

83.3 Syntax specifications for XPath expressions

Overview

Expression	Format	Example
XPath expression	<*container path*>[*predicate*]	`/books[title='xxx']`
<*container path*>	<*absolute path*> or <*relative path*>	
<*absolute path*>	`/a/b` or `//b`	`//books`
<*relative path*>	`../../b`, `./b` or `b`	`../../books`

Predicate specification

Expression	Format	Notes/Example
<predicate>	Example: A and (B or not(C)) A,B,C: *<atomic expression>*	Composition of: logical operators parentheses, not() and atomic expressions.

<atomic expression>	*<path><comparator><criterion>* or method(*<path>*,*<criterion>*)	`royalty = 24.5`
		`starts-with(title, 'Johnat')`
		`booleanValue = true`

<path>	*<relative path> or osd:label(<relative path>)*	Relative to the table that contains it: `../authorstitle`
<comparator>	• *<boolean comparator>* • *<numeric comparator>* or • *<string comparator>*	
<boolean comparator>	`=` or `!=`	
<numeric comparator>	`=` , `!=` , `<`, `>`, `<=`, or `>=`	
<string comparator>	`=` or `!=`	
<method>	• *<date method>* • *<string method>* • `osd:is-null` method or • `osd:is-not-null` method	
<date, time & dateTime method>	• `date-less-than` • `date-equal` or • `date-greater-than`	
<string method>	• `contains` • `osd:contains-case-insensitive` • `osd:contains-record-label` • `osd:is-equal-case-insensitive` • `osd:is-empty` • `osd:is-empty-or-nil` • `osd:is-neither-empty-nor-nil` • `osd:is-not-empty` • `ends-with` • `osd:ends-with-case-insensitive` • `matches` • `osd:search`	

Expression	Format	Notes/Example
	• `starts-with` or • `osd:starts-with-case-insensitive`	
<*criterion*>	• <*boolean criterion*> • <*date criterion*> • <*dateTime criterion*> • <*numeric criterion*> • <*string criterion*> or • <*time criterion*>	
<*boolean criterion*>	`true` , `false`	
<*numeric criterion*>	An integer or a decimal	`-4.6`
<*string criterion*>	Quoted character string	`'azerty'`
<*date criterion*>	Quoted and formatted as 'yyyy-MM-dd'	`'2007-12-31'`
<*time criterion*>	Quoted and formatted as 'HH:mm:ss' or 'HH:mm:ss.SSS'	`'11:55:00'`
<*dateTime criterion*>	Quoted and formatted as 'yyyy-MM-ddTHH:mm:ss' or 'yyyy-MM-ddTHH:mm:ss.SSS'	`'2007-12-31T11:55:00'`

XPath 1.0 formula

It is possible to use an XPath 1.0 formula in the criterion value part of an atomic predicate expression (right-hand side).

For example, instead of `[./a=3]`, you may use the expression `[./a=(floor(./d)+ 2.0)]`.

Due to the strong dependence of predicates on the data model node and the node type of the criterion, the path portion of the atomic predicate expression (left-hand side) must be a node path and cannot be an XPath formula. For example, the expression `/table[floor(./a) > ceiling(./d)]` is not valid.

Predicate on label

The `osd:label()` function can be applied to the path portion of the atomic predicate, in order to resolve the predicate on the label instead of the value. In this case, only string operators and string criteria can be used, i.e. `ends-with(osd:label(./price),'99')`.

A predicate on label is localized, so the criterion must be expressed in the same locale as the predicate-filtered request. For example: `request.setLocale(Locale.FRENCH); request.setXPathFilter("osd:label(./delivery_date)='30/12/2014'");`

> **Note**
>
> It is forbidden to use the `osd:label` function if the right part of the predicate is a contextual value.

> **Note**
>
> If the `osd:label` function is used in a data model, for example on a selection or in the filter predicate of a table reference node, the default locale of the data model (as defined in its module declaration) must be used for the criterion format (even though this is generally not recommended).

See also *`SchemaNode.displayOccurrence`*[API]

Contextual values

For predicates that are relative to a selected node, the criterion value (that is, the right-hand side of the predicate) can be replaced with a contextual path using the syntax `${<relative-path>}` where *<relative-path>* is the location of the element relative to the selected node. The criterion value must be a single value. That means that aggregated lists are not allowed as the target of the contextual path.

> **Note**
>
> When calling a method, the criterion is the second parameter, and the first parameter cannot be a relative value.

Aggregated lists

For predicates on aggregated lists, the predicate returns `true` regardless of the comparator if one of the list elements verifies the predicate.

In addition, the predicates `osd:is-empty` and `osd:is-not-empty` can be applied on any node which type is a list to very if the list is empty or not.

> **Note**
>
> Special attention must be paid to the comparator `!=`. For example, for an aggregated list, `./list != 'a'` is not the same as `not(./list = 'a')`. Where the list contains the elements `(e1,e2,..)`, the first predicate is equivalent to `e1 != 'a' or e2 != 'a' ...`, while the second is equivalent to `e1 != 'a' and e2 != 'a' ...`.

'Null' values

Null values must be explicitly treated in a predicate using the operators `osd:is-null` and `osd:is-not-null`.

For example, `/root/products[./price<100]` or `/root/products[./price!=100]` will not return any products whose prices are not set (`null`). For the latter case to return unset values as well, the predicate must instead be: `/root/products[./price!=100 or osd:is-null(./price)]`.

How to manage single and double quotes in literal expressions

By default, a literal expression is delimited by single quotes (`'`). If the literal expression contains single quotes and no double quotes, the expression must be delimited by double quotes (`"`). If the literal expression contains both single and double quotes, the single quotes must be doubled.

The method `XPathExpressionHelper.encodeLiteralStringWithDelimiters`[API] in the Java API handles this.

Examples of using `encodeLiteralStringWithDelimiters`

Value of Literal Expression	**Result of this method**
`Coeur`	`'Coeur'`
`Coeur d'Alene`	`"Coeur d'Alene"`
`He said: "They live in Coeur d'Alene".`	`'He said: "They live in Coeur d''Alene".'`

Extraction of foreign keys

In EBX®, the foreign keys are grouped into a single field with the `osd:tableRef` [p 919] declaration.

The standard XPath syntax has been extended so as to extract the value of any targeted primary key field.

Example

If the table `/root/tableA` has an `osd:tableRef` field named 'fkB' whose target is `/root/tableB` and the primary key of `tableB` has two fields, `id` of type `xs:int` and `date` of type `xs:date`, then the following expressions would be valid:

- `/root/tableA[ fkB = '123|2008-01-21' ]`, where the string "123|2008-01-21" is a representation of the entire primary key value.

  See **Syntax of the internal String representation of primary keys** `PrimaryKey.syntax`[API] for more information.
- `/root/tableA[ fkB/id = 123 and date-equal(fkB/date, '2008-01-21') ]`, where this predicate is a more efficient equivalent to the one in the previous example.
- `/root/tableA[ fkB/id >= 123 ]`, where any number operator could be used, as the targeted primary key field is of type `xs:int`.
- `/root/tableA[ date-greater-than( ./fkB/date, '2007-01-01' ) ]`, where any date operator could be used, as the targeted primary key field is of type `xs:date`;
- `/root/tableA[ fkB = "" ]` is not valid as the targeted primary key has two columns.
- `/root/tableA[ osd:is-null(fkB) ]` checks if a foreign key is `null` (not defined).

83.4 Java API

Using the XPath in the Java API:

In the Java API, the `XPathFilter` class allows to define XPath predicates and to execute requests on them.

The `XPathExpressionHelper` class provides utilitarian methods to handle XPath predicates and paths.

Localization

CHAPTER 84

Labeling and localization

This chapter contains the following topics:

1. Overview
2. Value formatting policies
3. Syntax for locales

84.1 Overview

TIBCO EBX® offers the ability to handle the labeling and the internationalization of data models.

Localizing user interactions

In EBX®, language preferences can be set for two scopes:

1. Session: Each user can select a default locale from the user pane.
2. Data model: If a data model has been localized into other languages than those natively supported by EBX®, the user can select one of those languages for that particular data model. See Extending TIBCO EBX® internationalization [p 453] for more information.

Textual information

In EBX®, most master data entities can have a label and a description, or can correspond to a user message. For example:

- Dataspaces, snapshots and datasets can have their own label and description. The label is independent of the unique name, so that it remains localizable and modifiable;
- Any node in the data model can have a static label and description;
- Values can have a static label when they are enumerated;
- Validation messages can be customized, and permission restrictions can provide text explaining the reason;
- Each record is dynamically displayed according to its content, as well as the context in which it is being displayed (in a hierarchy, as a foreign key, etc.);

All this textual information can be localized into the locales that are declared by the module.

See also

Labels and messages [p 955]

Tables declaration *[p 913]*

Foreign keys declaration *[p 919]*

84.2 Value formatting policies

When a value is displayed to the user, it is formatted according to its type and the formatting policy of the current locale. For example, a date will be displayed in some locales as "dd/MM/yyyy" and "MM/dd/yyyy" in others.

A formatting policy is used to define how to display the values of simple types [p 900].

For each locale declared by the module, its formatting policy is configured in a file located at `/WEB-INF/ebx/{locale}/frontEndFormattingPolicy.xml`. For instance, to define the formatting policy for Greek (`el`), the engine looks for the following path in the module:

```
/WEB-INF/ebx/el/frontEndFormattingPolicy.xml
```

If the corresponding file does not exist in the module, the formatting policy is looked up in the class-path of EBX®. If the locale-specific formatting policy is not found, the formatting policy of `en_US` is applied.

The content of the file `frontEndFormattingPolicy.xml` is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<formattingPolicy  xmlns="urn:ebx-schemas:formattingPolicy_1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ebx-schemas:formattingPolicy_1.0 ../schema/ebx-reserved/formattingPolicy_1.0.xsd">
 <date pattern="dd/MM" />
 <time pattern="HH:mm:ss" />
 <dateTime pattern="dd/MM/yyyy HH:mm" />
 <decimal pattern="00,00,00.000" groupingSeparator="|" decimalSeparator="^"/>
 <int pattern="000,000" groupingSeparator=" "/>
</formattingPolicy>
```

The elements `date`, `dateTime` and `time` are mandatory.

The group and decimal separators that appear in the formatted numbers can be modified by defining the attributes `groupingSeparator` and `decimalSeparator` for the elements `decimal` and `int`.

84.3 Syntax for locales

There are two ways to express a locale:

1. The XML recommendation follows the RFC 5646 - The Language Tag recommendation, which uses a hyphen '-' as the separator.
2. The Java specification uses an underscore '_' instead of a hyphen.

In any XML file (XSD, formatting policy file, etc.) read by EBX®, either syntax is allowed.

For a web path, that is, a path within the web application, only the Java syntax is allowed. Thus, formatting policy files must be located in directories whose locale names respect the Java syntax.

See also *Extending TIBCO EBX® internationalization* *[p 453]*

CHAPTER 85

Extending TIBCO EBX® internationalization

This chapter contains the following topics:

1. Overview of the native EBX® localization
2. Language packs software
3. Extending EBX® user interface localization
4. Localized resources resolution
5. Known limitations

85.1 Overview of the native EBX® localization

By default, the EBX® built-in user interface is provided in English (en-US) and French (fr-FR).

Localization consists of a formatting policy and a set of message files (resource bundle):

- For English, localization is provided by a formatting policy and a set of message files with no locale defined,
- For French, localization is provided by a formatting policy and a set of message files with locale set to "fr".

EBX® provides an option to add locales in order to extend the localization of the user interface and to internationalize the documentation of data models and associated services.

85.2 Language packs software

Language packs for TIBCO EBX® and its addons are available from TIBCO eDelivery according to the EBX® release. A language pack can also contain the online help, please see the `TIB_ebx-lp_<version>_readme.txt` file for more information.

> **Note**
>
> On each release, there is a certain delay between the GA and the availability of the corresponding pack; in this case, the previous pack can be deployed (but may not cover all messages).

85.3 Extending EBX® user interface localization

EBX® supports the localization of its user interface into any compatible language and region.

> **Note**
>
> Currently, Latin & Cyrillic characters are supported. Locales that use other character sets may be usable, but are not supported.

Adding a new locale

In order to add a new locale, the following steps must be followed:

- Declare the new locale in the EBX® main configuration file. For example:

  ebx.locales.available=en-US, fr-FR, xx

  - The first locale is always considered the default.
  - The built-in locales, en-US and fr-FR, can be removed if required.

  See Configuring EBX® localization [p 581].
- Package the following files in a jar file (sample name: `ebx-lp_<version>_languagepack-xx.jar`):
  - A formatting policy file, named `com.orchestranetworks.i18n.frontEndFormattingPolicy_xx.xml`,
  - A set of localized message files (*_xx.mxml) in a resource bundle.

    > **Note**
    >
    > The files must be ending with ".mxml".

- Deploy the jar file in the EBX® class-path it must be accessible from the `ebx.jar` class-loader.

85.4 Localized resources resolution

Since version 5.7.0, localized resources are resolved on a locale-proximity base, with the following lookup mechanism:

- resourceName + "_" + language + "_" + country + "_" + variant + ".mxml"
- resourceName + "_" + language + "_" + country + ".mxml"
- resourceName + "_" + language + ".mxml"
- resourceName + ".mxml"

> **Note**
>
> The resolution is done at the localized message level. It is therefore possible to define one or more files for a locale that only includes messages for which specific localization is required.

85.5 Known limitations

Non extendable materials

Localization of the following cannot be extended:

- EBX® HTML editor and viewer.

Persistence

CHAPTER 86

Overview of persistence

This chapter is an introduction to history tables and replicated tables.

> **Note**
>
> The term *mapped mode* [p 459] refers to any tables that are stored as-is, and thus whose contents can be accessed directly in the database.

This chapter contains the following topics:

1. Primary persistence of managed master data
2. Historization
3. Replication
4. Mapped mode

86.1 Primary persistence of managed master data

Data that is modeled in and governed by the EBX® repository are primarily persisted in the relational database, using generic tables (common to all datasets and data models).

86.2 Historization

Master data tables can activate historization in order to track modifications to their data, regardless of whether they are replicated.

The history itself is in mapped mode, meaning that it can be consulted directly in the underlying database.

See also *History* *[p 461]*

86.3 Replication

Replication enables direct SQL access to tables of master data, by making a copy of data in the repository to replica tables in the relational database. Replication can be enabled on any table regardless of whether it has history activated.

The replica tables are persisted in mapped mode, as their primary purpose is to make master data accessible to direct queries outside of EBX®.

See also *Replication [p 469]*

86.4 Mapped mode

Overview of mapped mode

Mapped mode refers to cases where tables are persisted in the underlying relational database in a format that allows their data to be accessed directly, outside of EBX®. History tables and replica tables are all examples of tables in mapped mode.

All cases of mapped mode involve automatic alterations of the database schema (the database tables, indexes, etc.) when necessary, by automatically executing required DDL statements in the background. Such procedures are always triggered at data model compilation time and the data model compilation report notifies of any resulting errors.

Another general consideration regarding mapped modes is that, in most cases, when a data model entity is removed, its corresponding database object is not deleted immediately. Instead, it is marked as disabled, which leaves the possibility of later re-enabling the object. In order to definitively drop the object and its associated data and resources from the database, it must be marked for purge. The removal then takes place during the next global purge.

See also

Database mapping administration [p 711]

Data model evolutions [p 475]

Structural constraints

When a mapped mode is set, some EBX® data model constraints will generate a "structural constraint" on the underlying RDBMS schema. This concerns the following constraining facets:

- facets `xs:maxLength` and `xs:length` on `string` elements;
- facets `xs:totalDigits` and `xs:fractionDigits` on `xs:decimal` elements.

Databases do not support as tolerant a validation mode as EBX®. Hence, the above constraints become *blocking constraints*. A blocking constraint means that updates are rejected if they do not comply: when a transaction does not comply with a blocking constraint, it is cancelled and a `ConstraintViolationException`[API] is thrown.

See also *Blocking and non-blocking constraints [p 945]*

Data model restrictions due to mapped mode

Due to the nature of persisting directly in the underlying database, some restrictions apply to all tables stored in mapped mode:

- Limitations of supported databases [p 527]
- Unlimited-length strings: All string fields, except foreign keys, of type `xs:string`, its derived types, and `xs:anyURI` must define a 'maxLength' or 'length' facet. Since a foreign key field is composed of the final primary key field(s) of its target table(s), this facet requirement applies to each of those final primary key fields instead of the foreign key field itself. Additionally, limitations of the underlying database concerning the maximum length of its character types apply, such as *VARCHAR* and *NVARCHAR2*.

- Fields of type `type="osd:password"` are ignored.
- Terminal complex types are supported; however, they cannot be globally set to `null` at record-level.

More generally, tables in mapped mode are subject to any limitations of the underlying RDBMS. For example, the maximum number of columns in a table applies (1000 for Oracle, 1600 for PostgreSQL). Note that a history table contains twice as many fields as declared in the schema (one functional field, plus one generated field for the operation code).

Data model evolutions may also be constrained by the underlying RDBMS, depending on the existing data model.

See also *Data model evolutions* *[p 475]*

CHAPTER 87

History

This chapter contains the following topics:

1. Overview
2. Configuring history
3. History views and permissions
4. SQL access to history
5. Impacts and limitations of historized mode

87.1 Overview

History is a feature allowing to track all data modifications on a table (records creation, update and deletion).

It is an improvement over the deprecated XML audit trail [p 736].

See also

History *[p 38]*

Replication *[p 469]*

Data model evolutions *[p 475]*

87.2 Configuring history

In order to activate historization for a table, a history profile has to be set for the table in the data model. This section describes history profiles and the way they are associated with tables.

Configuring history in the repository

A history profile specifies when the historization is to be created. In order to edit history profiles, select *Administration* > *History and logs*.

A history profile is identified by a name and defines the following information:

- An internationalized label.
- A list of dataspaces (branches) for which history is activated. It is possible to specify whether direct children and/or all descendants should also be concerned.

Some profiles are already created when installing the repository. These profiles can neither be deleted nor modified.

Profile Id	Description
`ebx-referenceBranch`	This profile is activated only on the reference dataspace.
`ebx-allBranches`	This profile is activated on all dataspaces.
`ebx-instanceHeaders`	This profile historizes dataset headers. However, this profile will only be setup in a future version, given that the internal data model only defines dataset nodes.

Configuring history in the data model

Activating table history

History can be activated on a table either through the data model assistant, or by editing the underlying data model.

To activate history by editing the data model, a history profile should be declared on the table using the `historyProfile` element.

```
<osd:table>
 <primaryKeys>/key</primaryKeys>
 <historyProfile>historyProfileForProducts</historyProfile>
</osd:table>
```

The data model assistant allows you to view the historization profiles defined in the repository.

Historization must be activated for each table separately. See model design [p 896] documentation for more details.

Disabling history on a specific field or group

For a historized table, the default behavior is to historize all its supported elements (see Impacts and limitations of historized mode [p 466]).

It is possible to disable history for a given field or group, either through the data model assistant, or by editing the underlying data model.

To disable the history of a field or group by editing the data model, use the element `osd:history` with the attribute `disable="true"`.

```
<xs:element name="longDescription" type="xs:string">
    <xs:annotation>
        <xs:appinfo>
            <osd:history disable="true" />
        </xs:appinfo>
    </xs:annotation>
</xs:element>
```

To disable the history of a field or group through the data model assistant, use the `History` property in the `Advanced properties` of the element.

When this property is defined on a group, history is disabled recursively for all its descendants. Once a group disables history, it is not possible to specifically re-enable history on a descendant.

> **Note**
>
> If the table containing the field or group is not historized, this property will not have any effect.
>
> It is not possible to disable history for primary key fields.

Integrity

If problems are detected at data model compilation, warning messages or error messages will be added to the validation report associated with this data model. Furthermore, if any error is detected, each associated instance (dataset) will be inaccessible. The most common error cases are the following:

- A table references a profile that is not defined in the repository.
- A history profile that is referenced in the data model mentions a non-defined or closed dataspace in the current repository.

> **Note**
>
> Deploying a data model on a repository that does not have the expected profiles requires the administrator to add them.

87.3 History views and permissions

Table history view

When the history has been activated on a table in the data model, it is possible to access the history view from various locations in the user interface: record, selection of records, table and dataset.

The next section explains how permissions are resolved.

For more information, see table history view [p 39] section. To access the table history view from Java, the method `AdaptationTable.getHistory`[API] must be invoked.

Permissions for table history

Data permissions are also applied to data history. History permissions are resolved automatically as the most restricted permission between data permissions and *read-only* access right.

This is true for user-defined permission rules and also for programmatic permission rules.

When defining a programmatic rule, it may be required to distinguish between the functional dataset context and the history view context, either because the expected permissions are not the same, or because some fields are not present in the history structure. This is the case for dataset fields, computed values and fields for which history has been disabled [p 462]. The methods `Adaptation.isHistory`[API] and `AdaptationTable.getHistory`[API] can then be used in the programmatic rule in order to implement specific behavior for history.

> **Note**
>
> There is currently a limitation when a table has a scripted permission rule on record specified: for security reason access to the table history is totally disabled for everyone. Access to history will be allowed in a future version.

Transaction history views

The transaction history view gives access to the executed transactions, independently of a table, a dataset or a data model, directly from the user interface.

To see the 'Transaction history' table, navigate to the Administration area and select 'History and logs' using the down arrow menu in the navigation pane. Transaction history can also be accessed from the Dataspaces area by selecting a historized dataspace and using the **Actions** menu in the workspace.

For more information, see transaction history view [p 39].

History service on the sidebar

Access a record's history view from the sidebar. This view shows a list of:

- transactions
- modified field values
- the user that modified the record
- timestamp

You can sort transactions by date and time, and filter them by field. History display supports both simple and complex data types, including multi-valued list types.

87.4 SQL access to history

This section describes how to directly access the history data by means of SQL.

Access restrictions

The database tables must be accessed only in read-only mode. It is up to the database administrator to forbid write access except for the database user used by TIBCO EBX®, as specified in the section Rules for the database access and user privileges [p 661].

Relational schema overview

Here is a description of the history tables in the database.

The database schema contains (see also the diagram in the next section):

Common and generic tables	The main table is `HV_TX`; each record of this table represents a transaction. Only transactions that involve at least one historized table are recorded. These common tables are all prefixed by "HV".
Specific generated tables	For each historized table, a specific history table is generated. This table contains the history of the data modifications on the table. In the EBX® user interface, the name of this table in database can be obtained by clicking on the table documentation pane (advanced mode). All the specific history tables are prefixed with "HG".

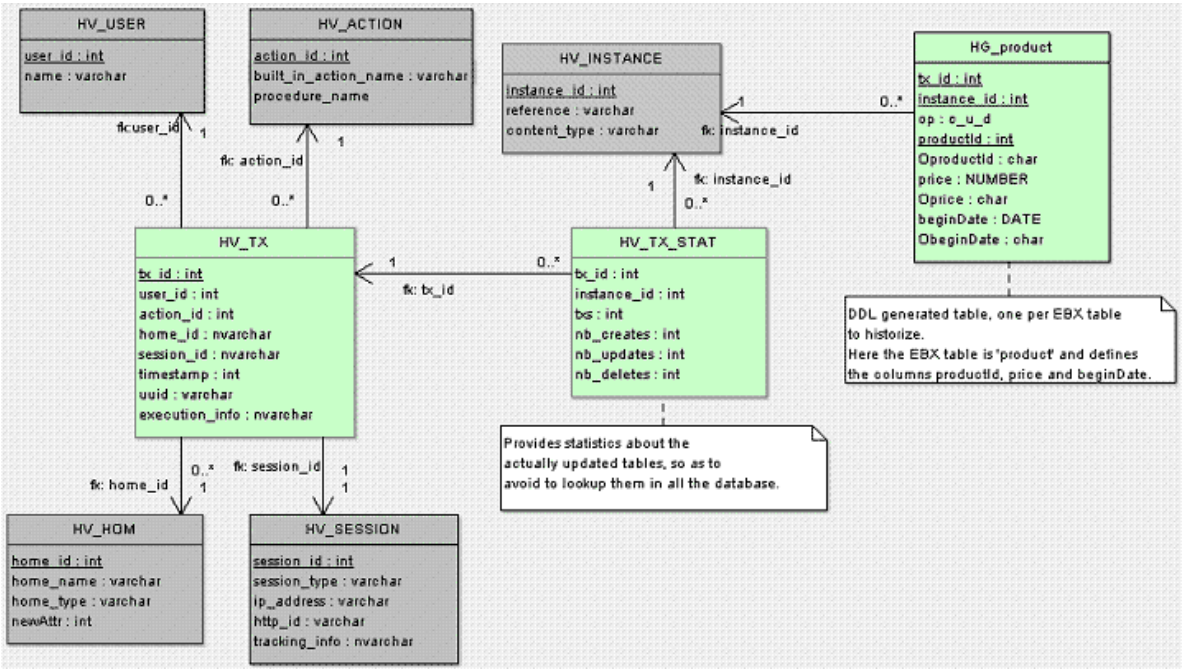
Example of a generated history table

In the following example, we are historizing a table called `product`. Let us assume this table declares three fields in EBX® data model:

Product

- productId: int
- price: int
- beginDate: Date

The diagram below shows the resulting relational schema:



Activating history on this table generates the HG_product table shown in the history schema structure above. Here is the description of its different fields:

- `tx_id`: transaction ID.
- `instance`: instance ID.
- `op`: operation type - C (create), U (update) or D (delete).
- `productId`: `productId` field value.
- `OproductId`: operation field for `productId`, see next section.
- `price`: `price` field value.
- `Oprice`: operation field for `price`, see next section.
- `beginDate`: `date` field value.
- `ObeginDate`: operation field for `beginDate`, see next section.

Combination of operations

If several operations are combined in the same transaction, the operation field is resolved as follows:

- `C + U -> C`

  `D + U -> D`

  `D + C -> U`

  `C + D -> {} (no entry in history)`

Values for operation fields

For each functional field, an additional operation field is defined, composed of the field name prefixed by the character `O`. This field specifies whether the functional field has been modified. It is set to one of the following values:

- `null`: if the functional field value has not been modified (and its value is not INHERIT).
- `M`: if the functional field value has been modified (not to INHERIT).
- `D`: if record has been deleted.

If inheritance [p 480] is enabled, the operation field can have three additional values:

- `T`: if the functional field value has not been modified and its value is INHERIT.
- `I`: if the functional field value has been set to INHERIT.
- `O`: if the record has been set to OCCULTING mode.

87.5 Impacts and limitations of historized mode

The history feature has some impacts and known limitations, which are listed in this section. If using historized mode, it is strongly recommended to read these limitations carefully and to contact Cloud Software Group, Inc. support in case of questions.

Validation

Some EBX® data model constraints become blocking constraints when table history is activated. For more information, see the section Structural constraints [p 459].

Data model restrictions for historized tables

Some restrictions apply to data models containing historized tables:

- Data model restrictions due to mapped mode [p 459]
- Limitations exist for two types of aggregated lists: aggregated lists under another aggregated list, and aggregated lists under a terminal group. Data models that contain such aggregated lists can be used, however these lists will be ignored (not historized).
- Computed values are ignored.
- Linked fields are ignored.
- User-defined attributes on historized tables result in data model compilation errors.

Data model evolutions may also be constrained by the underlying RDBMS, depending on the data already contained in the concerned tables.

See also *Data model evolutions [p 475]*

Other limitations of historized mode

- No data copy is performed when a table with existing data is activated for history.
- Global operations on datasets are not historized (create an instance and remove an instance), even if they declare a historized table.
- Default labels referencing a non-historized field are not supported for historized tables.

  As a consequence, default labels referencing a computed field are not supported for historized tables.

  The workaround is to implement the `UILabelRenderer` interface and adapt the label computation for history.
- D3: the history can be enabled in the delivery dataspace of a primary node, but in the delivery dataspace of the replica nodes, the historization features are always disabled.
- Recorded user in history: for some specific operations, the user who performs the last operation and the one recorded in the corresponding history record may be different.

  This is due to the fact that these operations are actually a report of the data status at a previous state:

  - Archive import: when importing an archive on a dataspace, the time and user of the last operation performed in the child dataspace are preserved, while the user recorded in history is the user who performs the import.
  - Programmatic merge: when performing a programmatic merge on a dataspace, the time and user of the last operation performed in the child dataspace are preserved, while the user recorded in history is the user who performs the merge.
  - D3: for distributed data delivery feature, when a broadcast is performed, the data from the primary node is reported on the replica node and the time and user of the last operation performed in the child dataspace are preserved, while the user recorded in history is 'ebx-systemUser' who performs the report on the replica node upon the broadcast.

CHAPTER 88

Replication

This chapter contains the following topics:

1. Overview
2. Configuring replication
3. Accessing a replica table using SQL
4. Requesting an 'onDemand' replication refresh
5. Impact and limitations of replication

88.1 Overview

Data stored in the TIBCO EBX® repository can be mirrored to dedicated relational tables to enable direct access to the data by SQL requests and views.

Like history, this data replication is transparent to end-users and client applications. Certain actions trigger automatic changes to the replica in the database:

- Activating replication at the model-level updates the database schema by automatically executing the necessary DDL statements.
- Data model evolutions that impact replicated tables, such as creating a new column, also automatically update the database schema using DDL statements.
- When using the 'onCommit' refresh mode: updating data in the EBX® repository triggers the associated inserts, updates, and deletions on the replica database tables.

See also

History *[p 461]*

Data model evolutions *[p 475]*

Repository administration *[p 660]*

> **Note**
>
> *replicated table*: refers to a primary data table that has been replicated
>
> *replica table* (or *replica*): refers to a database table that is the target of the replication

88.2 Configuring replication

Enabling replication

To define a *replication unit* on a data model, use the element `osd:replication` under the elements `annotation/appinfo`. Each replication unit specifies tables in a single dataset in a specific dataspace.

The nested elements are as follows:

Element	Description	Required
`name`	Name of the replication unit. This name identifies a replication unit in the current data model. It must be unique.	Yes
`dataSpace`	Specifies the dataspace relevant to this replication unit. It cannot be a snapshot.	Yes
`dataSet`	Specifies the dataset relevant to this replication unit.	Yes
`refresh`	Specifies the data synchronization policy. The possible policies are: • `onCommit`: The replica table content in the database is always up to date with respect to its source table. Every transaction that updates the EBX® source table triggers the corresponding insert, update, and delete statements on the replica table. • `onDemand`: The replication of specified tables is only done when an explicit refresh operation is performed. See Requesting an 'onDemand' replication refresh [p 472].	Yes
`table/path`	Specifies the path of the table in the current data model that is to be replicated to the database.	Yes
`table/nameInDatabase`	Specifies the name of the table in the database to which the data will be replicated. This name must be unique amongst all replications units.	Yes
`table/element/path`	Specifies the path of the aggregated list in the table that is to be replicated to the database.	Yes
`table/element/nameInDatabase`	Specifies the name of the table in the database to which the data of the aggregated list will be replicated. This name must be unique amongst all replications units.	Yes

For example:

```
<xs:schema>
 <xs:annotation>
  <xs:appinfo>
   <osd:replication>
    <name>ProductRef</name>
    <dataSpace>ProductReference</dataSpace>
    <dataSet>productCatalog</dataSet>
    <refresh>onCommit</refresh>
    <table>
     <path>/root/domain1/tableA</path>
     <nameInDatabase>PRODUCT_REF_A</nameInDatabase>
    </table>
```

```
    <table>
     <path>/root/domain1/tableB</path>
     <nameInDatabase>PRODUCT_REF_B</nameInDatabase>
     <element>
      <path>/retailers</path>
      <nameInDatabase>PRODUCT_REF_B_RETAILERS</nameInDatabase>
     </element>
    </table>
   </osd:replication>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:schema>
```

Notes:

- See Data model restrictions for replicated tables [p 473]
- If, at data model compilation, the specified dataset and/or dataspace does not exist in the current repository, a warning is reported, but the replica table is created in the database. Once the specified dataspace and dataset are created, the replication becomes active.
- At data model compilation, if a table replication is removed, or if some of the above properties has changed, the replica table is dropped from the database, and then recreated with the new definition if needed.

Disabling replication on a specific field or group

For a replicated table, the default behavior is to replicate all its supported elements (see Data model restrictions for replicated tables [p 473]).

It is possible to disable replication for a specific field or group, either through the data model assistant, or by editing the underlying data model.

To disable the replication of a field or group by editing the data model, use the element `osd:replication` with the attribute `disable="true"`.

```
<xs:element name="longDescription" type="xs:string">
    <xs:annotation>
        <xs:appinfo>
            <osd:replication disable="true" />
        </xs:appinfo>
    </xs:annotation>
</xs:element>
```

To disable the replication of a field or group through the data model assistant, use the `Replication` property in the `Advanced properties` of the element.

When this property is defined on a group, replication is disabled recursively for all its descendents. Once a group disables replication, it is not possible to specifically re-enable replication on a descendant.

> **Note**
>
> If the table containing the field or group is not replicated, this property will not have any effect.
>
> It is not possible to disable replication for primary key fields.

88.3 Accessing a replica table using SQL

This section describes how to directly access a replica table using SQL.

See also *SQL access to history [p 464]*

Finding the replica table in the database

For every replicated EBX® table, a corresponding table is generated in the RDBMS. Using the EBX® user interface, you can find the name of this database table by clicking on the documentation pane of the table. The table in the RDBMS contains the following additional technical columns:

- `T_CREATOR_ID`, reference of the user that has created the record
- `T_LAST_USER_ID`, reference of the last user that has updated the record
- `T_CREATION_DATE`, creation date of the record
- `T_LAST_WRITE`, time of last modification of the record

Columns `T_CREATOR_ID` and `T_LAST_USER_ID` have a maximum length of 320 characters. As a consequence, identifiers of users defined in EBX® directory must have a maximum length less than this limit.

Access restrictions

The replica database tables must only be directly accessed in read-only mode. It is the responsibility of the database administrator to block write-access to all database users except the one that EBX® uses.

See also *Rules for the database access and user privileges* *[p 661]*

SQL reads

Direct SQL reads are possible in well-managed, preferably short-lived transactions. However, for such accesses, EBX® permissions are not taken into account. As a result, applications given the privilege to perform reads must be trusted through other authentication processes and permissions.

88.4 Requesting an 'onDemand' replication refresh

The 'onDemand' refresh policy requires an explicit request to refresh the replicated table data.

There are several ways to request a replication refresh:

- **User interface**: In the dataset actions menu, use the action 'Refresh replicas' under the group 'Replication' to launch the replication refresh wizard.
- **Data services**: Use the replication refresh data services operation. See Replication refresh [p 1106] for data services for more information.
- **Java API**: Call the `ReplicationUnit.performRefresh`[API] methods in the `ReplicationUnit` API to launch a refresh of the replication unit.

88.5 Impact and limitations of replication

The replication feature has some known limitations and side-effects, which are listed below. If using replication, it is strongly recommended to read this section carefully and to contact Cloud Software Group, Inc. support in case of questions.

See Supported databases [p 527] for the databases for which replication is supported.

Validation

Some EBX® data model constraints become blocking constraints when replication is enabled. For more information, see Structural constraints [p 459].

Data model restrictions for replicated tables

Some restrictions apply to data models containing tables that are replicated:

- Data model restrictions due to mapped mode [p 459]
- Dataset inheritance is not supported for the 'onCommit' refresh policy if the specified dataset is not a root dataset or has not yet been created. See dataset inheritance [p 481] for more information.
- Field inheritance is also only supported for the 'onDemand' refresh policy. This means that, at data model compilation, an error is reported if the refresh mode is 'onCommit' and the table to be replicated has an inherited field. See inherited fields [p 482] for more information.
- Computed values are ignored.
- Linked fields are ignored.
- Limitations exist for two types of aggregated lists: aggregated lists under another aggregated list, and aggregated lists under a terminal group. Data models that contain such aggregated lists can be used, however these lists will be ignored (not replicated).
- User-defined attributes are not supported. A compilation error is raised if they are included in a replication unit.

Data model evolutions may also be constrained by the underlying RDBMS, depending on the data already contained in the concerned tables.

See also *Data model evolutions [p 475]*

Database configuration

The refresh operation is optimized to transmit only the rows of the source table that have been modified (with respect to creation and deletion) since the last refresh. However, depending on the volume of data exchanged, this can be an intensive operation, requiring large transactions. In particular, the first refresh operation can concern a large number of rows. It is necessary for the database to be configured properly to allow such transactions to run under optimal conditions.

For instance, with Oracle:

- It is mandatory for the bulk of all replica tables in a replication unit to fit into the 'UNDO' tablespace.
- It is recommended to provide enough space in the buffer cache to allow those transactions to run with minimal disk access.
- It is recommended to provision 'REDO' log groups big enough to avoid those transactions to wait for the 'db_writer' process.

Distributed data delivery (D3)

Replication is available on both D3 primary and replica delivery dataspaces. On the primary dataspace, the replication behavior is the same as in a standard semantic dataspace, but on replica dataspaces, the replicated content is that of the last broadcast snapshot.

In a replica delivery dataspace, some restrictions occur:

- The refresh policy defined in the data model has no influence on the behavior described above: replication always happens on snapshot.
- The action item `Refresh replicas` is not available.
- It is not allowed to invoke the `ReplicationUnit.performRefresh`[API] method.

See also *D3 overview [p 750]*

Other limitations of replication

- Limitations of supported databases [p 527]
- For inheritance, a replica record field cannot hold the "inherit value" flag (`AdaptationValue.INHERIT_VALUE`). It only holds the inherited value in such cases. More generally, it is not possible to distinguish inheriting state from overwriting state.

CHAPTER 89

Data model evolutions

This chapter describes the modifications that are possible on data models, as well as potential limitations.

Attention

Whenever the data modeler performs an evolution on the primary key of a table, the resulting definition is considered as a new table. In such cases, if existing data must be preserved in some ways, a data migration plan must be set up and operated before the new data model is published or deployed. It can also be noted that data is not destroyed immediately after the data model evolution; if the data model is rolled back to its previous state, then the previous data is retrieved.

Note

Certain types of data model evolutions cannot be performed directly in the user interface, and thus the data model must be exported, modified in XSD format, then re-imported. For changes to a data model that impact its configuration, not just its structure, the XSD must be imported into TIBCO EBX® from a module. Otherwise, the configuration modifications are not taken into account.

See also *Mapped mode* *[p 459]*

This chapter contains the following topics:

1. Types of permitted evolutions
2. Limitations/restrictions

89.1 Types of permitted evolutions

This section describes the possible modifications to data models after their creation.

Model-level evolutions

The following modifications can be made to existing data models:

- Replication units can be added to the data model. If their refresh policy is 'onCommit', the corresponding replica tables will be created and refreshed on next schema compilation.
- Replication units can be removed from the data model. The corresponding replica tables will be dropped immediately.

- The data model can be deleted. If it declares replication units, the corresponding replica tables will be dropped immediately. If it contains historized tables, this change marks the associated mapped tables as disabled. See Database mapping [p 711] for the actual removal of associated database objects.

Table-level evolutions

The following modifications can be made to a data model at the table-level:

- A new table can be added. Upon creation, the table can also declare one or more mapped modes.
- An existing table can be deleted. If it declares replication units, the corresponding replica tables will be dropped immediately. If it historized, this change marks the mapped table as disabled. See Database mapping [p 711] for the actual removal of associated database objects.
- History can be enabled or disabled on a table. History will not take into account the operations performed while it is disabled.
- A table can be renamed. Data should be manually migrated, by exporting then re-importing an XML or archive file, because this change is considered to be a combination of deletion and creation.

Field-level evolutions

The following modifications can be made to a data model at the field-level:

- A new field can be added.
- An existing field can be deleted. The data of the deleted field will be removed from each record upon its next update. For a replica table, the corresponding column is automatically removed. In history mode, the field is marked as disabled.
- A field can be specifically disabled from the history or replication which applies to its containing table, by using the attribute `disable="true"`. For a replica table, the corresponding column is automatically removed. For a history table, the column remains but is marked as disabled. See Disabling history on a specific field or group [p 462] and Disabling replication on a specific field or group [p 471].
- The facets of a field can be modified, except for the facets listed under Limitations/restrictions [p 476].

The following changes are accepted, but they can lead to a loss of data. Data should be migrated manually, by exporting then re-importing an XML or archive file, since these changes are considered to be a combination of deletion and creation:

- A field can be renamed.
- The type of a field can be changed.

89.2 **Limitations/restrictions**

Limitations related to primary key evolutions

When a primary key definition is modified:

- The content of the table will be reset to an empty content, in all datasets and dataspaces.

- If the new primary key has been used in the past, the content of the table will be reset to the previous data existing at the time this primary key was used, in all datasets and dataspaces.
- Type conversion is not supported for a primary key field. Thus, the content of the table will always be reset to empty when changing the type of a primary key even when reusing a type already used in previous primary key definition.
- The modification will be rejected if the table has - or has had - history activated in the existing dataspaces. A possible workaround: first drop the history table associated with the dedicated table, then proceed to modifying the primary key. For the procedure to purge mapped table database resources, see Database mapping [p 711].

> **Note**
>
> If the modified primary key is referenced in the primary key of another table, all the limitations mentioned above apply to the target table.

Limitations related to foreign key evolutions

- When the declaration of a `osd:tableRef` facet is added or modified, or when the primary key of its target table is modified, the existing values will restart from empty (except if this modification is reverting to a previous definition; in this case, the previous content will be retrieved).
- In replication mode, the structure of a foreign key field is set to match that of the target primary key. A single field declaring an `osd:tableRef` constraint may then be split into a number of columns, whose number and types correspond to that of the target primary key. Hence, the following cases of evolutions will have an impact on the structure of the mapped table:
  - declaring a new `osd:tableRef` constraint on a table field;
  - removing an existing `osd:tableRef` constraint on a table field;
  - adding (resp. removing) a column to (resp. from) a primary key referenced by an existing `osd:tableRef` constraint;
  - modifying the type or path for any column of a primary key referenced by an existing `osd:tableRef` constraint.

  These cases of evolution will translate to a combination of field deletions and/or creations. Consequently, the existing data should be migrated manually.

Limitations related to field-level evolutions

When changing the type of a field to an incompatible type or cardinality, the field will be considered as a new one, and start with an empty content. The previous content will be retrieved if the model is rolled back to a previous definition.

- The following types are fully inter-convertible (meaning these types have the same exact persistent representation, and can be substituted to each other in the following charts):
  - `xs:string`
  - `osd:color`
  - `osd:datasetName`
  - `osd:dataspaceKey`
  - `osd:email`

  - `osd:html`
  - `osd:local`
  - `osd:resource`
  - `xs:nmtoken`
  - `xs:nmtokens`
  - `osd:text`
  - `xs:anyUri`
  - `xs:name`
- The following conversions are fully supported (that is, regardless of their cardinalities):
  - `xs:decimal` to `xs:string`
  - `xs:datetime` to `xs:string`
  - `xs:date` to `xs:string`
  - `xs:integer` to `xs:string`
  - `xs:int` to `xs:decimal`
  - `xs:integer` to `xs:decimal`
  - `xs:decimal` to `xs:integer` (losing the decimal part)
  - `xs:int` to `xs:integer`
  - `xs:datetime` to `xs:date` (losing the time part)
  - `xs:date` to `xs:datetime` (defaulting the time part to 0)
- The following conversions are possible only if the original type is single-valued:
  - `xs:boolean` to `xs:string`
  - `xs:time` to `xs:string`
  - `xs:int` to `xs:string`
  - `xs:long` to `xs:string`

The cardinality of a type can be changed; when the conversion is supported, it has the following behavior:

- When changing a single element to an aggregated list, the previous single value is preserved and added to the new aggregated list.
- When changing an aggregated list to a single element, only the last value of the aggregated list is preserved in the single element. Other values are lost.

Attention

Groups and complex types do not support conversion to (and from) any other types. Moreover, when a group or complex type changes between single-occurrenced and multi-occurrenced, the conversion is supported only if the group or complex type is terminal.

Other

CHAPTER 90

Inheritance and value resolution

This chapter contains the following topics:

1. Overview
2. Dataset inheritance
3. Inherited fields
4. Optimize & Refactor service

90.1 Overview

The principle of inheritance is to mutualize resources that are shared by multiple contexts or entities. TIBCO EBX® offers mechanisms for defining, factorizing and resolving data values: *dataset inheritance* and *inherited fields*.

Furthermore, *functions* can be defined to compute values.

> **Note**
>
> Inheritance mechanisms described in this chapter should not be confused with "structural inheritance", which usually applies to models and is proposed in UML class diagrams for example.

See also *Inheritance (glossary)* *[p 37]*

Dataset inheritance

Dataset inheritance is particularly useful when data applies to global enterprise contexts, such as subsidiaries or business partners.

Given a hierarchy of datasets, it is possible to factorize common data into the root or intermediate datasets and define specialized data in specific contexts.

The dataset inheritance mechanisms are detailed below in Dataset inheritance [p 481].

Inherited fields

Contrary to dataset inheritance, which exploits global built-in relationships between datasets, inherited fields exploit finer-grained dependencies that are specific to the data structure. It allows factorizing and specializing data at the business entities-level.

For example, if the model specifies that a 'Product' is associated with a 'FamilyOfProducts', it is possible that some attributes of 'Product' inherit their values from the attributes defined in the associated 'FamilyOfProducts'.

> **Note**
>
> When using both inheritance mechanisms in the same dataset, field inheritance has priority over dataset inheritance.

See also *Inherited fields performance considerations [p 617]*

Computed values (functions)

In the data model, it is also possible to specify that a node holds a *computed value*. In this case, the specified JavaBean function will be executed every time the value is requested.

The function is able to take into account the current context, such as the values of the current record or computations based on another table, and to send requests to third-party systems.

See also *Computed values [p 951]*

90.2 Dataset inheritance

Dataset inheritance declaration

The dataset inheritance mechanism is declared from DMA in the data model (Configuration > Data model properties) or directly in the XML Schema Definition, when using (Actions > Import data model):

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ebxbnd="urn:ebx-schemas:binding_1.0">
  <xs:annotation>
    <xs:appinfo>
      <osd:inheritance>
        <dataSetInheritance>all</dataSetInheritance>
      </osd:inheritance>
    </xs:appinfo>
  </xs:annotation>
   ...
</xs:schema>
```

The element `osd:inheritance` defines the property `dataSetInheritance` to specify the use of inheritance on datasets based on this data model. The following values can be specified:

- `all`, indicates that inheritance is enabled for all datasets based on the data model.
- `none`, indicates that inheritance is disabled for all datasets based on the data model.

If not specified, the inheritance mechanism is disabled.

Value lookup mechanism

The dataset inheritance lookup mechanism for values proceeds as follows:

1. If the value is locally defined, it is returned.

   It can be explicitly `null`.
2. Otherwise, looks up the first locally defined value according to the built-in child-to-parent relationship of the dataset in the hierarchy of datasets.
3. If no locally defined value is found, the default value is returned.

If no default value is defined, `null` is returned.

Note: Default values cannot be defined on:

- A single primary key node
- Auto-incremented nodes
- Nodes defining a computed value

Record lookup mechanism

Like values, table records can also be inherited as a unit by multiple contexts, but they can also be partially redefined (*overwritten*), defined for a specific context (*root mode*), or be *occulted*.

Formally, a table record has one of four distinct definition modes:

root record	Locally defined in the table and has no parent. This means that no record with the same primary key exists in the parent table, or that this parent is an occulting record.
overwriting record	Locally defined in the table and has a parent record. This means that a record with the same primary key exists in the parent table, and that this parent is not an occulting record. The overwriting record inherits its values from its parent, except for the values that it explicitly redefines.
inherited record	Not locally defined in the current table and has a parent record. All values are inherited. Functions are always resolved in the current record context and are not inherited.
occulting record	Specifies that, if a parent with the same primary key is defined, this parent will not be visible in table descendants.

See also *Dataset inheritance [p 169]*

Defining inheritance behavior at the table level

It is also possible to specify management rules in the declaration of a table in the data model.

See also *Properties related to dataset inheritance [p 917]*

90.3 Inherited fields

The specific inheritance mechanism allows fetching a value of a field according to its relationship to other tables.

Field inheritance declaration

Specific inheritance must be specified:

- on the node's advanced properties in the underlying data model.

- on terminal nodes in the underlying data model and is declared as follows:

```
<xs:element name="sampleInheritance" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:inheritance>
    <sourceRecord>
     /root/table1/fkTable2, /root/table2/fkTable3
    </sourceRecord>
    <sourceNode>color</sourceNode>
   </osd:inheritance>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

The element `sourceRecord` is an expression that describes how to look up the record from which the value is inherited. It is a foreign key, or a sequence of foreign keys, from the current element to the source table.

If `sourceRecord` is not defined in the data model, the inherited fields are fetched from the current record.

The element `sourceNode` is the path of the node from which to inherit in the source record.

The following conditions must be satisfied for specific inheritance:

- The element `sourceNode` is mandatory.
- The expression for the path to the source record must be a consistent path of foreign keys, from the current element to the source record. This expression must involve only one-to-one and zero-to-one relationships.
- The `sourceRecord` cannot contain any aggregated list elements.
- Each element of the `sourceRecord` must be a foreign key.
- If the inherited field is also a foreign key, the `sourceRecord` cannot refer to itself to get the path to the source record of the inherited value.
- Every element of the `sourceRecord` must exist.
- The source node must belong to the table containing the source record.
- The source node must be terminal.
- The source node must be writeable.
- The source node type must be compatible with the current node type.
- The source node cardinalities must be compatible with those of the current node.
- The source node cannot be the same as the inherited field if the fields to inherit from are fetched into the same record.

Value lookup mechanism

The lookup mechanism for inherited fields values proceeds as follows:

1. If the value is locally defined, it is returned.

   It can be explicitly `null`
2. Otherwise, looks up the source record and value to inherit from, according to the properties that are defined in the data model.
3. The process is recursive; if the source node does not locally define a value, it is then looked up according to the inheritance behavior of the source node.

Disabling validation

In some cases, it can be useful to disable the validation of inherited fields for performance purpose. The property `osd:disableValidation= "true"` specified in the data model allows to disable the validation of an inherited fied (see example below).

Example:

```
<xs:element name="sampleInheritance" type="xs:string" osd:disableValidation="true">
 <xs:annotation>
  <xs:appinfo>
   <osd:inheritance>
    <sourceRecord>
     /root/table1/fkTable2, /root/table2/fkTable3
    </sourceRecord>
    <sourceNode>color</sourceNode>
   </osd:inheritance>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

90.4 Optimize & Refactor service

EBX® provides a built-in user service for optimizing the dataset inheritance in the hierarchy of datasets. This service performs the following functions:

- **Handles duplicated values:** Detects and removes all parameter values that are duplicates of the inherited value.
- **Mutualizes common values:** Detects and mutualizes the common values among the descendants of a common ancestor.

Procedure details

Datasets are processed from the bottom up, which means that if the service is run on the dataset at level *N*, with *N+1* being the level of its children and *N+2* being the level of its children's children, the service will first process the datasets at level *N+2* to determine if they can be optimized with respect to the datasets at level *N+1*. Next, it would proceed with an optimization of level *N+1* against level *N*.

> **Note**
>
> - These optimization and refactoring functions do not handle default values that are declared in the data model.
> - The highest level considered during the optimization procedure is always the dataset on which the service is run. This means that optimization and refactoring are not performed between the target dataset and its own ancestors.
> - Table optimization is performed on records with the same primary key.
> - Inherited fields are not optimized.
> - *The optimization and refactoring functions do not modify the resolved view of a dataset, if it is activated.*

Service availability

The 'Optimize & Refactor' service is available on datasets that have child datasets and have the 'Activated' property set to 'No' in their dataset information.

The service is available to any profile with write access on current dataset values. It can be disabled by setting restrictive access rights on a profile.

> **Note**
>
> For performance reasons, access rights are not verified on every node and table record.

CHAPTER 91

Permissions

Permissions dictate the access each user has to data and actions.

This chapter contains the following topics:

1. Overview
2. Important considerations about permissions
3. Defining confidentiality
4. Defining user-defined rules
5. Defining dynamic rules
6. Resolving permissions on data
7. Resolving permissions on services
8. Resolving permissions on actions

91.1 Overview

Permissions are related to whether actions are authorized or not. They are also related to access rights, that is, whether an entity is hidden, read, or read-write. The main entities controlled by permissions are:

- Dataspace
- Dataset
- Table
- Group
- Field

Users, roles and profiles

The definition and resolution of permissions make extensive use of the notion of *profiles*, which is the generic term applied to users or roles.

Each user can participate in several roles, and a role can be shared by several users.

These relationships are defined in the user and roles directory. See Users and roles directory [p 703].

Special definitions:

- A *built-in administrator* is a member of the built-in role 'ADMINISTRATOR'.

- An *owner of a dataset* is a member of the *owner* attribute specified in the information of a root dataset. In this case, the built-in role 'OWNER' is activated when permissions are resolved in the context of the dataset.
- An *owner of a dataspace* is a member of the *owner* attribute specified for a dataspace. In this case, the built-in role 'OWNER' is activated when permissions are resolved in the context of the dataspace.

Permission rules

A permission rule defines the authorization granted to a profile for a particular entity.

User-defined permission rules are created through the user interface. See the section Defining user-defined rules [p 492].

Dynamic permission rules can be either programmatic rules created by developers, or scripted rules created by administrators. See the section Defining dynamic rules [p 499].

Resolution of permissions

Permissions are always resolved in the context of an authenticated user session, thus permissions are mainly based on the user profiles.

In general, resolution of permissions is performed restrictively between a given level and its parent level. Thus, at any given level, a user cannot have a higher permission than the one resolved at a parent level.

Dynamic permissions are always considered to be restrictive.

> **Note**
>
> In the Java API, the class `SessionPermissions`[API] provides access to the resolved permissions.

See also

Resolving permissions on data [p 501]

Resolving permissions on services [p 504]

Resolving permissions on actions [p 506]

Owner and administrator special permissions

On a dataset

A built-in administrator or owner of a dataset can perform the following actions:

- Manage its permissions
- Change its owner, if the dataset is a root dataset

- Change its general information (localized labels and descriptions)

> **Attention**
> While the definition of permissions can restrict a built-in administrator or dataset owner's right to view data or perform certain actions, it remains possible for them to modify their own access, as they will always have access to permissions management.

On a dataspace

To be a *super owner* of a dataspace, a user must either:

- Own the dataspace and be allowed to manage its permissions, or
- Own a dataspace that is an ancestor of the current dataspace and be allowed to manage the permissions of that ancestor dataspace.

A built-in administrator or super owner of a dataspace can perform the following actions:

- Manage its permissions of dataspace.
- Change its owner
- Lock it or unlock it
- Change its general information (localized labels and descriptions)

Furthermore, in a workflow, when using a "Create a dataspace" or "Create a snapshot" built-in script task, resolved permissions are computed using the owner defined in the script task's configuration, rather than the current session. This is because, in these cases, the current session is associated with a system user.

> **Attention**
> While the definition of permissions can restrict a built-in administrator or dataspace owner's right to view data or perform certain actions, it remains possible for them to modify their own access, as they will always have access to permissions management.

Impact of merge on permissions

When a dataspace is merged, the permissions of the child dataset are merged with those of the parent dataspace if and only if the user specifies to do so during the merge process. The permissions of its parent dataspace are never impacted.

If some elements are hidden for the profile attempting to perform a merge, it will not be possible to proceed as the impacts of the merge on data will not be fully visible.

91.2 Important considerations about permissions

In this section are listed some very important information that must be kept in mind while working with permissions.

Actions and user services granting high privileges

The following actions and their related user services must only be allowed to trusted administrators:

- 'Create dataspace' (gives the 'owner' role, which grants the right to define the dataspace permissions)
- 'Create dataset' (gives the 'owner' role, which grants the right to define the dataset permissions)
- 'Import archive' (allows writing the archive content regardless of any permission)

> **Note**
>
> See the Owner and administrator special permissions [p 488] section for more information about the privileges granted to these profiles.

API access without permission checks

Developers and administrators must be aware that some parts of the API can run without any permission check. In general if the code run in a context with a `Session`[API] provided, it means that permissions will be checked. Here are some specific cases where permissions are not checked:

- When a Java procedure disables all permission checks by using `ProcedureContext.setAllPrivileges`[API].
- When accessing EBX® data by directly querying your database, in the case a table enables the replication mode [p 458], or the historization [p 458]. This is because EBX® permissions are not "translated" in the underlying database. As a consequence, either the database access must be globally restricted or proper permissions must be defined in it.

Using permission for hiding information in the UI

Using the permissions only to hide in the UI some non sensitive information is unadvised, especially if this information is likely to be used for filtering / joining / sorting in some queries. In such cases, UI-only hiding methods should be used instead. For instance by setting the field as hidden for default views [p 963] in the data model property and/or by creating views [p 685] for the concerned users.

When the above suggestions can not be applied and permissions have to be used to hide non sensitive information, the nodes must be set as non confidential in the model. For more information, see Defining confidentiality [p 492].

Limitations of the permission checks in Query API

The permission check performed when specifying a session in a `Query`[API] or `Request`[API] will throw a `QueryPermissionException`[API] if any confidential field used in the query is hidden for the current user. All fields are considered confidential except:

- Fields belonging to primary keys;
- Fields explicitly defined as non-confidential. For more information , see Defining confidentiality [p 492];

> **Attention**
>
> Confidential nodes with a record-dependent `AccessRule`[API] are only partially confidential. Indeed, the method `AccessRule.getPermission`[API] is only called with a dataset as the `aDataSetOrRecord` parameter while performing the permission check, never with a record. As a consequence, the record-dependent logic of the rule will not apply during this check.

Scripted permission rules on records and table history

You can use scripted permission rules to filter table records and table history. By default, table history access is disabled for all users. This behavior helps enforce security. However, you can create a list of profiles authorized to access table history by editing the permission rule and adding the desired profiles to the **Profiles authorized to access history** list.

Using hidden fields in custom display labels

Resolution of custom display labels for tables ('defaultLabel' property) and relationships ('display' property) takes into account permission. As soon as an hidden field is detected in the label, the primary key will be displayed instead.

> **Note**
>
> This is not the case when using API like `TableRefDisplay`[API] or `Adaptation.getLabelOrName`[API]. Since the provided contexts do not contain the current session, no permission check can be performed. As a consequence, developer should make sure that no confidential data is exposed when using these APIs.

> **Note**
>
> Also note that quick search will ignore nodes with hidden fields in custom display label in the context of history view and/or in a child dataset.

Because of this behavior it is highly discouraged to use labels for filtering in a query. When labels with hidden fields are used, it will be replaced by the pk value and the filter will become inconsistent.

Linked field permission check

When a linked field [p 933] access permission is computed, the result is the minimum between the permission applying to the node in the main table and the node in the target table. Practically it means that if a field is hidden in a table, all linked fields pointing on it in other tables will also be hidden.

Table action permission related limitations

When performing actions on a table (create, delete, overwrite or occult) in a procedure, the current user session access right on the table node is ignored during the permission resolution. Should this check be performed, the client code must explicitly call `SessionPermissions.getNodeAccessPermission`[API] beforehand in the procedure.

Permission cache life cycle

To optimize the resolution of permissions for both data and user services, a dedicated cache is implemented at the session level. All permissions are cached including dynamic rules, it means that a rule result should not change for the duration of the cache which is explained below.

The session cache life cycle depends on the context, as described hereafter:

- In the UI, the cache is cleared for every non-ajax event (i.e on page display, pop-up opening, etc.).

- In programmatic procedures, the cache lasts until the end of the procedure, unless explicitly cleared (see below).

> **Attention**
> When modifying permissions in a procedure context (by importing an EBX® archive or merging a dataspace programmatically), the session cache **must** be cleared via a call to `Session.clearCache`[API]. Otherwise, these modifications will not be reflected until the end of the procedure.

Permissions on a dataset with fatal errors

When a dataset has fatal errors, only owners and administrators have access and only the following actions are available:

- View the validation report
- View the information page
- Delete the dataset

91.3 Defining confidentiality

The confidentiality of a node defines whether or not it can be used in a `Query`[API] or a `Request`[API] when hidden for a specific user. By default, all nodes are confidential.

Defining confidentiality on a node has the following impacts on the relationships and views that use it:

- A relationship (association or table reference) is hidden if at least one of its defining nodes is confidential and hidden.
- A relationship is hidden if at least one of the nodes used in its additional XPath filter is confidential and hidden.
- A view is hidden if at least one of the nodes that is used in its filter criteria or in its sort criteria is confidential and hidden.

For details about relationship defining nodes and relationship additional XPath filters see Foreign keys [p 919] and Associations [p 923].

> **Attention**
> It is recommended to be very vigilant before setting a field as non confidential because although hidden, a malicious user could then "guess" the value by filtering or sorting using this node. It can not always be done, but it is generally a better option to use views to handle these kinds of use cases, as recommended in Using permission for hiding information in the UI [p 490].

The confidentiality of nodes can be configured in the DMA. For more details, see Node confidentiality [p 93].

91.4 Defining user-defined rules

Each level has a similar schema, which allows defining permission rules for profiles.

Defining dataspace user-defined rules

For a given dataspace, the allowable permissions for each profile are as follows:

Dataspace access	Authorization
Write	• Can view the dataspace. • Can access datasets according to dataset permissions.
Read-only	• Can view the dataspace and its snapshots. • Can view child dataspaces, if allowed by permissions. • Can view contents of the dataspace, though cannot modify them.
Hidden	• Can neither see the dataspace nor its snapshots. • If allowed to view child dataspace, can see the current dataspace but cannot select it. • Cannot access the dataspace contents, including datasets. • Cannot perform any actions on the dataspace.

Restriction policy	Indicates whether this dataspace profile-permission association should have priority over other permissions rules.
Create a child dataspace	Indicates whether the profile can create child dataspaces from the current dataspace.
Create a child snapshot	Indicates whether the profile can create snapshots of the current dataspace.
Initiate merge	Indicates whether the profile can merge the current dataspace with its parent dataspace.
Export archive	Indicates whether the profile can export the current dataspace as an archive.
Import archive	Indicates whether the profile can import an archive into the current dataspace.
Close a dataspace	Indicates whether the profile can close the current dataspace.
Close a snapshot	Indicates whether the profile can close a snapshot of the current dataspace.
Rights on services	Indicates if a profile has the right to execute services on the dataspace. By default, all dataspace services are

	allowed. A built-in administrator or super owner of the current dataspace or a given user who is allowed to modify permissions on the current dataspace can modify these permissions to restrict dataspace services for certain profiles.
Permissions of child dataspace when created	When a user creates a child dataspace, the permissions of this new dataspace are automatically assigned to the profile's owner, based on the permissions defined under 'Permissions of child dataspace when created' in the parent dataspace. If multiple permissions are defined for the owner through different roles, the owner's profile behaves like any other profile and permissions are resolved [p 488] as usual.

Defining dataset user-defined rules

For a given dataset, the allowable permissions for each profile are as follows:

Actions on datasets

Restriction policy	Indicates whether this dataset profile-permission association should have priority over other permissions rules.
Create a child dataset	Indicates whether the profile has the right to create a child dataset of the current dataset.
Duplicate dataset	Indicates whether the profile has the right to duplicate the current dataset.
Change the dataset parent	Indicates whether the profile has the right to change the parent dataset of a given child dataset.

Actions on tables

The action rights on default tables are defined at the dataset level. It is then possible to override these default rights for one or more tables. The allowable permissions for each profile are as follows:

Create a new record	Indicates whether the profile has the right to create records in the table.
Overwrite inherited record	Indicates whether the profile has the right to overwrite inherited records in the table.
Occult inherited record	Indicates whether the profile has the right to occult inherited records in the table.
Delete a record	Indicates whether the profile has the right to delete records in the table.

Access rights on node values

Permissions defined on specific terminal nodes override their default access rights.

Read-write	Can view and modify node values.
Read	Can view nodes, but cannot modify their values.
Hidden	Cannot view nodes.

Permissions on services

A built-in administrator or an owner of the current dataspace can modify the service default permission to either restrict or grant access to certain profiles.

Enabled	Grants service access to the current profile.
Disabled	Forbids service access to the current profile. It will not be displayed in menus, nor will it be launchable via web components.
Default	Sets the service permission to enabled or disabled, according to the default permission defined upon service declaration. See `ActivationContext.setDefaultPermission`[API] for more information.

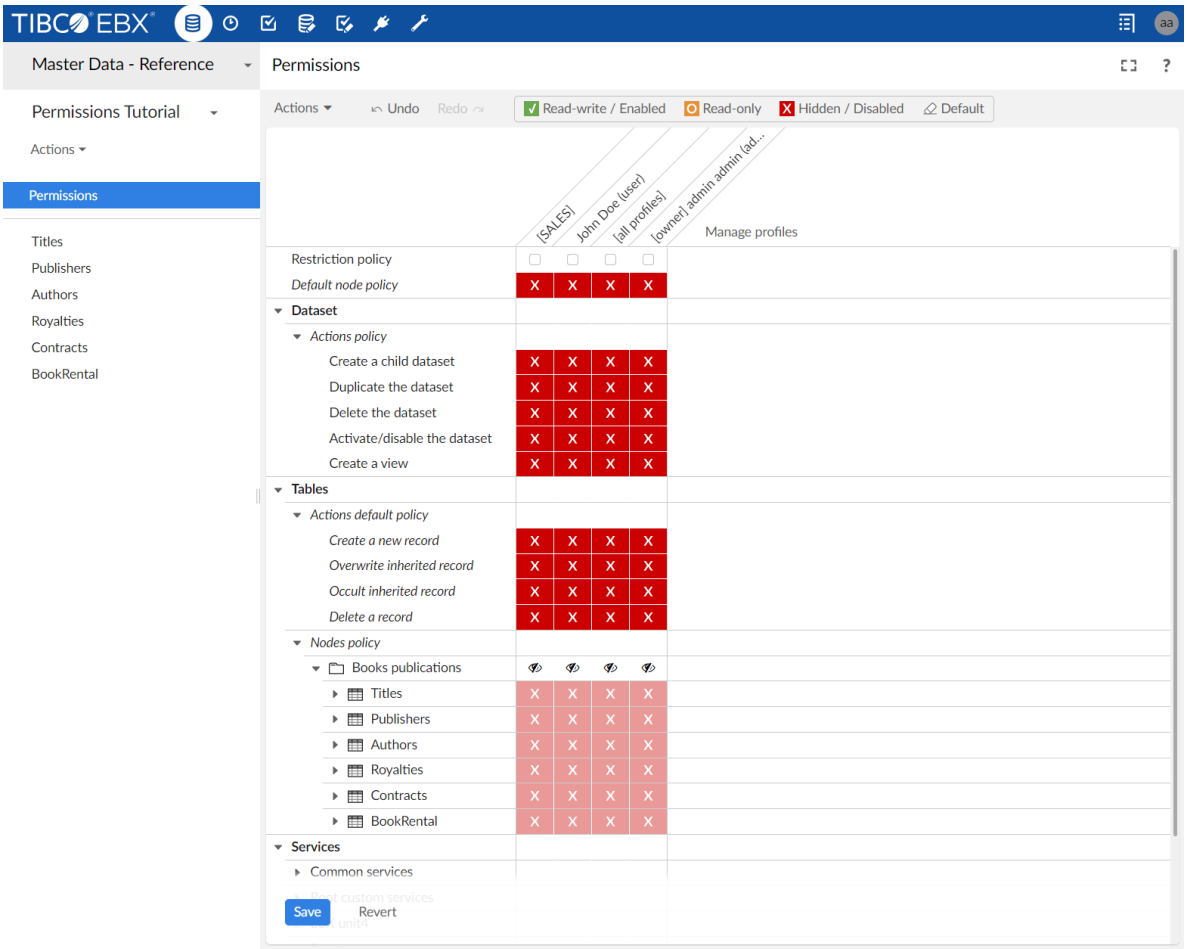
Using the Dataset Permissions configuration service

User-defined access rules at the dataset level are defined using the **Permissions** service available from the dataset's **Actions** menu. From this service, administrators can review and configure user access to data, actions, and services inside this dataset.

Permissions service UI overview

The service is presented as a grid in which:

- Each column represents a rule for one profile in the dataset. As shown in the example screenshot below, the profiles are: SALES, John Doe (user), [all profiles], and [owner] admin admin. See Managing profiles [p 496] for more information.
- Each line represents a node, an action, or a service on which a rule applies. See Managing permissions [p 498] for more information.
- Each modifiable cell represents a permission level to access a node, an action, or a service for a given profile.



Managing profiles

Profile management tasks include:

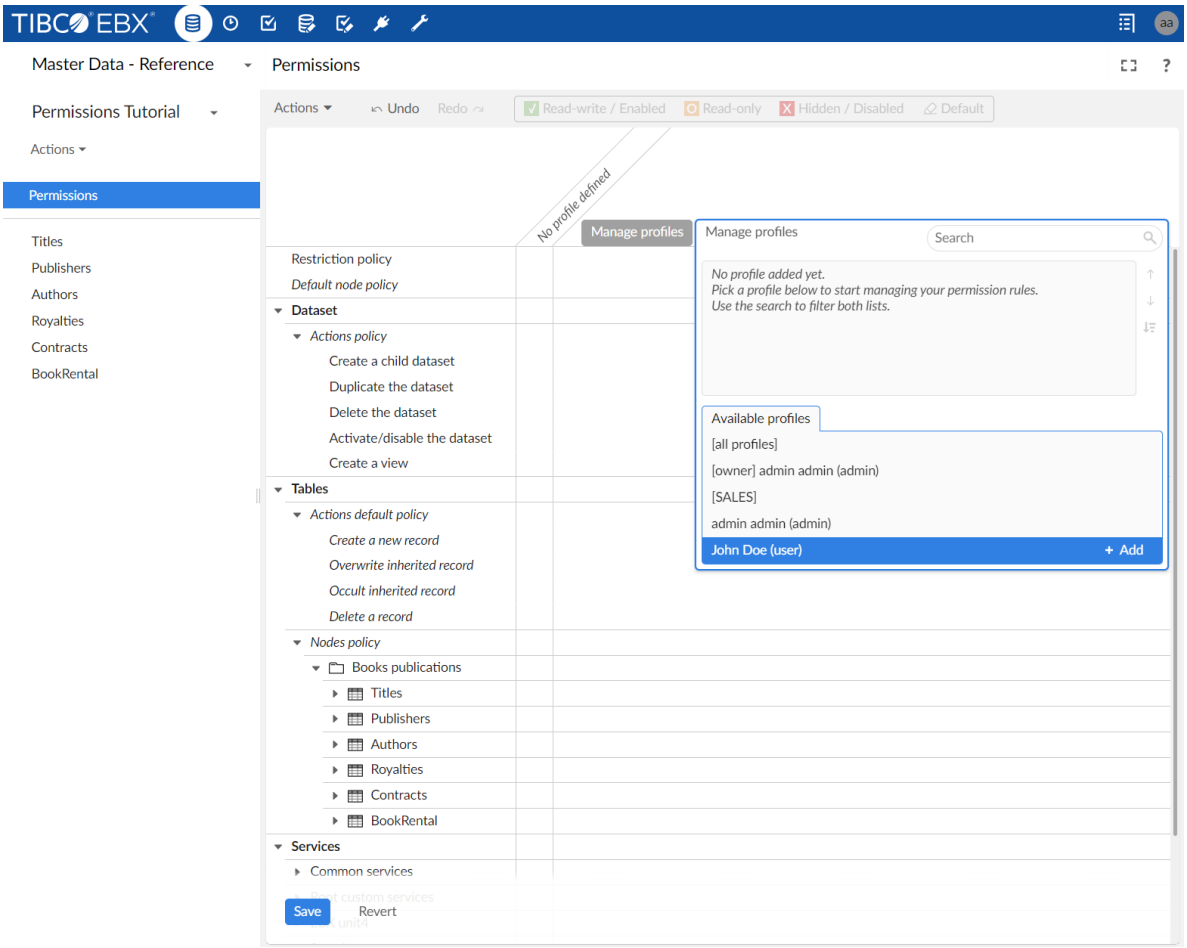
- Adding a user-defined access rule [p 497]

- Duplicating a profile configuration into another profile [p 497]
- Deleting a profile configuration [p 497]

Adding a user-defined access rule

To add a rule for a profile:

- Click the **Manage profiles** button at the top of the grid.
- In the popup menu, select or hover over a profile from the **Available profiles** tab.
- Click the **Add** button.



Duplicating a profile configuration into another profile

To duplicate an existing profile's configuration into another profile:

- Click the **Manage profiles** at the top of the grid.
- In the popup menu, select or hover over the source profile in the **Manage profiles** window.
- Click the **Duplicate** button that displays to the right of the source profile's name.
- Select the destination profile under the **Available profiles**.

Deleting a profile configuration

To delete a profile configuration:

- Click the **Manage profiles** at the top of the grid.

- In the popup menu, select or hover over a profile in the list.
- Click the **Delete** button that display to the right of the source profile's name.

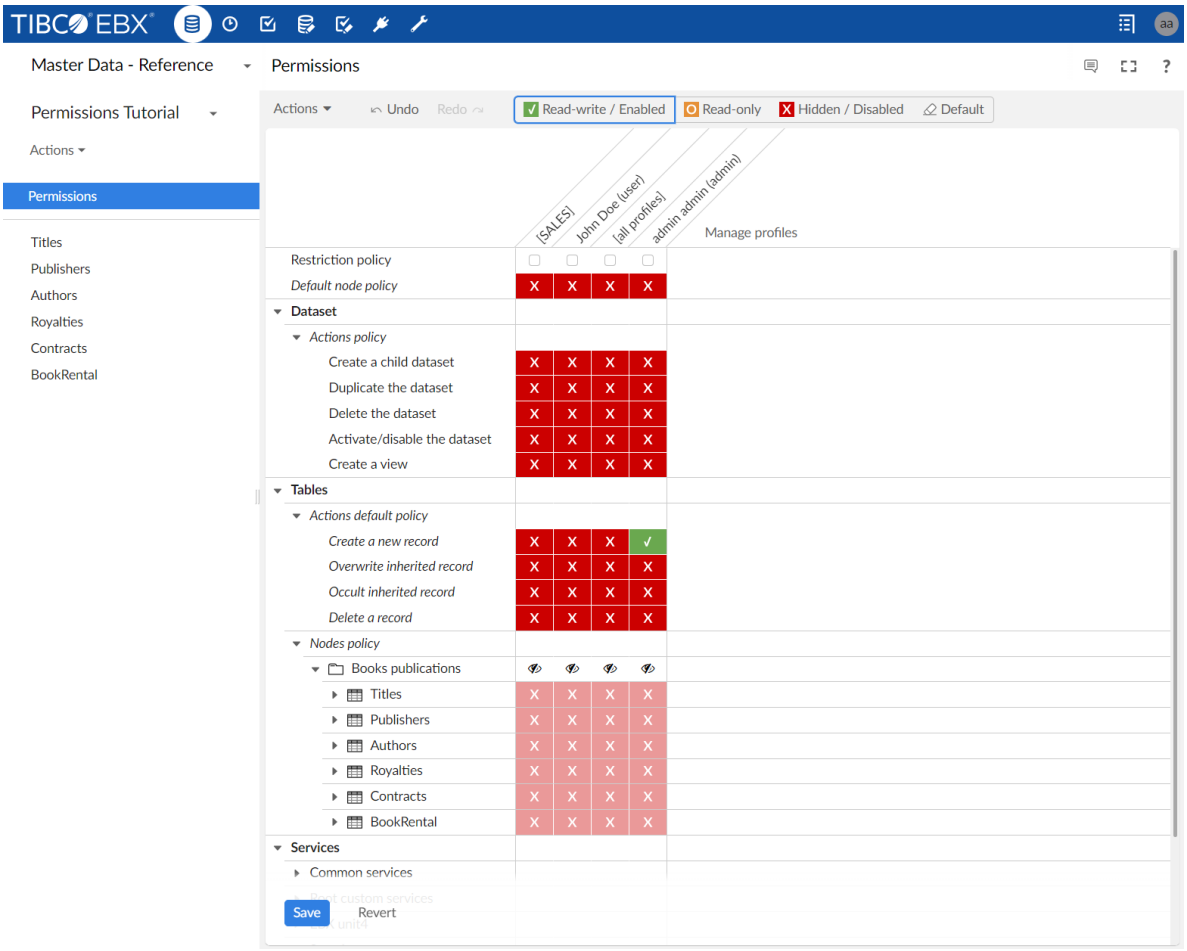
Managing permissions

There are 2 ways to assign permissions on nodes, actions, and services. You can:

- Select one or more cells, then apply a permission using the toolbar or a shortcut.
- Select a permission using the toolbar or a shortcut when no cell is selected, then apply this permission to one or more cells. When using this method an icon displays next to the mouse pointer and changes depending on the context.

> **Note**
>
> In the permissions toolbar, the **Read-Write / Enabled** and **Hidden / Disabled** value changes depending on whether the selection is a node, or an action/service. The actual value is **Read-Write** and **Hidden** for a node and **Enabled** and **Disabled** for an action/service.

In the screenshot below, the permission cell corresponding to the **Create a new record** service and to the user **John Doe** is set to **Enabled**:



91.5 Defining dynamic rules

Dynamic rules give the possibility to define more precisely the conditions for accessing data or user services depending on the context.

There are different types of programmatic rules:

- the `AccessRule`[API], described in the section below Defining access rules on data [p 499].
- the scripted record permission rule, described in the section below Defining scripted permission rules on data [p 499].
- the ServiceActivationRule [p 500], described in the section below Defining activation rules on service [p 500].
- the `ServicePermissionRule`[API], described in the section below Defining permission rules on service [p 500].

Defining scripted permission rules on data

scripted permission rules are rules that dynamically define, depending on the context, the read/write rights on the records of a table.

To define such a rule, a record permission script [p 1301] must be created in the DMA. A script editor is available on the table node definition, in the "Extensions" tab.

Defining access rules on data

`AccessRules` are rules that programmatically define, depending on the context, the read/write rights on a data model node or on the records of a table.

The definition of an `AccessRule` is performed as follows:

1. Creation of a rule in the form of a Java class implementing the `AccessRule`[API] or `AccessRuleForCreate`[API] interface.
2. Assignment of this rule to concerned nodes in the schema extension: `SchemaExtensions`[API].

   According to the rule target (model node(s) or records) and type (`AccessRule` or `AccessRuleForCreate`), several methods such as `SchemaExtensionsContext.setAccessRuleOnOccurrence`[API] or `SchemaExtensionsContext.setAccessRuleForCreateOnNode`[API] can be used.

   The rule thus assigned is said to be "local" and is only executed when the target entity is requested. See Resolving permissions on data [p 501] for more information.

> **Attention**
>
> Only one `AccessRule` can be defined for each node, dataspace or record. Only one `AccessRuleForCreate` can be defined for each table child node. The definition of a new programmatic rule of one type will lead to the replacement of the existing one.

Defining activation rules on service

The `ServiceActivationRules` allow to specify if a service is activated or not for a given dataspace or dataset. A service that has been deactivated through this rule is never available in the entity for which it is deactivated, regardless of the current profile, for execution or display, even in permission screens.

The definition of a `ServiceActivationRule` is carried out as follows:

1. Creation of a rule in the form of a Java class implementing the `ServiceActivationRuleForDataspace`[API] interface or `ServiceActivationRuleForDataset`[API], depending on the service type.
2. Assignment of this rule to the impacted services at their declaration level, depending on the service type, via the `ActivationContextOnDataspace.setActivationRule`[API] or `ActivationContextWithDatasetSet.setActivationRule`[API] methods.

   The resulting assigned rule will be evaluated during the service activation evaluation. See Resolving permissions on services [p 504] for more information.

Defining permission rules on service

The `ServicePermissionRules` are advanced rules allowing to dynamically define the display and execution conditions of a service depending on the context (current session, selected entity, etc.). The service should be activated for the current context beforehand for this type of rule to be triggered.

The definition of a `ServicePermissionRule` is carried out as follows:

1. Creation of a rule in the form of a Java class implementing the `ServicePermissionRule`[API] interface.
2. Assignment of this rule to the impacted services:
   - Either, for new services, at their declaration level via the `ActivationContext.setPermissionRule`[API] method.

     The rule thus assigned is said to be "global" and is only executed when the service is activated for the current context. See Resolving permissions on services [p 504] for more information.
   - Or, for existing services, in the **schema extension** `SchemaExtensions`[API] via the `SchemaExtensionsContext.setServicePermissionRuleOnNode`[API] and `SchemaExtensionsContext.setServicePermissionRuleOnNodeAndAllDescendants`[API] methods. It is thus possible to assign a rule to any service, including standard services provided by EBX®, on one or more data model nodes: a table node, an association node, etc.

     The rule thus assigned is said to be "local" and is only executed in the extended schema context and when the node corresponds to the one specified. See Resolving permissions on services [p 504] for more information.

> **Attention**
>
> Only one `ServicePermissionRule` can be defined for each model node. Thus, the definition of a new programmatic rule will replace the existing one.

91.6 Resolving permissions on data

Resolving user-defined rules

Access rights defined using the user interface are resolved on four levels: dataspace, dataset, record (if applicable) and node.

If a profile is associated with restrictive access rights at a given level, the minimum of all restrictive rights defined at that level is resolved. If no restrictions are defined at that level, the maximum of all access rights defined at that level is resolved.

When a restrictive permission is defined for a profile, it takes precedence over the other permissions potentially granted by the user's other roles. Generally, for all user-defined permission rules that match the current user session:

- If some rules with restrictions are defined, the minimum permissions of these restricted rules are applied.
- If no rules having restrictions are defined, the maximum permissions of all matching rules are applied.

Examples:

Given two profiles *P1* and *P2* concerning the same user, the following table lists the possibilities when resolving that user's permission to a service.

P1 authorization	P2 authorization	Permission resolution
Enabled	Enabled	Enabled. Restrictions do not make any difference.
Disabled	Disabled	Disabled. Restrictions do not make any difference.
Enabled	Disabled	Enabled, unless P2's authorization is a restriction.
Disabled	Enabled	Enabled, unless P1's authorization is a restriction.

The same restriction policy is applied for data access rights resolution.

In another example, a dataspace can be hidden from all users by defining a restrictive association between the built-in profile "Profile.EVERYONE" and the access right "hidden".

At any given level, the most restrictive access rights between those resolved at this level and higher levels are applied. For instance, if a user's dataset access permissions resolve to read-write access, but the container dataspace only allows read access, the user will only have read-only access to this dataset.

> **Note**
>
> The dataset inheritance mechanism applies to both values and access rights. That is, access rights defined on a dataset will be applied to its child datasets. It is possible to override these rights in the child dataset.

Access rights resolution example

In this example, there are three users who belong to the following defined roles and profiles:

User	Profile
User 1	• user1 • role A • role B
User 2	• user2 • role A • role B • role C
User 3	• user3 • role A • role C

The access rights of the profiles on a given element are as follows:

Profile	Access rights	Restriction policy
user1	Hidden	Yes
user3	Read	No
Role A	Read/Write	No
Role B	Read	Yes
Role C	Hidden	No

After resolution based on the role and profile access rights above, the rights that are applied to each user are as follows:

User	Resolved access rights
User 1	Hidden
User 2	Read
User 3	Read/Write

Resolving dataspace and snapshot access rights

At dataspace level, access rights are resolved as follows:

- If a user has several rights defined through multiple profiles:
  - If the rights include restrictions, the minimum of the restrictive profile-rights associations is applied.
  - Otherwise, the maximum of the profile-rights associations is applied.
- If the user has no rights defined:
  - If the user is a built-in administrator or owner of the dataspace, read-write access is given for this dataspace.
  - Otherwise, the dataspace will be hidden.

Resolving dataset access rights

At the dataset level, the same principle applies as at the dataspace level. After resolving the access rights at the dataset level alone, the final access rights are determined by taking the minimum rights between the resolved dataspace rights and the resolved dataset rights. For example, if a dataspace is resolved to be read-only for a user and one of its datasets is resolved to be read-write, the user will only have read-only access to that dataset.

Resolving node access rights

At the node level, the same principle applies as at the dataspace and dataset levels. After resolving the access rights at the node level alone, the final access rights are determined by taking the minimum rights between the resolved dataset rights and the resolved node rights.

Specific access rights can be defined at the node level. If no specific access right is defined, the default access right is used for the resolution process.

> **Note**
>
> The resolution procedure is slightly different for table and table child nodes.

Special case for table and table child nodes

This describes the resolution process used for a given table node or table record *N*.

For each user-defined permission rule that matches one of the user's profiles, the access rights for *N* are either:

1. The locally defined access rights for *N*;
2. Inherited from the access rights defined on the table node;
3. Inherited from the default access rights for dataset values.

All matching user-defined permission rules are used to resolve the access rights for *N*. Resolution is done according to the restriction policy [p 501].

The final resolved access rights will be the minimum between the dataspace, dataset and the resolved access right for *N*.

Resolving dynamic rules

There are three levels of resolution for dynamic access right rules: dataset, record and node. Since only one programmatic access rule can be set for a given level, the last rule set is the one used by the resolution procedure. However, a scripted rule can be specified on top of a programmatic rule at the table level.

Rule resolution on dataset

For a dataset, the last rule set is considered as the resolved rule

Rule resolution on record

For a record, the resolved rule is the minimum between the resolved rule set on the dataset and the rule set on this record.

See `SchemaExtensionsContext.setAccessRuleOnOccurrence`[API] for more details.

Rule resolution on node

For a node that is a child node of a record, the resolved rule is the minimum between the resolved rule on the record and the rule set on this node.

For a child node of a dataset, the resolved rule is the minimum between the resolved rule set on the dataset and the rule set on this node.

See `SchemaExtensionsContext.setAccessRuleOnNode`[API] for more details.

Display policy for foreign key drop-down menus

If a record is hidden due to access rules, it will not appear in foreign key drop-down menus.

> **Attention**
> The resolved access rights on a dataset or dataset node is the minimum between the resolved access rights defined in the user interface and the resolved dynamic rules, if any.

91.7 Resolving permissions on services

User services give the possibility to execute specific and advanced features from the user interface. Depending on their definition, these services can be called from a menu, as an action in a workflow, as a perspective item, or can be executed directly from a URL as a Web component [p 422].

See also *Overview* *[p 1027]*

The permissions of a service are resolved as the service is called from the user interface, namely:

- During the execution, just before the service is displayed.

  If the permission resolved in the user context is not `enabled`, a restriction message is displayed in place of the service.
- During the display of menus if the service is defined as displayable in menus.

  If the permission resolved in the context for the user is not `enabled`, the service will not be displayed in the menu.

Thus, upon every request the resolution of permissions for a service is carried out as follows, in the following order and as long as conditions are respected:

1. The service activation has to correspond to the current context. This activation considers:
   - the selected entity type (dataset, table, record, etc.);
   - static activation rules defined within the `UserServiceDeclaration.defineActivation`[API] method;
   - the potential dynamic activation rule (ServiceActivationRule [p 500]) also defined within the `UserServiceDeclaration.defineActivation`[API] method.
2. When the service is activated for the current context, permissions for the user session will be evaluated:
   - If permissions have been defined via the user interface for the current user (or for their roles), their resolution must return `enabled`.

     For more information, please refer to the Resolving user-defined rules [p 505] section.
   - If a global permission rule [p 500] is defined for the service, it must return `enabled` for the context provided (see `ServicePermissionRuleContext`[API]).
   - If a local permission rule [p 500] is defined for the selected node, it must return `enabled` for the context provided (see `ServicePermissionRuleContext`[API]).

Resolving user-defined rules

Example

In this example, there are two users belonging to different roles and profiles:

User	Profiles
User 1	• user1 • role A • role B
User 2	• role C • role D

The permissions associated with the roles and profiles defined on the dataset level are as follows:

Profile	**Built-in service create** (`@creation`)	**Built-in service duplicate** (`@duplicate`)	**Built-in service compare** (`@compare`)	**Custom service 1** (`custom1`)	**Custom service 2** (`custom2`)	**Restriction policy**
user1	Enabled	Disabled	Enabled	Disabled	Enabled	No
Role A	Enabled	Enabled	Disabled	Enabled	Disabled	Yes
Role B	Enabled	Disabled	Enabled	Enabled	Disabled	Yes
Role C	Enabled	Enabled	Disabled	Disabled	Disabled	No
Role D	Enabled	Disabled	Disabled	Enabled	Disabled	No

The services available to each user after permission resolution are as follows:

Users	**Available services**
User 1	Built-in service create (`@creation`)
	Custom service 1 (`custom1`)
User 2	Built-in service create (`@creation`)
	Built-in service duplicate (`@duplicate`)
	Custom service 1 (`custom1`)

See also *Resolving user-defined rules* *[p 501]*

91.8 Resolving permissions on actions

Actions are low-level operations for EBX® object manipulation on which it is possible to define execution rights for a profile. Unlike permissions on user services, which only impact the user interface, these rights are also applicable when an operation is carried out programmatically (i.e. via a `Procedure`[API]) or indirectly (for example during data import, actions on the table (create, override, occult and delete) are evaluated).

Here is the list of actions on which rights can be defined:

Action object	Available actions
Dataspace	Create a child dataspace
	Create a snapshot
	Launch a merge
	Export an archive
	Import an archive
	Close the dataspace
	Close the snapshot
	Create a dataset
Dataset	Duplicate the dataset
	Delete the dataset
	Activate/deactivate the dataset
	Create a view
Table	Create a new record
	Override records
	Occult records
	Delete records

For the resolution of permissions on actions, only the permissions defined via the user interface for the current user (or their roles) will be taken into account, the restriction policy being applied as for any other permission defined via the user interface.

For more information, please refer to the Resolving user-defined rules [p 508] section.

Resolving user-defined rules

Example

In this example, we have two users belonging to different roles and profiles:

User	Profiles
User 1	• user1 • role A • role B
User 2	• role C • role D

Rights associated with roles and profiles on the actions of a given table are as follows:

Profile	Create a record	Override a record	Occult a record	Delete a record	Restriction policy
user1	No	Yes	No	Yes	No
Role A	Yes	No	Yes	No	Yes
Role B	No	Yes	Yes	No	Yes
Role C	Yes	No	No	No	No
Role D	No	No	Yes	No	No

The actions available to each user after resolving the rights are as follows:

Users	Available actions
User 1	Occult a record
User 2	Create a record
	Occult a record

See also *Resolving user-defined rules* *[p 501]*

CHAPTER 92

Criteria editor

This chapter contains the following topics:

1. Overview
2. Conditional blocks
3. Atomic criteria

92.1 Overview

The criteria editor is included in several different areas of the user interface. It allows defining table filters, as well as validation and computation rules on data. This editor is based on the XPath 1.0 W3C Recommendation.

Two types of criteria exist: atomic criteria and conditional blocks.

See also *Supported XPath syntax* *[p 441]*

92.2 Conditional blocks

Conditional blocks are made up of atomic criteria and other conditional blocks. They express a condition based on the criteria. The following types of blocks exist:

- **No criteria match**: None of the criteria in the block match.
- **Not all criteria match**: At least one criterion in the block does not match.
- **All criteria match**: All criteria in the block match.
- **At least one criterion matches**: One or more of the criteria match.

92.3 Atomic criteria

An atomic predicate is defined by a field, an operator, and an expression (either a value or an XPath formula).

Field	Specifies the field of the table to which the criterion applies.
Operator	Specifies the operator used. Available operators depend on the data type of the field.
Value	Specifies the value or expression. See Expression [p 510] below.
Code only	If checked, specifies searching the underlying values for the field instead of labels, which are searched by default.

Expression

The expression can either be a fixed value or a formula. When creating a filter, only fixed values are authorized. During creation of a validation or computation rule, a formula can be created using the wizard.

Known limitation: The formula field does not validate input values, only the syntax and path are checked.

CHAPTER 93

Search

This chapter contains the following topics:

1. Overview
2. Search strategies for string fields
3. Search strategy for primary key fields
4. Search strategy for foreign key fields
5. Search strategy for associations
6. Excluding a field from search ('Void' indexing)
7. Excluding a field from search (via property)
8. Assigning a search strategy to a field

93.1 Overview

A search strategy defines how a field is indexed and queried. Any field is associated with a default search strategy, primarily based on its data type.

Search strategies are specified in the Data Model Assistant:

- when editing a field, its search strategies can be set in the 'Extensions' tab;
- at the data model level, custom search strategies can be specified, under 'Extensions > Search' element in the left pane;

See also *Quick Search* *[p 128]*

Value-labeling

Value-labeling is a global feature in EBX® to display user-friendly labels instead of raw values. For example, in the user interface, a foreign key field displays the label of the linked record, or a field based on a static enumeration displays the localized label associated with the raw value, as specified by the data model.

If a field supports value-labeling, the Quick search and the sort in the user interface usually apply on the displayed label, to preserve an intuitive user interface.

There are some exceptions, where raw value is still used by the quick search and the sort operation:

- Programmatic labels and programmatic enumeration constraints (a foreign key specifying a `TableRefDisplay` or whose display depends on a `UILabelRenderer` specified on the target table, or

a field constrained by a `ConstraintEnumeration`). It is recommended to use alternative solutions (display patterns and foreign keys).

- Enumeration constraint defined using another node (`<osd:enumeration osd:path=...`). It is recommended to use an alternative solution (a foreign key).

Obviously, if a field is displayed through a `UIWidget` (or a `UIBean`), to preserve an intuitive user interface, it is expected for the custom component to display the label (or the value, if this field does not enable value-labeling).

Limitations

The following fields are not optimized for search and other operators:

- Computed fields that:
  - Implement the `ValueFunction` API.
  - Use computation rules and scripts that depend on fields from outside of the container table (non-local dependencies).
- Inherited fields.

As a consequence, these are generally excluded from the quick search [p 128] and xpath search on table [p 442] via `osd:search('', 'pattern')`.

In the specific cases of history view or mapped tables, legacy search is used. This implies that the size of the table cannot be quickly estimated, and might not be presented in the UI. It also implies that quick search:

- considers all searchable fields (including computed fields with non-local dependency);
- behaves like a 'contains' (Lucene syntax cannot be used);
- does not support sort by relevancy;
- may perform poorly on tables with large volumes.

In the case of a node defining a display pattern, non sortable search strategies are prohibited for the default search template. Those are yet allowed in the other search templates.

93.2 Search strategies for string fields

Basic built-in strategies for strings

'**Text**'	The 'Text' search strategy is intended to contain multiple words, such as descriptions, texts or comments. This strategy supports full-text search and fuzzy search. Sorting, and some functions such as the ‘equals’ and 'starts-with' operators, are irrelevant and are not supported. This strategy is lightweight, consuming little disk space. See also Quick Search [p 128]
'**Code**'	The 'Code' search strategy is intended for codes and identifiers. Values are considered as one single token, allowing any kind of case-sensitive and case-insensitive filter. Full-text search is irrelevant and is replaced by a 'contains' by default. This can be modified to a ‘starts-with’ by defining a custom ‘Code’ strategy. For large volumes, a starts-with is preferable to achieve better performance.
'**Name**'	The 'Name' search strategy is intended for names and labels that contain only a few words. Besides having the same search capabilities as 'Text', 'Name' strategy also allows sort, and supports the same filters as 'Code'. This strategy has the most capabilities, but consumes more disk space. If the purpose of the field allows it, it is advised to choose the 'Text', 'Code' or 'Excluded from search' strategy, rather than this one.

Advanced strategies for strings

Advanced search strategies are meant to support searching using alternative fuzzy algorithms, and does not support sorting or filtering operations. It is advised to use them in a custom search template, and use a basic strategy with the default search template.

'**Levenshtein**'	The 'Levenshtein' strategy is intended for names and labels that contain a few words. This strategy only supports fuzzy search, based on an edit distance algorithm. The search syntax (+,-,...) is not supported. This strategy does not perform well on large volumes. The parameters of the strategy allow favorizing the performance, by setting the maximum edit distance to 1, and using an invariant prefix.
'**Soundex**'	The 'Soundex' strategy is intended for names and labels that contain a few words. This strategy only supports fuzzy search, based on the Soundex approximate phonetic algorithm. This algorithm is meant for english, and ignores digits.
'**Double Metaphone**'	The 'Double metaphone' strategy is intended for names and labels that contain a few words. This strategy only supports fuzzy search, based on the Double metaphone approximate phonetic algorithm. This algorithm is more modern than Soundex, and gives better results, especially for non-english languages. Digits are ignored.
'**NGram strategy**'	The 'NGram' strategy is intended for names and labels that contain a few words. This strategy only supports fuzzy search, based on a distance algorithm. This algorithm splits the values into smaller sequences called 'grams'. It performs better than the 'Levenshtein' search strategy, but consumes more disk space.

Default strategy for string fields

The 'Name' strategy is applied to string fields by default, except:

- If the field is part of the primary key, it is set by default to 'Code'.
- If the field is a foreign key, it is forced to 'Code' and cannot be changed.
- If the field has a built-in datatype extending `xs:string`, then it has a strategy relevant to its datatype; for instance `osd:text`, `xs:Name`, `osd:email`, `osd:html`, etc.

As the default strategy 'Name' can be irrelevant and consumes more disk space, the data model compilation reports warnings for fields with the 'Name' strategy set as default, so as to ensure that strategies are defined on purpose. We advise to choose the 'Text' strategy, when the length of the expected values is greater than 80, as a rough estimate. Long values (> 32766 bytes once encoded into UTF-8) will not be fully indexed with the 'Name' or 'Code' strategy. Quick search is not affected, but

sorting will consider only the first 1000 characters, and some operators ('equals' and 'ends-with', SQL DISTINCT and COUNT DISTINCT, ...) will not return the correct results.

Advanced custom strategies

Some strategies accept parameters, for example to define stop words, or a specific language. This is done by creating a record in the 'Search strategies' table of the 'Search' data model extension. The new parameterized strategy will be available for selection in the 'Extension' tab, for compatible fields.

Synonyms and stop words

It is possible to define stop words and synonyms lists in the 'Search' data model extension. Create a new record in the 'Stop words lists' or 'Synonyms lists' table and select the created list in the parameters tab of the 'Custom search strategies' table.

93.3 Search strategy for primary key fields

Primary key fields must have a sortable search strategy defined on the default search template. This excludes the 'Void' strategy for all data types, and the 'Text' strategy for strings. Do note that it is still possible to use non sortable search strategy for a primary key field, if it is defined within a search template other than the default search template.

93.4 Search strategy for foreign key fields

Foreign key fields have two levels of search:

- First, if applicable, the search is performed on each field of the displayed label of the foreign key. Each field strategy is inherited from the field in its target table. This first level is not always applicable, for instance when the search string cannot be converted to any of the target field data types.
- Secondly, when the first level of search cannot be applied, the search is performed on the string representation of the target primary key. Modifying the search strategy of a foreign key field in the 'Extension' tab in the Data Model Assistant only affects this second level of search. It can only be a 'Code' search strategy (built-in or customized).

See also `PrimaryKey.syntax`[API]

See also *Value-labeling* *[p 511]*

93.5 Search strategy for associations

In the case of associations, the search is performed on each field used in the label of the association records (each field strategy is inherited from the field in its target table). To be applicable, the target field must be optimized for search, and the search criterion must be convertible to its type.

93.6 Excluding a field from search ('Void' indexing)

The 'Excluded from search' (or `Void`) strategy deactivates indexing, making filter, search, or sort impossible. It is available for all data types, and is intended for fields that are never queried. Values can still be accessed through their record. Disabling the indexing reduces the disk space consumed and speeds up some operations like data import.

93.7 Excluding a field from search (via property)

It is also possible to exclude fields from the quick search tool using the property `osd:defaultView/hiddenInQuickSearch="true|false"`.

See Default view [p 73] for more information.

93.8 Assigning a search strategy to a field

A search strategy can be associated with a field, by means of a **search template** `SearchTemplate`[API]. This is done in the 'Extension' tab of the field, in the Data Model Assistant. Assigning multiple search strategies to a field requires registering additional search templates into a module. Only the addons EBX® Information Search and EBX® Match and merge are concerned by additional search templates.

Administration Guide

CHAPTER 94

Administration overview

The Administration section in TIBCO EBX® is the main point of entry for all administration tasks. In this overview are listed all the topics that an administrator needs to master. Click on your topic of interest in order to access the corresponding chapter or paragraph in the documentation.

This chapter contains the following topics:

1. Repository management
2. Disk space management
3. Data model
4. Data validation
5. Perspectives
6. Home page
7. Log configuration

94.1 Repository management

For storage optimization, it is recommended to maintain a repository (persistence RDBMS) to the necessary minimum. To this end, it is recommended to regularly perform a purge of snapshots and obsolete dataspaces and to consider using a backup file system.

See also Cleaning up dataspaces, snapshots, and history [p 666] and Deleting dataspaces, snapshots, and history [p 667].

It is also possible to archive files of the file system type in order to reduce the storage costs, see EBX® monitoring [p 664].

Administration tasks can be scheduled by means of the task scheduler, using built-in tasks, see Task scheduler [p 723].

Object cache

EBX® maintains an object cache in memory. The object cache size should be managed on a case by case basis according to specific needs and requirements (pre-load option and pre-validate on the reference dataspaces, points of reference, and monitoring), while continuously monitoring the repository health report.

See also *Memory management [p 615]*

Limitations

The following limitations apply to EBX®:The value for the G_DSP table's dsp_id column is limited to 2,147,483,648. This value increments with each dataspace creation and only decreases if the application restarts after a purge that removes the last created dataspaces. Up to 2,147,483,648 HomeKey instances can be created in a single JVM run. The value for the G_DST table's dst_id column is limited to 65,536. This value increments each time a dataset is created and persisted with a new dataset reference. A dataspace can have up to 8,192 datasets. For performance considerations, it is recommended not to exceed 32 datasets. A datamodel can declare up to 65,535 tables. For performance considerations, it is recommended not to exceed 256 tables.

Obsolete contents

Keeping obsolete contents in the repository can lead to a slow server startup and slow responsiveness of the interface. It is strongly recommended to delete obsolete content.

For example: datasets referring to deleted data models or undeployed add-on modules. See Managing TIBCO EBX® add-ons [p 605].

Workflow

Cleanup

The workflow history and associated execution data have to be cleaned up on a regular basis.

The workflow history stores information on completed workflows, their respective steps and contexts. This leads to an ever-growing database containing obsolete history and can thus lead to poor performance of the database if not purged periodically. See Workflow history [p 722] for more information.

Email configuration

It is required to configure workflow emails beforehand in order to be able to implement workflow email notifications. See Configuration [p 720] for more information.

94.2 Disk space management

Purge of logs

The log file size will vary according to the log level (and to the selected severity level) thus disk space needs to be accordingly managed.

EBX® main configuration file mechanism

An automatic purge is provided with EBX®, allowing to define how many days should log files be stored. After the defined period, log files are deleted.

Any customized management of the purge of logs (backup, archiving, etc.) is the administrator's responsibility.

See Custom 'ebxFile' appender [p 584] for more information.

XML audit trail (deprecated)

EBX® is provided with an XML audit trail manager, now disabled by default. Any customized management (including purge, backups, etc.) is the administrator's responsibility.

See Activating the legacy XML audit trail (deprecated) [p 581] and Legacy XML Audit trail (deprecated) [p 736] for more information.

94.3 Data model

Publication management

The management of publications of embedded data models [p 99]. See Data model administration [p 709] for more information on the management of these publications and the administration tasks that can be performed (delete, import and export).

Refresh data models

It is possible to update the data models that are using XML Schema documents not managed by EBX®. See Data model refresh tool [p 891] for more information.

94.4 Data validation

Resetting validation reports

Validation reports can be reset from the administration menu. It is possible to reset the validation reports of all the datasets contained in the repository or only those of the datasets selected in the tabular view. Therefore, the related datasets will be fully validated at the next explicit validation request.

See `Adaptation.resetValidationReport`[API] for more information.

94.5 Perspectives

EBX® offers extensive UI customization options. Simplified interfaces (Recommended perspectives) [p 684] dedicated to each profile accessing the system can be parameterized by the administrator. According to the profile of the user logging in, the interface will offer more or less options and menus. This allows for a streamlined work environment.

See Advanced perspective [p 672] for more information.

94.6 Home page

EBX® offers a customizable home page, which can be accessed through the 'Perspective' menu, or directly using the `/ebx-hub` URL.

In its current state, the links that are displayed in this page open their target in a new tab. This behavior may change in a future version of EBX®.

> **Note**
>
> The home page requires the `ebx-hub` war to be deployed. It won't be available if the latter is not deployed.

94.7 Log configuration

You can configure EBX® logs using Apache Log4j™ 2, the `ebx.properties` main configuration file, or a combination thereof.

EBX® log management relies heavily on Apache Log4j™ 2, which means logs can take advantage of the Log4j automatic configuration on initialization and automatic reconfiguration mechanisms. See Apache Log4j™ 2 configuration documentation and Configuring the EBX® logs [p 582] for more detailed information on these mechanisms.

The `ebx.properties` main configuration file centralizes EBX® configuration properties. However, Apache Log4j™ 2 automatic configuration on initialization has precedence over this mechanism. Additionally, the web application server start up process makes the application initialization order unpredictable. Due to these factors, log messages might not be generated according to the configuration specified in the `ebx.properties` file. See Configuring the EBX® logs [p 582] for more information.

Order and combination of log mechanisms

On web application server start up, the Apache Log4j™ 2 automatic configuration executes first. Then during the web application initialization phase, the `ebx.properties` main configuration file is processed and logs are reconfigured. Reconfiguration entails replacing the entire configuration.

If the Apache Log4j™ 2 automatic reconfiguration mechanism is used, then logs may be reconfigured based on the Log4j configuration context at any time after the first initialization (that is before, or after the EBX® main configuration file is processed). If the `ebx.properties` file is updated, then logs may be reconfigured based on the file content at any time after the first processing.

Since the mechanisms are not exclusive, you can set the initial configuration using the Apache Log4j™ 2 automatic configuration on initialization and then replace it with the one from the `ebx.properties` file.

> **Note**
>
> To bypass log reconfiguration, remove *ebx.log4j.rootCategory* and all *ebx.log4j.category.** properties from the `ebx.properties` file.

Installation & configuration

CHAPTER 95

Supported environments

This chapter contains the following topics:

1. Browsing environment
2. Supported application servers
3. Supported databases

95.1 Browsing environment

Supported web browsers

The TIBCO EBX® web interface supports the following browsers on computers only; it is not supported on mobile devices:

Microsoft Edge Chromium	As Microsoft Edge Chromium is updated frequently and it is not possible to deactivate automatic updates, Cloud Software Group, Inc. only tests and makes the best effort to support the latest version available.
Mozilla Firefox ESR	As Mozilla Firefox ESR is updated frequently and it is not possible to deactivate automatic updates, Cloud Software Group, Inc. only tests and makes the best effort to support the latest version available.
Google Chrome	As Google Chrome is updated frequently and it is not possible to deactivate automatic updates, Cloud Software Group, Inc. only tests and makes the best effort to support the latest version available.

Screen resolution

The minimum screen resolution for EBX® is 1024x768.

Refreshing pages

Browser page refresh is not supported by EBX®. When a page refresh is performed, the last user action is re-executed, and therefore could cause issues. It is thus imperative to use the action buttons and links offered by EBX® instead of refreshing the page.

'Previous' and 'Next' buttons

The 'previous' and 'next buttons of the browser are not supported by EBX®. When navigating through page history, an obsolete user action is re-executed, and therefore could cause issues. It is thus imperative to use the action buttons and links offered by EBX® rather than the browser buttons.

Zoom troubleshooting

Zooming in or out may cause some minor display issues (for example extra scrollbar or misalignment). Those issues can be fixed by refreshing the screen using the provided navigation links.

Browser configuration

The following features must be activated in the browser configuration, for the user interface to work properly:

- JavaScript
- Ajax
- Pop-ups
- Cookies

Attention

Avoid using any browser extensions or plug-ins, as they could interfere with the proper functioning of EBX®.

Limitations

Some browsers may have a limitation on the number of iframes that can be embedded. If this is the case, it limits to the number of items that can be pushed in the breadcrumb. Please check the browser documentation for more details.

95.2 Supported application servers

The information in this section provides details on the application servers compatible with EBX®. Supported versions are thoroughly tested prior to their approval and are actively tested to ensure continued compatibility. This same approach is used when adopting the latest technology. EBX® now supports Java 25; testing began even before this version was finalized as an LTS release. This testing yielded positive results. For example, a bug fix introduced as early as Java 23 impacted migration from EBX® 5 to EBX® 6. This issue has since been resolved. However, we were able to raise a warning to help prevent potential migration issues.

EBX® supports the following configurations:

- Java Runtime Environments (JRE) from: OpenJDK-based Long-Term Support (LTS) versions 25, 21, or 17, or WebSphere Liberty JDK 17 or 21. JRE 25 is preferred over 21 or 17. To ensure

that reported bugs or issues can be addressed, they must be reproducible on OpenJDK. Check with your JDK vendor if you are unsure as to whether your JDK is OpenJDK-based.

EBX® is compiled with JDK 17, and we recommend upgrading your JRE to the latest supported LTS version (see the note above regarding OpenJDK 21.0.9) to benefit from security and performance improvements. Checks should be done regularly to ensure you have the latest JRE updates.

EBX® support is based on JRE and application servers (e.g., Tomcat, JBoss) and does not rely on the operating system. Therefore, as long as the JRE and application server are supported on a given OS, EBX® supports that OS.

> **Note**
>
> The Unicode CLDR has changed with JRE versions and impacts locale-sensitive formats such as integer or decimal representations. Configure the search order of locale sensitive services using the `java.locale.providers` system property. You can use the value `COMPAT` to ensure compatibility with JRE 8. See java.util.spi.LocaleServiceProvider for more details.

- Any Servlet/JSP container that complies with Servlet 6.0 and above. For example, Tomcat 10.1.X, but only with the `unpackWARs` option set to `true`. See Tomcat Application Server and Java Virtual Machine configuration [p 558] for more information. Any Java Application Server that supports Java SE 25, 21 or 17 LTS and using Jakarta EE 10 or greater. For example, WebSphere Liberty 23 or higher for Jakarta EE 10, JBoss EAP 8.0.0 or higher. See Jakarta EE deployment [p 542].
- The application server must support the JSON Processing 1.1 (JSR 374), or allow the uses of the implementation embedded in the `ebx.jar` library. For example, Tomcat does not provide any library to support this specification (only the embedded one can be used), WebSphere Application Server Liberty and JBoss EAP allow including or excluding the available libraries.
- The application server must use UTF-8 encoding for HTTP query strings from EBX®. This can be set at the application server level.

  For example, on Tomcat, you can set the server to always use the UTF-8 encoding, by setting `URIEncoding` to 'UTF-8' on the `<Connector>` in the `server.xml` configuration file. Alternatively, you can instruct the server to use the encoding of the request body by setting the parameter `useBodyEncodingForURI` to 'true' in `server.xml`.

Attention

- Limitations apply regarding clustering and hot deployment/undeployment:

  Clustering: EBX® does not include a cache synchronization mechanism, thus it cannot be deployed into a cluster of active instances. See Technical architecture [p 660] for more information.

  Hot deployment/undeployment: EBX® does not support hot deployment/undeployment of web applications registered as EBX® modules, or of EBX® built-in web applications.

95.3 Supported databases

The EBX® repository supports the relational database management systems listed below, with the suitable JDBC drivers. It is important to follow the database vendor recommendations and update policies regarding the database itself, as well as the JDBC driver.

Oracle Database 19c, 23c or higher.	The distinction of `null` values bears certain limitations. On simple `xs:string` elements, Oracle does not support the distinction between empty strings and `null` values. See Empty string management [p 950] for more information. The user with which EBX® connects to the database requires the following privileges: • CREATE SESSION, • CREATE TABLE, • ALTER SESSION, • CREATE SEQUENCE, • A non-null quota on its default tablespace.
PostgreSQL 14 or higher.	The user with which EBX® connects to the database needs the CONNECT privilege on the database hosting the EBX® repository. Other than this, the default privileges on the public schema of this database are suitable. Also, see this limitation [p 477] regarding the evolution of datamodels in mapped modes.
Amazon Aurora PostgreSQL (compatible with PostgreSQL 14 or higher).	The comments in the above section for PostgreSQL apply.
Google Cloud SQL for PostgreSQL (compatible with PostgreSQL 14 or higher).	The comments in the above section for PostgreSQL apply.
Microsoft Azure Database for PostgreSQL (compatible with PostgreSQL 16 or higher).	The comments in the above section for PostgreSQL apply.
Microsoft SQL Server 2016 or higher.	When used with Microsoft SQL Server, EBX® uses the default database collation to compare and sort strings stored in the database. This applies to strings used in the data model definition, as well as data stored in history tables. The default database collation can be specified when the database is created. Otherwise, the collation of the database server is used. To avoid naming conflicts or unexpected

	behaviors, a case- and accent-sensitive collation must be used as the default database collation (the collation name is suffixed by "CS_AS" or the collation is binary). The default setting to enforce transaction isolation on SQL Server follows a pessimistic model. Rows are locked to prevent any read/write concurrent accesses. This may cause liveliness issues for mapped tables (history or relational). To avoid such issues, it is recommended to activate snapshot isolation on your SQL Server database. The user with which EBX® connects to the database requires the following privileges: • CONNECT, SELECT and CREATE TABLE on the database hosting the EBX® repository, • ALTER, CONTROL, UPDATE, INSERT, DELETE on its default schema.
Microsoft Azure SQL Database	EBX® has been qualified on Microsoft Azure SQL Database v12 (12.00.700), and is regularly tested to verify compatibility with the current version of the Azure database service. When used with Microsoft Azure SQL, EBX® uses the default database collation to compare and sort strings stored in the database. This applies to strings used in the data model definition, as well as data stored in history tables. The default database collation can be specified when the database is created. Otherwise, the database engine server collation is used. To avoid naming conflicts or unexpected behaviors, a case- and accent-sensitive collation must be used as the default database collation (the collation name is suffixed by "CS_AS" or the collation is binary). The user with which EBX® connects to the database requires the following privileges: • CONNECT, SELECT and CREATE TABLE on the database hosting the EBX® repository, • ALTER, CONTROL, UPDATE, INSERT, DELETE on its default schema.
H2 2.2.224 or higher.	H2 is not supported for production environments. The default H2 database settings do not allow consistent reads when records are modified.

For other relational databases, please contact the Support team at https://support.tibco.com.

Attention

In order to guarantee the integrity of the EBX® repository, it is strictly forbidden to perform direct modifications to the database (for example, using direct SQL writes).

See also

Repository administration *[p 660]*

Data source of the EBX® repository *[p 540]*

Configuring the EBX® repository *[p 574]*

CHAPTER 96

Disk requirements

This chapter contains the following topics:

1. Location and disk type
2. Anti-virus
3. Disk sizing

96.1 Location and disk type

Use the `ebx.repository.directory` property in the `ebx.properties` file to set the EBX® root directory location. This directory will contain the Lucene indexes and should be hosted on local, low latency hard disks. SSD disks are recommended. This direction applies to the repository and documentation indexes to prevent slow access and errors in index management. Using the local hard disk for indexes presents no risk of data loss, as EBX® is able to rebuild indexes from data stored in the relational database. However, the recovery operation can be time-consuming.

Attention
The EBX® root directory should never be shared by multiple EBX® instances.

You must avoid storing EBX® indexes over network attached volumes based on protocols, such as NFS, SMB, or SAMBA. The same restrictions apply to object-based storage (Amazon S3, Google Cloud Storage, Azure Blob) or file systems mounted over object storage (Amazon S3 file gateway). This may cause stability and performance issues; the more common symptoms are slow search speeds and running out of open files.

A high-performance Storage Area Network (SAN) can be an acceptable solution, especially if high performance technologies such as Fiber Channel (FC) are used.

On the cloud, EBX® has been tested with the following disks:

- Amazon Web Services (AWS): Amazon Elastic Block Store (EBS) general purpose SSD volumes (gp3)
- Microsoft Azure: Azure Standard SSD locally redundant storage (LRS)
- Google cloud: Google Balanced persistent disks (pd-balanced)

96.2 **Anti-virus**

Anti-virus software should not be used on the root directory as it can severely impact performance and can corrupt the index repository if a file is quarantined or removed. Exclusion by extension is not possible because the extensions used by Lucene can change at any time.

96.3 **Disk sizing**

As a rule of thumb for the disk size, plan for 10 times the space occupied by the table G_BLK in the relational database.

CHAPTER 97

Jakarta EE deployment

This chapter contains the following topics:

1. Introduction
2. Software components
3. Embedded third-party libraries
4. Required third-party libraries
5. Web applications
6. Deployment details
7. Installation notes

97.1 Introduction

This chapter details deployment specifications for TIBCO EBX® on a Java application server. For specific information regarding supported application servers and inherent limitations, see Supported environments. [p 524]

Refer to the Security Best Practices [p 870] for information on navigating secure deployments.

97.2 Software components

EBX® uses the following components:

- Library `ebx.jar`
- Embedded [p 534] and required [p 534] third-party Java libraries
- EBX® built-in web applications [p 538] and optional custom web applications [p 538]
- EBX® main configuration file [p 571]
- EBX® repository [p 660]
- Default user and roles directory [p 703], integrated within the EBX® repository, or a third-party system (LDAP, RDBMS) for the user authentication

See also *Supported environments [p 524]*

97.3 Embedded third-party libraries

To increase EBX® independence and interoperability, it embeds its own third-party libraries. Even if some of them have been modified, preventing conflicts, others must remain unchanged since they are official Java APIs.

The ones that can produce conflicts are:

- Apache Geronimo JSON
- Apache Geronimo JSON Bind
- Jakarta Activation API
- Jakarta Annotations API
- Jakarta JSON API
- Jakarta JSON Binding
- Jakarta Transaction API
- Jakarta WS RS
- Jakarta XML Bind
- Jakarta XML SOAP messaging API
- Jakarta XML WS API
- MicroProfile OpenAPI

For more information regarding the versions or the details of the Third-Party Library, please refer to the: `TIB_ebx_6.2.3_license.pdf`.

Since those libraries are already integrated, custom web applications should not include them anew, otherwise linkage errors can occur. Furthermore, they should not be deployed aside from the `ebx.jar` library for the same reasons.

97.4 Required third-party libraries

EBX® requires several third-party Java libraries. These libraries must be deployed and be accessible from the class-loader of `ebx.jar`. Depending on the application server and the Java runtime environment being used, these libraries may already be present or may need to be added manually.

Data compression library

Starting with EBX® version 6.2.3, deployment of a separate compression library is no longer required. The following 2 compression modes are supported:

- Native is the optimal, faster mode but only available if using Java 21 or newer on Linux/Windows, x86_64.
- A pure Java implementation is also available when using Java 17.

To see the compression type you are using, open `kernel.log` and look for:

- `Compression: Java`, or
- `Compression: Native`

If native compression is not being used, the reason will display in the logs.

If you find compression performance is an issue, use the following parameters in `ebx.properties` to adjust it:

- `ebx.compression.dictSize` controls the dictionary size and defaults to 16 KB. The recommended range is between 1 KB to 64 KB. However, you should be aware of the following tradeoff that occurs with a larger dictionary: The compression ratio (Logarithmic progression) is better, but compression is slower (linear increase).

  > **Note**
  >
  > You must set this value ahead of time. After EBX® creates a dictionary for a given table definitions, it does not create another one. The new value only applies to tables that don't have a dictionary, or to new table definitions.

- `ebx.compression.level` controls the compression level. Possible values are FAST, BALANCED, and HIGH (default). Lower the level to speed up compression, but expect a lower compression ratio. This setting immediately applies to the next compression operation. This setting has no noticeable impact on decompression.

Database drivers

The EBX® repository requires a database. Generally, the required driver is configured along with a data source, if one is used. Depending on the database defined in the main configuration file, one of

the following drivers is required. Keep in mind that, whichever database you use, the version of the JDBC client driver must be equal to or higher than the version of the database server.

H2	Version 2.2.224 validated. Note that H2 is not supported in production environments. https://www.h2database.com/ If the H2 repository has been built with version 1.0 driver, this migration to v2 process must be applied.
Oracle JDBC	Oracle database 23ai is validated on their latest patch set update. Determine the driver that should be used according to the database server version and the Java runtime environment version. Download the `ojdbc17.jar` certified library with JDK 17 or 21. Oracle database JDBC drivers download.
SQL Server JDBC	SQL Server 2016 and greater, with all corrective and maintenance patches applied, are validated. Remember to use an up-to-date JDBC driver, as some difficulties have been encountered with older versions. For example, include the `mssql-jdbc-13.2.1.jre11.jar` library, depending on the Java runtime environment version you use. Download Microsoft JDBC Driver 13.2.1 for SQL Server (zip).
PostgreSQL	PostgreSQL Java 8 validated Include the latest JDBC driver version 4.2 released for your database server and Java runtime environment. PostgreSQL JDBC drivers download.

See also

Data source of the EBX® repository *[p 540]*

Configuring the EBX® repository *[p 574]*

SMTP and emails

According to the web application server being used, the library `Jakarta Mail API` for email management may already be provided, or must be added manually.

EBX® requires a library that is compatible with version 2.1 of this API. See Activating and configuring SMTP and emails [p 586] for more information on the configuration.

To facilitate manual installation, the `jakarta.mail-2.0.5.jar` has been provided and placed under the `ebx.software/lib/lib-mail` directory.

See also *Jakarta Mail 2.1*

Secure Socket Layer (SSL)

SSL features activation, configuration, and management are tightly linked to the web application server used. Consequently, you should refer to the appropriate web application server documentation. For example, Tomcat SSL/TLS Configuration How-To.

See also *TIBCO EBX® main configuration file* *[p 571]*

Java Message Service (JMS)

When using JMS, version 3.1 or higher is required.

Depending on whether a Jakarta EE application server or a Servlet/Java Server Pages (JSP) implementation is being used, the library required is as follows:

- For an application server based on Jakarta EE (Enterprise Edition), the required JMS provider library is available by default. See https://jakarta.ee/specifications/platform/ for more information.
- For a Servlet/Java Server Pages (JSP) implementation using Java SE (Java Platform Standard Edition), for example Apache Tomcat, a JMS provider library such as Apache ActiveMQ may need to be added. See https://www.oracle.com/java/technologies/java-se-glance.html for more information.

> **Note**
>
> In EBX®, the supported JMS model is exclusively Point-to-Point (PTP). PTP systems allow working with queues of messages.

See also

TIBCO EBX® main configuration file *[p 571]*

Jakarta Messaging 3.1

97.5 **Web applications**

EBX® provides pre-packaged EARs that can be deployed directly if your company has no custom EBX® module web applications to add. If deploying custom web applications as EBX® modules, it is recommended to rebuild an EAR containing the custom modules packaged at the same level as the built-in web applications.

> **Attention**
>
> Web application deployment on `/` path context is no more supported. The path context must not be empty nor equals to `/`. Moreover, web applications deployment on paths of different depth is deprecated. Every web application path context must be set on the same path depth.

For more information, see the note on repackaging the EBX® EAR [p 543] at the end of this chapter.

EBX® built-in web applications

EBX® includes the following built-in web applications.

Web application name	Description	Required
`ebx`	EBX® entry point, which handles the initialization on start up. See Deployment details [p 539] for more information.	Yes
`ebx-root-1.0`	EBX® root web application. Any application that uses EBX® requires the root web application to be deployed.	Yes
`ebx-ui`	EBX® user interface web application.	Yes
`ebx-authentication`	EBX® authentication application.	Yes
`ebx-manager`	EBX® user interface web application.	Yes
`ebx-dma`	EBX® data model assistant, which helps with the creation of data models through the user interface. **Note:** The data model assistant requires the `ebx-manager` user interface web application to be deployed.	Yes
`ebx-dataservices`	EBX® data services web application. Data services allow external interactions with the EBX® repository using the SOAP operations [p 1079] and Web Services Description Language WSDL generation [p 1071] standards or using the Built-in RESTful services [p 1125]. **Note:** The EBX® web service generator requires the deployment of the `ebx-manager` user interface web application.	Yes
`ebx-ide`	EBX® Integrated Development Environment web application.	Yes
`ebx-assistant`	EBX® assistant web application.	Yes
`ebx-hub`	EBX® Hub web application.	No

EBX® add-on web applications

It is possible to extend the behavior of EBX® by deploying add-ons applications.

See also *Managing TIBCO EBX® add-ons [p 605]*

Custom web applications

It is possible to extend and customize the behavior of EBX® by deploying custom web applications which conform to the EBX® module requirements.

See also

Packaging TIBCO EBX® modules [p 879]

Declaring modules as undeployed [p 598]

97.6 Deployment details

Introduction

This section describes the various options available to deploy the 'ebx' web application. These options are available in its deployment descriptor (`WEB-INF/web.xml`) and are complemented by the properties defined in the main configuration file.

Attention

For JBoss application servers, any unused resources must be removed from the `WEB-INF/web.xml` deployment descriptor.

See also

TIBCO EBX® main configuration file [p 571]

Supported application servers [p 525]

User interface and web access

The web application 'ebx' (packaged as `ebx.war`) contains the servlet `FrontServlet`, which handles the initialization and serves as the sole user interface entry point for the EBX® web tools.

Configuring the deployment descriptor for 'FrontServlet'

In the file `WEB-INF/web.xml` of the web application 'ebx', the following elements must be configured for `FrontServlet`:

`/web-app/servlet/load-on-startup`	To ensure that `FrontServlet` initializes upon EBX® start up, the `web.xml` deployment descriptor must specify the element `<load-on-startup>1</load-on-startup>`.
`/web-app/servlet-mapping/url-pattern`	`FrontServlet` must be mapped to the path '`/`'.

Configuring the application server for 'FrontServlet'

`FrontServlet` must be authorized to access other contexts, such as `ServletContext`.

For example, on Tomcat, this configuration is performed using the attribute `crossContext` in the configuration file `server.xml`, as follows:

```
<Context path="/ebx" docBase="(...)" crossContext="true"/>
```

Data source of the EBX® repository

> **Note**
>
> If the EBX® main configuration specifies the property `ebx.persistence.url`, then the environment entry below will be ignored by EBX® runtime. This option is only provided for convenience; it is always recommended to use a fully-configurable datasource. In particular, the size of the connection pool must be set according to the number of concurrent users. See Configuring the EBX® repository [p 574] for more information on this property.

The JDBC datasource for EBX® is specified in the deployment descriptor `WEB-INF/web.xml` of the '`ebx`' web application as follows:

Reserved resource name	Default JNDI name	Description
`jdbc/EBX_REPOSITORY`	JBoss: `java:/EBX_REPOSITORY`	JDBC data source for EBX® Repository. Java type: `javax.sql.DataSource`

See also

Configuring the EBX® repository [p 574]

Rules for the database access and user privileges [p 661]

Mail sessions

> **Note**
>
> If the EBX® main configuration does not set `ebx.mail.activate` to 'true', or if it specifies the property `ebx.mail.smtp.host`, then the environment entry below will be ignored by EBX® runtime. See SMTP [p 586] in the EBX® main configuration properties for more information on these properties.

SMTP and email is declared in the deployment descriptor `WEB-INF/web.xml` of the '`ebx`' web application as follows:

Reserved resource name	Default JNDI name	Description
`mail/EBX_MAIL_SESSION`	JBoss: `java:/EBX_MAIL_SESSION`	Java Mail session used to send emails from EBX®. Java type: `jakarta.mail.Session`

JMS connection factory

> **Note**
>
> If the EBX® main configuration does not activate JMS through the property `ebx.jms.activate`, the environment entry below will be ignored by the EBX® runtime. See JMS [p 588] in the EBX® main configuration properties for more information on this property.

The JMS connection factory is declared in the deployment descriptor `WEB-INF/web.xml` of the '`ebx`' web application as follows:

Reserved resource name	Default JNDI name	Description	Required
`jms/EBX_JMSConnectionFactory`	JBoss: `java:/EBX_JMSConnectionFactory`	JMS connection factory used by EBX® to create connections with the JMS provider configured in the operational environment of the application server. Java type: `jakarta.jms.ConnectionFactory`	Yes

> **Note**
>
> For deployment on WildFly, JBoss application servers with JNDI capabilities, you must update `EBX.ear` for additional mappings of all required resource names to JNDI names.

JMS for data services

To configure data services to use JMS instead of the default HTTP, you must configure the JMS connection factory [p 541] and the following queues, declared in the `WEB-INF/web.xml` deployment descriptor of the '`ebx`' web application. This is the only method for configuring JMS for data services.

When a SOAP request is received, the SOAP response is optionally returned if the header field `JMSReplyTo` is defined. If so, the fields `JMSCorrelationID` and `JMSType` are retained.

See JMS [p 588] for more information on the associated EBX® main configuration properties.

> **Note**
>
> If the EBX® main configuration does not activate JMS through the property `ebx.jms.activate`, then the environment entries below will be ignored by EBX® runtime. See JMS [p 588] in the EBX® main configuration properties for more information on this property.

Reserved resource name	Default JNDI name	Description	Required
`jms/EBX_QueueIn`	JBoss: `java:/jms/EBX_QueueIn`	JMS queue for incoming SOAP requests sent to EBX® by other applications. Java type: `jakarta.jms.Queue`	No
`jms/EBX_QueueFailure`	JBoss: `java:/jms/EBX_QueueFailure`	JMS queue for failures. It contains incoming SOAP requests for which an error has occurred. This allows replaying these messages if necessary. Java type: `jakarta.jms.Queue` **Note:** For this property to be read, the main configuration must also activate the queue for failures through the property `ebx.jms.activate.queueFailure`. See JMS [p 588] in the EBX® main configuration properties for more information on these properties.	No

JAR files scanner

To speed up the web applications server startup, the JAR files scanner configuration should be modified to exclude, at least, the `ebx.jar` and `ebx-addons.jar` libraries.

For example, on Tomcat, this should be performed in the `tomcat.util.scan.StandardJarScanFilter.jarsToSkip` property from the `catalina.properties` file.

97.7 Installation notes

EBX® can be deployed on any Jakarta EE 10 or greater application server that supports Servlet 6.0 and above. The following documentation on deployment and installation notes are available:

- Installation notes for Tomcat 10.1.X [p 557]
- Installation notes for JBoss EAP 8.0.X [p 545]

- Installation notes for WebSphere Application Server Liberty 23.X [p 551]

Attention

- The EBX® installation notes on Jakarta EE 10 application servers do not replace the native documentation for each application server.
- These are *not* general installation recommendations, as the installation process is determined by architectural decisions, such as the technical environment, application mutualization, delivery process, and organizational decisions.
- In these examples, no additional EBX® modules are deployed. To deploy custom or add-on modules, the best practice is to rebuild an EAR with the module as a web application at the same level as the other EBX® modules. The web application must declare its class path dependency as specified by the Java™ 2 Platform Enterprise Edition Specification, v1.4:

  J2EE.8.2 Optional Package Support

  (...)

  A JAR format file (such as a JAR file, WAR file, or RAR file) can reference a JAR file by naming the referenced JAR file in a Class-Path header in the Manifest file of the referencing JAR file. The referenced JAR file is named using a URL relative to the URL of the referencing JAR file. The Manifest file is named META-INF/MANIFEST.MF in the JAR file. The Class-Path entry in the Manifest file is of the form:

  ```
  Class-Path: list-of-jar-files-separated-by-spaces
  ```

  In an "industrialized" process, it is strongly recommended to develop a script that automatically builds the EAR, with the custom EBX® modules, the EBX® web applications, as well as all the required shared libraries.
- In order to avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar` or other libraries in the class-loading system.

CHAPTER 98

Installation notes for JBoss EAP 8.0.X

This chapter contains the following topics:

1. Overview
2. Requirements
3. JBoss Application Server installation
4. EBX® home directory configuration
5. JBoss Application Server and Java Virtual Machine configuration
6. JNDI entries configuration
7. Data source and JDBC provider configuration
8. EBX.ear application update
9. EBX.ear application deployment
10. EBX® application start

98.1 Overview

Attention

- This chapter describes a *quick installation example* of TIBCO EBX® on JBoss Application Server.
- It does not replace the documentation of this application server.
- They are *not* general installation recommendations, as the installation process is determined by architectural decisions, such as the technical environment, application mutualization, delivery process, and organizational decisions.
- The complete description of the components needed by EBX® is given in chapter Jakarta EE deployment [p 533].
- To avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar`, or other libraries in the class-loading system.
- Refer to the Security Best Practices [p 870] for information on navigating secure deployments.

98.2 Requirements

- Java SE 17 LTS
- JBoss Application Server EAP 8.0
- Database and JDBC driver
- EBX® CD
- EBX® license key
- No CDI features in EBX®'s additional modules (since CDI will be automatically disable)

See also *Supported environments [p 524]*

98.3 JBoss Application Server installation

This quick installation example is performed for an unix operating system.

1. Download JBoss EAP 8.0.X `Installer` jar version from:
   https://developers.redhat.com/products/eap/download/
2. Run the `Installer` using *java -jar* command line.
   For further installation details, refer to the documentation.
3. Perform a standard installation:
   1. Select the language and click 'OK',
   2. Accept the License and click 'Next',
   3. Choose the installation path and click 'Next',
   4. Keep the 'Component Selection' as it is and click 'Next',
   5. Enter 'Admin username', 'Admin password' and click 'Next',

6. On 'Installation Overview' click 'Next',
7. On 'Component Installation' click 'Next',
8. On 'Configure Runtime Environment' leave the selection as it is and click 'Next',
9. When 'Processing finished' appear, click 'Next',
10. Generate 'installation script and properties file' in the JBoss EAP 8.0 installation root directory,
11. Click 'done'.

98.4 EBX® home directory configuration

1. Create the *EBX_HOME* directory, for example `/opt/ebx/home`.
2. Copy from the *EBX® CD*, the `ebx.software/files/ebx.properties` file to *EBX_HOME*. In our example, we will have the following file:

   `/opt/ebx/home/ebx.properties`.
3. Create an `ebx-license.properties` file in *EBX_HOME* as shown below:

   `/opt/ebx/home/ebx-license.properties`

   The content of the file:

```
######################################################################
## EBX® License Key
##    Ensure the file is writable by EBX®
##    to enable updates using the UI license key wizard process.
######################################################################
ebx.license=<your_license_key>
```

4. If needed, edit the `ebx.properties` file to override the default database. By default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and the `h2.standalone` one must be commented.

98.5 JBoss Application Server and Java Virtual Machine configuration

1. Open the `standalone.conf` configuration file, placed in `<JBOSS_HOME>/bin` (or `jboss-eap.conf` file placed in `<JBOSS_HOME>/bin/init.d` for running the server as a service).
2. Add 'ebx.properties' and 'ebx.home' properties to the 'JAVA_OPTS' environment variable respectively set with `ebx.properties` file's path and *EBX_HOME* directory's path.
3. Set the 'JBOSS_MODULES_SYSTEM_PKGS' environment variable like the following:

```
JBOSS_MODULES_SYSTEM_PKGS="org.jboss.byteman,net.jpountz"
```

4. Open the `standalone.sh` script file, placed in `<JBOSS_HOME>/bin`.
5. Replace the launch command options for foreground and background executions like the following:

```
if [ "x$LAUNCH_JBOSS_IN_BACKGROUND" = "x" ]; then
   # Execute the JVM in the foreground
   eval \"$JAVA\" -D\"[Standalone]\" $JAVA_OPTS \
      -cp "$CLASSPATH" \
      \"-Dorg.jboss.boot.log.file="$JBOSS_LOG_DIR"/server.log\" \
      \"-Dlogging.configuration=file:"$JBOSS_CONFIG_DIR"/logging.properties\" \
      org.jboss.modules.Main \
      $MODULE_OPTS \
```

```
      -mp \""${JBOSS_MODULEPATH}"\" \
      org.jboss.as.standalone \
      -Djboss.home.dir=\""$JBOSS_HOME"\" \
      -Djboss.server.base.dir=\""$JBOSS_BASE_DIR"\" \
      "$SERVER_OPTS"
      JBOSS_STATUS=$?

else
   # Execute the JVM in the background
   eval \"$JAVA\" -D\"[Standalone]\" $JAVA_OPTS \
      -cp "$CLASSPATH" \
      \"-Dorg.jboss.boot.log.file="$JBOSS_LOG_DIR"/server.log\" \
      \"-Dlogging.configuration=file:"$JBOSS_CONFIG_DIR"/logging.properties\" \
      org.jboss.modules.Main \
      $MODULE_OPTS \
      -mp \""${JBOSS_MODULEPATH}"\" \
      org.jboss.as.standalone \
      -Djboss.home.dir=\""$JBOSS_HOME"\" \
      -Djboss.server.base.dir=\""$JBOSS_BASE_DIR"\" \
      "$SERVER_OPTS" "&"
...
fi
```

98.6 **JNDI entries configuration**

1. Open the `standalone-full.xml` file placed in `<JBOSS_HOME>/standalone/configuration`.
2. Add, at least, the following lines to the `server` tag in `messaging-activemq` subsystem:

```
<connection-factory
    name="jms/EBX_JMSConnectionFactory"
    entries="java:/EBX_JMSConnectionFactory"
    connectors="To Be Defined"/>
<jms-queue
    name="jms/EBX_D3ReplyQueue"
    entries="java:/jms/EBX_D3ReplyQueue"
    durable="true"/>
<jms-queue
    name="jms/EBX_QueueIn"
    entries="java:/jms/EBX_QueueIn"
    durable="true"/>
<jms-queue
    name="jms/EBX_QueueFailure"
    entries="java:/jms/EBX_QueueFailure"
    durable="true"/>
<jms-queue
    name="jms/EBX_D3MasterQueue"
    entries="java:/jms/EBX_D3MasterQueue"
    durable="true"/>
<jms-queue
    name="jms/EBX_D3ArchiveQueue"
    entries="java:/jms/EBX_D3ArchiveQueue"
    durable="true"/>
<jms-queue
    name="jms/EBX_D3CommunicationQueue"
    entries="java:/jms/EBX_D3CommunicationQueue"
    durable="true"/>
```

   Caution: the `connectors` attribute value, from the `connection-factory` element, has to be defined. Since the kind of connectors is strongly reliant on the environment infrastructure, a default configuration can not be provided.

   See configuring messaging old documentation, for more information.

3. Add, at least, the following line to `mail` subsystem:

```
<mail-session name="mail" debug="false" jndi-name="java:/EBX_MAIL_SESSION"/>
```

98.7 **Data source and JDBC provider configuration**

1. After the launch of the JBoss Server, run the management CLI without the use of '--connect' or '-c' argument.

2. Use the 'module add' management CLI command to add the new core module. Sample for PostgreSQL configuration:

```
module add \
  --name=org.postgresql \
  --resources=<PATH_TO_JDBC_JAR> \
  --dependencies=wildflyee.api
```

3. Use the 'connect' management CLI command to connect to the running instance.
4. Register the JDBC driver. When running in a managed domain, ensure to precede the command with '/profile=<PROFILE_NAME>'. Sample for PostgreSQL configuration:

```
/subsystem=\
  datasources/jdbc-driver=\
    postgresql:add(\
      driver-name=postgresql,\
      driver-module-name=org.postgresql,\
      driver-xa-datasource-class-name=org.postgresql.xa.PGXADataSource\
    )
```

5. Define the data source using the 'data-source add' command, specifying the appropriate argument values. Sample for PostgreSQL configuration:

```
data-source add \
  --name=jdbc/EBX_REPOSITORY \
  --jndi-name=java:/EBX_REPOSITORY \
  --driver-name=postgresql \
  --connection-url=jdbc:postgresql://<SERVER_NAME>:<PORT>/<DATABASE_NAME> \
  --user-name=<PERSISTENCE_USER> \
  --password=<PERSISTENCE_PASSWORD>
```

> **Note**
>
> The data source can also be specified through the EBX® main configuration file. See Data source of the EBX® repository [p 540] for more information.

98.8 EBX.ear application update

1. Copy from the *EBX® CD*, the `ebx.software/webapps/ear-packaging/EBX.ear` file to your working directory.
2. Uncompress the ear archive to add the application's specific required third-party libraries and additional web modules.

   Mail: see SMTP and emails [p 536] for more information.

   SSL: see Secure Socket Layer (SSL) [p 537] for more information.

   JMS: see Java Message Service (JMS) [p 537] for more information.
3. Update the `/META-INF/application.xml` and `/META-INF/jboss-deployment-structure.xml` files according to the added additional web modules.
4. Compress anew the ear archive.

98.9 EBX.ear application deployment

1. Copy EBX.ear into the *`JBOSS_HOME`*`/standalone/deployments` directory.

98.10 **EBX® application start**

1. After the launch of the JBoss Application Server, with the `<JBOSS_HOME>/bin/standalone.sh -c standalone-full.xml` command line or through the service command, run the EBX® web application by entering the following URL in the browser: http://localhost:8080/ebx/.
2. At first launch, EBX® Wizard [p 603] helps to configure the default properties of the initial repository.

CHAPTER 99

Installation notes for WebSphere Application Server Liberty 23.X

This chapter contains the following topics:

1. Overview
2. Requirements
3. WebSphere Application Server Liberty installation
4. EBX® home directory configuration
5. EBX® and third-party libraries deployment
6. JNDI entries configuration
7. Data source and JDBC provider configuration
8. Java Virtual Machine configuration
9. EBX® application deployment
10. EBX® application start

99.1 Overview

Attention

- This chapter describes a *quick installation example* of TIBCO EBX® on WebSphere Application Server Liberty.
- It does not replace the documentation of this application server.
- They are *not* general installation recommendations, as the installation process is determined by architectural decisions, such as the technical environment, application mutualization, delivery process, and organizational decisions.
- The complete description of the components needed by EBX® is given in the chapter Jakarta EE deployment [p 533].
- To avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar`, or other libraries in the class-loading system.
- The description below uses the variable name <WLP_HOME> to refer to the WebSphere Application Server Liberty installation directory, and from which relative paths are resolved.
- Refer to the Security Best Practices [p 870] for information on navigating secure deployments.

99.2 Requirements

- Java SE 17 or 21 LTS. It is recommended using IBM Semeru, or Eclipse Temurin Java distributions.
- WebSphere Application Server Liberty 23.0.0.12 or higher for Jakarta EE 10
- Database and JDBC driver
- EBX® CD
- EBX® license key
- No CDI features in EBX®'s additional modules (since CDI will be automatically disabled)

See also *Supported environments* *[p 524]*

99.3 WebSphere Application Server Liberty installation

This quick installation example is performed for a Linux operating system.

1. Download WebSphere Application Server Liberty 23.X latest version (zip distribution) from: https://www.ibm.com/support/pages/230012-websphere-application-server-liberty-230012 .
   The following example uses *Liberty with Jakarta EE 10 Web Profile* zip distribution.
2. Extract the content of this package to a directory on your local machine.
3. Create a new application server (example: 'ebxServer') by running the command:
   ```
   <WLP_HOME>/wlp/bin/server create ebxServer
   ```
4. Edit `<WLP_HOME>/usr/servers/<SERVER_NAME>/server.xml`file. In our example, <SERVER_NAME> is 'ebxServer'.

- Replace the `<feature>webProfile-10.0</feature>` tag with, at least, the following lines:

```
<feature>mail-2.1</feature>
<feature>jdbc-4.2</feature>
<feature>jndi-1.0</feature>
<feature>servlet-6.0</feature>
```

- Add the following line into the `<server>` tag:

```
<webContainer deferServletLoad="false"  allowQueryParamWithNoEqual="true"/>
```

> **Note**
>
> The `deferServletLoad` attribute must be set to `false` since EBX® requires servlets to be initialized on application server startup. By default, WebSphere Application Server Liberty defers servlets loading until a request is received for the associated web application. See Specifying when servlets are loaded and initialized for more information.

> **Note**
>
> The `allowQueryParamWithNoEqual` attribute must be set to `true` since, according to the Servlet API specification, a value of a request parameter must be returned as a `String`, or `null` if the parameter does not exist. Thus, EBX® expects a non `null` value for an existing request parameter.

99.4 EBX® home directory configuration

1. Create the *EBX_HOME* directory, for example `/opt/ebx/home`.
2. Copy from the *EBX® CD*, the `ebx.software/files/ebx.properties` file to *EBX_HOME*. In our example, we will have the following file:

   `/opt/ebx/home/ebx.properties`.
3. Create an `ebx-license.properties` file in *EBX_HOME* as shown below:

   `/opt/ebx/home/ebx-license.properties`

   The content of the file:

```
######################################################################
## EBX® License Key
##    Ensure the file is writable by EBX®
##    to enable updates using the UI license key wizard process.
######################################################################
ebx.license=<your_license_key>
```

4. If needed, edit the `ebx.properties` file to override the default database. By default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and the `h2.standalone` one must be commented.

99.5 EBX® and third-party libraries deployment

1. Create the *EBX_LIB* directory. For example, `/opt/ebx/home/lib`.
2. Copy third-party libraries from the *EBX® CD*, or from other sources, to the *EBX_LIB* directory. In our example and for a PostgreSQL database, we will get:

   `postgresql-X.X.X-driver.jar`, coming from another source than the *EBX® CD*.

The complete description of these components is given in the chapter Jakarta EE deployment [p 533]. If those components are already deployed on the class-loading system, they do not have to be duplicated (for example, Jakarta Mail API and runtime implementation support are provided when the `mail-2.1` feature is activated on this server).

99.6 JNDI entries configuration

1. For *JMS* configuration on WebSphere Application Server Liberty, see Configuring JMS connection factories.
2. For *Java Mail* configuration on WebSphere Application Server Liberty, see Administering JavaMail on Liberty.

99.7 Data source and JDBC provider configuration

1. To create the JDBC driver library, edit the `<WLP_HOME>/usr/servers/<SERVER_NAME>/server.xml` file by adding the following lines into the `<server>` tag:

```
<library id="jdbcDriver">
  <file name="${EBX_LIB}/To be completed"/>
</library>
```

Caution: the `name` attribute value, from the `file` tag, has to be completed. In our example, it should be `${EBX_LIB}/postgresql-X.X.X-driver.jar`.

2. To create the JDBC data source, edit the same file and same XML tag by adding the following lines:

```
<dataSource id="ebxRepository" jndiName="jdbc/EBX_REPOSITORY">
  <jdbcDriver libraryRef="jdbcDriver"/>
  <properties.toBeCompleted/>
</dataSource>
```

Caution: the `properties` tag, has to be completed. Since the kind of tag is strongly reliant on the JDBC driver used, a default configuration can not be provided. For a PostgreSQL driver, it may looks like:

```
<properties.postgresql databaseName="EBXDB" serverName="localhost" portNumber="5432" user="ebx"
 password="ebx"/>
```

> **Note**
>
> To hide the password for a JDBC connection in the `server.xml` file of WebSphere Application Server Liberty, refer to Authentication Data (authData), containerAuthData from Data Source (dataSource) and Securing Liberty and its applications.

> **Note**
>
> The data source can also be specified through the EBX® main configuration file. See Data source of the EBX® repository [p 540] for more information.

99.8 Java Virtual Machine configuration

1. Set server's environment variables by adding the following lines into `<WLP_HOME>/usr/servers/<SERVER_NAME>/server.env` file:

```
EBX_HOME="<path_to_the_directory_ebx_home>"
EBX_LIB="<path_to_the_directory_ebx_lib>"
JAVA_HOME="<path_to_the_java_home>"
```

2. Set Java options by creating a new `jvm.options` file under the `<WLP_HOME>/usr/servers/<SERVER_NAME>/` directory. This file will hold, at least, the following lines:

```
-Debx.home="${EBX_HOME}"
-Debx.properties="${EBX_HOME}/ebx.properties"
```

99.9 EBX® application deployment

1. Create the `<EBX_HOME>/ear/` directory.
2. Copy from the *EBX® CD*, the `ebx.software/webapps/ear-packaging/EBX.ear` to the `<EBX_HOME>/ear/` directory. In our example, we will get:

   `/opt/ebx/home/ear/EBX.ear`

99.10 EBX® application start

1. After the launch of the WebSphere Application Server Liberty, with the `<WLP_HOME>/wlp/bin/server start ebxServer` command line, run the EBX® web application by entering the following URL in the browser:

   http://localhost:9080/ebx/

2. At first launch, EBX® Wizard [p 603] helps to configure the default properties of the initial repository.

CHAPTER 100

Installation notes for Tomcat 10.1.X

This chapter contains the following topics:

1. Overview
2. Requirements
3. Tomcat Application Server installation
4. EBX® home directory configuration
5. Tomcat Application Server and Java Virtual Machine configuration
6. EBX® and third-party libraries deployment
7. EBX® web applications deployment
8. EBX® application start

100.1 Overview

Attention

- This chapter describes a *quick installation example* of TIBCO EBX® on Tomcat Application Server.
- It does not replace the documentation of this application server.
- They are *not* general installation recommendations, as the installation process is determined by architectural decisions, such as the technical environment, application mutualization, delivery process, and organizational decisions.
- The complete description of the components needed by EBX® is given in chapter Jakarta EE deployment [p 533].
- To avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar`, or other libraries in the class-loading system.
- The description below uses the variable name `$CATALINA_HOME` to refer to the Tomcat installation directory, and from which most relative paths are resolved. However, if the `$CATALINA_BASE` directory has been set for a multiple instances configuration, it should be used for each of these references.
- Refer to the Security Best Practices [p 870] for information on navigating secure deployments.

100.2 Requirements

- Java SE 17 LTS
- Apache Tomcat 10.1.X. EBX® version 6.2.0 and later is not compatible with Tomcat 11.
- Database and JDBC driver
- EBX® CD
- EBX® license key

See also *Supported environments* *[p 524]*

100.3 Tomcat Application Server installation

1. Download Tomcat 10.1.X core binary distribution from:
   https://tomcat.apache.org/download-10.cgi ,
2. Run the installer or extract the archive and perform a standard installation with default options.

100.4 EBX® home directory configuration

1. Create *EBX_HOME* directory, for example `C:\EBX\home`, or `/home/ebx`,
2. Copy from *EBX® CD* the `ebx.software/files/ebx.properties` file to *EBX_HOME*. In our example, we will have the following file:

   `C:\EBX\home\ebx.properties`, or `/home/ebx/ebx.properties`,
3. Create an `ebx-license.properties` file in *EBX_HOME* as shown below:

   `C:\EBX\home\ebx-license.properties`, or `/home/ebx/ebx-license.properties`

   The content of the file:

```
#################################################################
## EBX® License Key
##    Ensure the file is writable by EBX®
##    to enable updates using the UI license key wizard process.
#################################################################
ebx.license=<your_license_key>
```

4. If needed, edit the `ebx.properties` file to override the default database. By default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and the `h2.standalone` one must be commented.

100.5 Tomcat Application Server and Java Virtual Machine configuration

1. Modify `$CATALINA_HOME/conf/server.xml` (or `$CATALINA_BASE/conf/server.xml`) file.
   1. Add the following attributes to the `Connector` element:
      - `encodedSolidusHandling="passthrough"`
      - `allowBackslash="true"`

2. Set the `unpackWARs` option, of the `<Host>` element, to `true`.

   > **Note**
   >
   > When `unpackWARs` is *not* set to `true`, TIBCO EBX® cannot scan web application classes and is prevented from proper startup. Only `true` is supported for this option.

3. Add the following line to the `<Host>` element:

   ```
   <Context path="/ebx" crossContext="true" docBase="ebx.war"/>
   ```

   In our example, we will have:

   ```
   <Connector encodedSolidusHandling="passthrough" allowBackslash="true" .../>
   <Host unpackWARs="true" name=...>
   ... ...
   <Context path="/ebx" crossContext="true" docBase="ebx.war"/>
   ... ...
   </Host>
   ```

2. Modify the `$CATALINA_HOME/conf/catalina.properties` (or `$CATALINA_BASE/conf/catalina.properties`) file by adding the following lines to the `tomcat.util.scan.StandardJarScanFilter.jarsToSkip` property:

   ```
   ebx.jar,\
   ebx-addons.jar,\
   ```

3. Configure the Java Virtual Machine properties,

   - For Windows' Command Prompt launch;

     Set the environment variables by creating a `setenv.bat` file either into `$CATALINA_HOME\bin` or `$CATALINA_BASE\bin`. This file will hold, at least, the following lines:

     ```
     set EBX_HOME=<path_to_the_directory_ebx_home>
     set EBX_OPTS=-Debx.home="%EBX_HOME%" -Debx.properties="%EBX_HOME%\ebx.properties"
     set JAVA_OPTS=%EBX_OPTS% %JAVA_OPTS%
     ```

     Where <$CATALINA_HOME_or_$CATALINA_BASE> must be replaced by `%CATALINA_HOME%` or `%CATALINA_BASE%` if they have been configured. Otherwise this piece of text must be replaced by the Tomcat installation directory's path.

   - For Windows users that have installed Tomcat as a service;

     Set Java options through the Tomcat service manager GUI (Java tab).

     Be sure to set options on separate lines in the *Java Options* field of the GUI:

     ```
     -Debx.home="<path_to_the_directory_ebx_home>"
     -Debx.properties="<path_to_the_directory_ebx_home>\ebx.properties"
     ```

     Where <$CATALINA_HOME_or_$CATALINA_BASE> must be replaced by `%CATALINA_HOME%` or `%CATALINA_BASE%` if they have been configured. Otherwise this piece of text must be replaced by the Tomcat installation directory's path.

   - For Unix shell launch;

     Set the environment variables by creating a `setenv.sh` file either into `$CATALINA_HOME/bin` or `$CATALINA_BASE/bin`. This file will hold, at least, the following lines:

     ```
     EBX_HOME="<path_to_the_directory_ebx_home>"
     ```

```
EBX_OPTS="-Debx.home=${EBX_HOME} -Debx.properties=${EBX_HOME}/ebx.properties"
export JAVA_OPTS="${EBX_OPTS} ${JAVA_OPTS}"
```

Where <$CATALINA_HOME_or_$CATALINA_BASE> must be replaced by `${CATALINA_HOME}` or `${CATALINA_BASE}` if they have been configured. Otherwise this piece of text must be replaced by the Tomcat installation directory's path.

Caution: Accounts used to launch EBX® must have create/update/delete rights on *EBX_HOME* directory.

> **Note**
>
> <path_to_the_directory_ebx_home> is the directory where we copied `ebx.properties`. In our example, it is `C:\EBX\home`, or `/home/ebx`.

100.6 EBX® and third-party libraries deployment

1. Copy third-party libraries from the *EBX® CD* to `$CATALINA_HOME/lib/` (or `$CATALINA_BASE/lib/`) directory. In our example, we will have:

   `$CATALINA_HOME/lib/jakarta.mail-2.0.5.jar` coming from `ebx.software/lib/lib-mail` directory.

   `$CATALINA_HOME/lib/h2-2.2.224.jar` (default persistence factory) coming from `ebx.software/lib/lib-h2` directory.

   The exact description of these components is given in chapter Software components [p 533]. Obviously, if those components are already deployed on the class-loading system, they do not have to be duplicated.

2. Copy from *EBX® CD* the `ebx.software/lib/ebx.jar` file to `$CATALINA_HOME/lib/` (or `$CATALINA_BASE/lib/`) directory. In our example, we will have:

   `$CATALINA_HOME/lib/ebx.jar`

100.7 EBX® web applications deployment

1. Copy from the *EBX® CD* the war files in `ebx.software/webapps/wars-packaging` to the `$CATALINA_HOME/webapps/` (or `$CATALINA_BASE/webapps/`) directory. In our example, we will have:

   `$CATALINA_HOME/webapps/ebx.war`: Initialization servlet for EBX® applications,

   `$CATALINA_HOME/webapps/ebx-assistant.war`: AI assistant web application,

   `$CATALINA_HOME/webapps/ebx-authentication.war`: Central authentication application,

   `$CATALINA_HOME/webapps/ebx-dataservices.war`: Data Services web application,

   `$CATALINA_HOME/webapps/ebx-dma.war`: Data Model Assistant web application,

   `$CATALINA_HOME/webapps/ebx-hub.war`: Hub web application,

   `$CATALINA_HOME/webapps/ebx-manager.war`: Master Data Management web application,

   `$CATALINA_HOME/webapps/ebx-root-1.0.war`: Provides a common default module for data models,

   `$CATALINA_HOME/webapps/ebx-ui.war`: User Interface web application,

   `$CATALINA_HOME/webapps/ebx-ide.war`: Integrated Development Environment web application,

100.8 **EBX® application start**

1. After Tomcat launch, run EBX® web application by entering the following URL in the browser: http://localhost:8080/ebx/
2. At first launch, EBX® Wizard [p 603] helps to configure the default properties of the initial repository.

CHAPTER 101

Installation notes for EBX® Container Edition

This chapter contains the following topics:

1. Overview
2. Required EBX® add-ons
3. EBX® Metadata Classifier database
4. EBX® Metadata Agent
5. Verification
6. Limitations

101.1 Overview

The EBX® Metadata Management application is available as part of TIBCO EBX® Container Edition.

101.2 Required EBX® add-ons

When selecting **Metadata** from the interactive, or the batch installer, the following EBX® add-ons will be automatically installed in the Docker image:

- TIBCO EBX® Digital Asset Manager Add-on (DAMA)
- TIBCO EBX® Data Exchange Add-on (DINT)
- TIBCO EBX® Data Model and Data Visualization Add-on (DMDV)
- TIBCO EBX® Insight Add-on (DPRA)
- TIBCO EBX® Information Search Add-on (TESE)

> **Note**
>
> See the installer instructions for Building the image [p 628] and including the EBX® Metadata Management application.

101.3 EBX® Metadata Classifier database

Requirements

The Classifier database must be one of the following: SQLite, PostgreSQL, Microsoft SQL Server or Oracle Database (via ORM SQLAlchemy).

See Supported databases [p 527] for more information.

Customizing and running

By default, the container uses an embedded SQLite database. To use PostgreSQL or another database, add the corresponding driver to your custom Docker image and run the image with the required environment variables.

See Adding a driver for the Metadata Classifier database [p 657] and Metadata classifier database connectivity [p 639] for more information.

101.4 EBX® Metadata Agent

See Installing the TIBCO® Metadata Agent [p 815] for more information.

101.5 Verification

After the Docker container starts successfully, use the following to verify that the EBX® Metadata Management application has been installed correctly:

- Check the log for the following message: `Module ebx-metadata repository initialization complete.`
- In the EBX® UI check whether the list of perspectives in the main toolbar includes **Metadata**. If it does, it indicates that installation was successful.

See Accessing the metadata perspective [p 259] for more information.

101.6 Limitations

Installation of the EBX® Metadata Classifier out the Container is not supported.

CHAPTER **102**

Installation notes for on-prem Tomcat 10.1.X

This chapter contains the following topics:

1. Overview
2. EBX® Metadata Classifier and Classifier database
3. EBX® Metadata Management module
4. EBX® Metadata Agent
5. Final verification
6. Limitations

102.1 Overview

This chapter describes a quick installation example of the EBX® Metadata Management application for on-premise for Tomcat where TIBCO EBX® is already installed.

The EBX® Metadata Management application requires the following:

- EBX® 6.2.3,
- EBX® Metadata Classifier and Classifier database [p 565],
- EBX® Add-ons web applications [p 568],
- EBX® Metadata Agent [p 570], and
- EBX® Metadata Management module [p 568].

After installing and configuring the above components, perform a Final verification [p 570] to ensure the EBX® Metadata Management application is available.

See Installation notes for Tomcat 10.1.X [p 557] for more information.

102.2 EBX® Metadata Classifier and Classifier database

Use the information in this section to install, configure, and run the classifier and classifier database.

Requirements

Before beginning, ensure you have the following installed on your system:

- Either Python 3.11.x + pipenv v2026.5.2 or Docker CLI (see Configuration and startup [p 566])
- Database: SQLite, PostgreSQL, Microsoft SQL Server, or Oracle database (via ORM SQLAlchemy)

See Supported databases [p 527] for more information.

Configuration and startup

Use one of the following methods to install, configure, and run the EBX® Metadata Classifier python application in the target environment:

- Option 1: Standalone installation [p 566]
- Option 2: Run as Docker container [p 567]

Option 1: Standalone installation

1. Copy the `ebx.software/ebx-metadata/classifier` directory from the EBX® installation CD to your working directory.

   In this example, the directory is copied to `/opt/ebx`, so the EBX® Metadata Classifier directory will be located at: `/opt/ebx/classifier`

2. Update the system properties in the `setEnv.sh` (Linux) or `setEnv.bat` (Windows) file located in the `/opt/ebx/classifier` directory to use the correct logging directory and database connection.

   Attention

   Make sure that the directory specified in *EBX_METADATA_CLASSIFIER_LOG_DIR* is writeable.

   Attention

   If an SQLite database is used, make sure that the directory file defined in *EBX_METADATA_CLASSIFIER_SQLALCHEMY_URL* is writeable.

   The following example applies to Linux:

```
######################################################################
## Set the path to the EBX Metadata Classifier application log directory
######################################################################
export EBX_METADATA_CLASSIFIER_LOG_DIR="ebx/logs"

######################################################################
## The database connection URL in SQLAlchemy format.
##    Examples:
##      SQLite:      sqlite:///<sqlite_db_file>
##      PostgreSQL:  postgresql://<username>:<password>@<host>:<port>/<database_name>
######################################################################
export EBX_METADATA_CLASSIFIER_SQLALCHEMY_URL=sqlite:///ebx/data/sqlite/ebx-metadata-classifier.db
```

   If the EBX® Metadata Classifier is configured to use a database other than SQLite, you must install the corresponding driver. For example with PostgreSQL, you must install the `psycopg2-binary` Python module by running the following command:

```
export PIPENV_PIPFILE="/opt/ebx/classfifier/ebx_metadata_classifier/Pipfile"
```

```
pipenv install psycopg2-binary
```

> **Note**
>
> See SQLAlchemy Database URLs for more information.

3. Run `install-classifier.sh` (Linux) or `install-classifier.bat` (Windows) to configure and install the required Python modules.

> **Attention**
> Perform this step again whenever EBX® is updated to a new version.

4. Start and test the Classifier service
   - On Linux: `start-classifier.sh`
   - On Windows: `start-classifier.bat`

   After the service starts, ensure that the following URL can be accessed: `https://<EBX_METADATA_CLASSIFIER_HOST>:<EBX_METADATA_CLASSIFIER_PORT>/api`
5. (Optional): Run `stop-classifier.sh` (Linux) or `stop-classifier.bat` (Windows) to stop the Classifier service.

Option 2: Run as Docker container

Use a Dockerfile-based installation to run an EBX® Metadata Classifier docker container on the hosted machine.

1. Copy the `ebx.software/ebx-metadata/classifier` directory from the EBX® installation CD to your working directory.

   In this example, the directory is copied to `/opt/ebx`, so the EBX® Metadata Classifier directory will be located at: `/opt/ebx/classifier`
2. To create the EBX® Metadata Classifier Docker image, run the following command in the `/opt/ebx/classifier` directory:

```
docker build -f docker/Dockerfile -t ebx-metadata-classifier:6.2.3 .
```

   By default, this image defines the following docker volumes:
   - `/app/ebx/logs` — for EBX® Metadata Classifier log files
   - `/app/ebx/data/sqlite` — for storing the SQLite database file
3. To start the EBX® Metadata Classifier Docker image, run the following command:

```
docker run -p 5000:5000 -v /my_custom/data:/app/ebx/data -d ebx-metadata-classifier:6.2.3
```

The EBX® Metadata Classifier docker container can be configured using the following environment variables:

Environment variable name	Description	Default value
EBX_METADATA_CLASSIFIER_LOG_LEVEL	Logging level for the EBX® Metadata Classifier module.	`INFO`
EBX_METADATA_CLASSIFIER_SQLALCHEMY_URL	The database connection URL in SQLAlchemy format.	`sqlite:////app/ebx/data/sqlite/ebx-metadata-classifier.db`

> **Note**
>
> Several containers can be run on a host, the entry port should be different (for example: `-p 5001:5000`)

102.3 EBX® Metadata Management module

The following topics in this section describes how configure and deploy the EBX® Metadata Management module:

- Requirements [p 568]
- Configuration [p 568]
- Deployment [p 569]

Requirements

Prior to configuring and deploying the EBX® Metadata Management module, use the instructions in Installation notes for Tomcat 10.1.X [p 557] and Supported application servers [p 525] to ensure EBX® 6.2.3 and the following add-ons installed:

- TIBCO EBX® Digital Asset Manager Add-on (DAMA)
- TIBCO EBX® Data Exchange Add-on (DINT)
- TIBCO EBX® Data Model and Data Visualization Add-on (DMDV)
- TIBCO EBX® Insight Add-on (DPRA)
- TIBCO EBX® Information Search Add-on (TESE)

See Managing TIBCO EBX® add-ons [p 605] for more information.

Configuration

1. Copy `ebx.software/ebx-metadata/config/ebx-metadata.properties` from the EBX® installation CD to your *EBX_HOME* directory
2. Update the `ebx-metadata.properties` file in your *EBX_HOME* directory as follows:

```
################################################################################################################
###
### EBX Metadata configuration
###
################################################################################################################

################################################################################################################
# Directory for DAMA files
```

```
################################################################################################################
ebx-metadata.dama.directory=${ebx.home}/metadata/dama

################################################################################################################
# Specific directories for DAMA files
#  Set one of these properties to use a different location for a DAMA data drive
################################################################################################################
#ebx.metadata.dama.builtins.path=${ebx-metadata.dama.directory}/builtins
#ebx.metadata.dama.allusers.path=${ebx-metadata.dama.directory}/allusers
#ebx.metadata.dama.admin.path=${ebx-metadata.dama.directory}/admin
#ebx.metadata.dama.learning.path=${ebx-metadata.dama.directory}/learning
#ebx.metadata.dama.bow.path=${ebx-metadata.dama.directory}/bow

################################################################################################################
# Directory for startup and trial staging archives
################################################################################################################
ebx-metadata.init.directory=${ebx.home}/metadata/init

################################################################################################################
## Directory for temporary resources
################################################################################################################
# When set, allows to specify a directory for temporary files different from java.io.tmpdir.
# Default value is java.io.tmpdir
ebx-metadata.temp.directory=${java.io.tmpdir}

################################################################################################################
# Startup waiting time for ebx-metadata (in seconds)
################################################################################################################
ebx-metadata.waiting.time=600

################################################################################################################
# Maximum number of predictions returned by Metadata Classifier
################################################################################################################
ebx-metadata.classifier.maxNumberOfPredictions=10
```

3. Copy `ebx.software/ebx-metadata/config/init` folder from the EBX® installation CD to the `ebx-metadata.init.directory` specified in `ebx-metadata.properties`.

> **Attention**
> Make sure that the directory specified in *ebx-metadata.dama.directory* is writeable.

4. Configure logging for EBX® Metadata by adding following lines to your `ebx.properties` file:

```
#Metadata Log4J Properties:
ebx.log4j.category.log.wbp.ebx-metadata=           INFO, Console, ebxFile:ebx-metadata
```

5. Add the environment variables `ebx-metadata.properties` and `ebx-metadata-classifier.url` by modifying the `setenv.sh` (Linux) or `setenv.bat` (Windows) file in either `$CATALINA_HOME/bin` or `$CATALINA_BASE/bin`.
   - For Unix shell launch:

```
EBX_OPTS="${EBX_OPTS} -Debx-metadata.properties=${EBX_HOME}/ebx-metadata.properties -Debx-metadata-
classifier.url=http://<EBX_METADATA_CLASSIFIER_HOST>:<EBX_METADATA_CLASSIFIER_PORT>"
```

   - For Windows Command Prompt launch:

```
set EBX_OPTS="%EBX_OPTS%" -Debx-metadata.properties="%EBX_HOME%\ebx-metadata.properties" -Debx-
metadata-classifier.url="http://<EBX_METADATA_CLASSIFIER_HOST>:<EBX_METADATA_CLASSIFIER_PORT>"
```

Deployment

To deploy the EBX® Metadata module for Tomcat:

1. Copy the `ebx.software/ebx-metadata/webapps/wars-packaging/ebx-metadata.war` file from the EBX® installation CD in `$CATALINA_HOME/webapps/` (or `$CATALINA_BASE/webapps/`) directory
2. After Tomcat launch, run EBX® web application by entering the following URL in the browser: http://localhost:8080/ebx/

102.4 EBX® Metadata Agent

See Installing the TIBCO® Metadata Agent [p 815] for more information.

102.5 Final verification

After Tomcat starts successfully, use the following to verify that the EBX® Metadata Management application has been installed correctly:

- Check the log for the following message: `Module ebx-metadata repository initialization complete.`
- In the EBX® UI check whether the list of perspectives in the main toolbar includes **Metadata**. If it does, it indicates that installation was successful.

See Accessing the metadata perspective [p 259] for more information.

102.6 Limitations

When using EBX® Container Edition, it is not supported to install the EBX® Metadata Classifier outside the container.

CHAPTER 103

TIBCO EBX® main configuration file

This chapter contains the following topics:

1. Overview
2. Setting an EBX® license key properties file
3. Setting automatic installation on first launch
4. Setting the EBX® root directory
5. Configuring the EBX® repository
6. Configuring the user and roles directory
7. Configuring EBX® localization
8. Setting temporary files directories
9. Activating the audit trail
10. Activating the legacy XML audit trail (deprecated)
11. Configuring the EBX® logs
12. Activating and configuring SMTP and emails
13. Configuring data services
14. Activating and configuring JMS
15. Configuring distributed data delivery (D3)
16. Configuring REST toolkit services
17. Keystore sensitive property file location
18. Activating and configuring staging
19. Configuring Web access from end-user browsers
20. Security configuration
21. Configuring failover
22. Tuning the EBX® repository
23. Advanced index configuration
24. Miscellaneous

103.1 Overview

The EBX® main configuration file, by default named `ebx.properties`, contains most of the basic parameters for running EBX®. It is a Java properties file that uses the standard simple line-oriented format.

The main configuration file complements the Jakarta EE deployment descriptor [p 539]. Administrators can also perform further configuration through the user interface, which is then stored in the EBX® repository.

See also

Deployment details [p 539]

UI administration [p 671]

Location of the file

The access path to the main configuration file can be specified in several ways. In order of descending priority:

1. By defining the Java system property 'ebx.properties'. For example, this property can be set by adding the option `-Debx.properties=<filePath>` to the `java` command-line command. See Java documentation.
2. By defining the servlet initialization parameter 'ebx.properties'.

   This standard Jakarta EE setting must be specified in the `web.xml` file of the web application '`ebx`'. EBX® accesses this parameter by calling the method `ServletConfig.getInitParameter("ebx.properties")` in the servlet `FrontServlet`.

   See getInitParameter in the `ServletConfig` documentation.
3. By default, if nothing is specified, the main configuration file is located at `WEB-INF/ebx.properties` of the web application '`ebx`'.

> **Note**
>
> In addition to specifying properties in the main configuration file, it is also possible to set the values of properties directly in the system properties. For example, using the `-D` argument of the `java` command-line command.

Custom properties and variable substitution

The value of any property can include one or more variables that use the syntax `${`*`propertyKey`*`}`, where *`propertyKey`* is either a system property, or a property defined in the main configuration file.

For example, the default configuration file provided with EBX® uses the custom property `ebx.home` to set a default common directory, which is then included in other properties.

103.2 Setting an EBX® license key properties file

See also *Initialization and first-launch assistant [p 603]*

The license key can be retrieved from the TIBCO eDelivery site.

```
#################################################################
## EBX® license file
```

```
## Define the license file using one of the following methods.
## If the license file is defined in multiple locations, the
## order of priority is shown below:
## 1. System property -Debx.license.file=<properties-file>.
## 2. Main ebx.properties configuration 'ebx.license.file'
##    property value.
## 3. Default: ${ebx.home}/ebx-license.properties
##    The file should have an 'ebx.license' property containing
##    the license key.
#################################################################
ebx.license.file=location_of_license_properties
```

The license key properties file must contain.

```
#################################################################
## EBX® License Key
##    Ensure the file is writable by EBX®
##    to enable updates using the UI license key wizard process.
#################################################################
ebx.license=<your_license_key>
```

You can update the license key without restarting the EBX® server. When the server detects a change, it writes a kernel log entry with the license status. If the key is invalid, the license wizard displays when a user accesses the interface.

> **Note**
>
> For a time-limited license key and some time before its expiration, a daily kernel WARN log informs that the license will expire soon.

103.3 **Setting automatic installation on first launch**

Repository can be automatically installed on first startup.

```
################################################
## Installation on first launch.
## All values are ignored if the repository is  already installed.
################################################
## Enables repository installation on first startup (default is false).
## If true, property ebx.license from ebx.license.file should also be set to a valid license.
ebx.install.enabled=true

## Following properties configure the repository. Values are optional and defaults are
## automatically generated.
ebx.install.repository.id=00275930BB88
ebx.install.repository.label=A Test

## Following properties specify the administrator. These are ignored if a custom
## directory is defined.
ebx.install.admin.login=admin
ebx.install.admin.firstName=admin
ebx.install.admin.lastName=admin
ebx.install.admin.email=adamin@example.com

## Following property specifies the non-encrypted password used for the administrator.
## It is ignored if a custom directory is defined. It cannot be set if
## property ebx.install.admin.password.encrypted is set.
#ebx.install.admin.password=admin

## Following property specifies the encrypted password used for the administrator.
## It is ignored if a custom directory is defined. It cannot be set if
## property ebx.install.admin.password is set.
## Password can be encrypted by using command:
## java -cp ebx.jar com.orchestranetworks.service.directory.EncryptPassword <login> <password_to_encrypt>
ebx.install.admin.password.encrypted=ff297ae08f7eeb63230b55f7c45a720a017bc71d22eaaec...
```

103.4 Setting the EBX® root directory

The EBX® root directory contains the Lucene indexes directory, the archives and, when the repository is persisted on H2 standalone mode, the H2 database files. It also contains the legacy XML audit trail (deprecated).

```
################################################
## Path for EBX® XML repository
################################################
ebx.repository.directory=${ebx.home}/ebxRepository
```

See also *Monitoring and clean up of the file system* *[p 669]*

103.5 Configuring the EBX® repository

Before configuring the persistence properties of the EBX® repository, carefully read the section Technical architecture [p 660] in the chapter 'Repository administration'.

The required library (driver) for each supported database is described in the chapter Database drivers [p 535].

See also

Repository administration *[p 660]*

Rules for the database access and user privileges *[p 661]*

Supported databases *[p 527]*

Data source of the EBX® repository *[p 540]*

Database drivers *[p 535]*

```
################################################
## The maximum time to set up the database connection,
## in milliseconds.
################################################
ebx.persistence.timeout=10000
################################################
## The prefix to add to all table names of persistence system.
## This may be useful for supporting multiple repositories in the relational database.
## Default value is 'EBX_'.
################################################
ebx.persistence.table.prefix=

################################################
## Case EBX® persistence system is H2 'standalone'.
################################################
ebx.persistence.factory=h2.standalone
ebx.persistence.user=sa
ebx.persistence.password=

################################################
## Case EBX® persistence system is H2 'server mode',
################################################
#ebx.persistence.factory=h2.server

## Specific properties to be set only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url=jdbc:h2:tcp://127.0.0.1/ebxdb
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


################################################
## Case EBX® persistence system is Oracle database.
################################################
#ebx.persistence.factory=oracle

## Specific properties to be set only if you want to ignore the standard
```

```
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url=jdbc:oracle:thin:@127.0.0.1:1521:ebxDatabase
#ebx.persistence.driver=oracle.jdbc.OracleDriver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


## Activate to use VARCHAR2 instead of NVARCHAR2 on Oracle; never modify on an existing
## repository.
#ebx.persistence.oracle.useVARCHAR2=false

################################################
## Case EBX® persistence system is Microsoft SQL Server.
################################################
#ebx.persistence.factory=sqlserver

## Specific properties to be set only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url= \
#jdbc:sqlserver://127.0.0.1:1036;databasename=ebxDatabase
#ebx.persistence.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


################################################
## Case EBX® persistence system is Microsoft Azure SQL database.
################################################
#ebx.persistence.factory=azure.sql

## Specific properties to be set only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url= \
#jdbc:sqlserver://myhost.database.windows.net:1433;database=ebxDatabase;encrypt=true;\
#trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;
#ebx.persistence.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


################################################
## Case EBX® persistence system is PostgreSQL.
################################################
#ebx.persistence.factory=postgresql

## Specific properties to be set only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url=jdbc:postgresql://127.0.0.1:5432/ebxDatabase
#ebx.persistence.driver=org.postgresql.Driver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy
```

103.6 **Configuring the user and roles directory**

To configure the directory, set and update the following dedicated properties:

See also

Users and roles directory *[p 703]*

`DirectoryFactory`[API]

```
#####################################################################
##
## EBX DIRECTORY CONFIGURATION
##
#####################################################################
## 1. Choose the main directory implementation.
## EBX operates with a main directory, which serves as the default authentication source.
## This main directory can be the default directory, the built-in LDAP directory,
## or any custom directory.
## The value must be the fully qualified name of the Java class that implements
## com.orchestranetworks.service.directory.DirectoryFactory
## To choose a main directory
##  - Default directory       : keep empty
##  - LDAP build-in directory : com.orchestranetworks.service.directory.ldap.LdapDirectoryFactory
```

```
##  - Custom directory        : the Java directory factory class name
## Depending on the selected directory, additional properties may be required:
## ebx.directory.ldap.*
################################################
#ebx.directory.factory=
################################################
## 2. Enabling and configuring SSO if needed.
## any non-empty value from the following enables the SSO:
##    - "OIDC"     :   to use OIDC.
##    - "SAML2"    :   to use SAML2.
## Depending on the selected protocol, additional properties may be required from:
## ebx.directory.saml2.* , ebx.directory.oidc.* and ebx.directory.bearer.*
#ebx.directory.sso.protocol=OIDC
################################################
## 3- Enable the synchronized directory
## When SSO or Centralized Authentication is enabled, a second directory,
## referred to as the "synchronized directory", can be added to manage users
## authenticated via the identity provider. In this configuration,
## the main directory is preferably reserved for technical or service users,
## while the synchronized directory handles user accounts coming from the external system.
## any non-empty value from the following enables the synchronized directory:
##    - "LDAP"    :    to use the built-in  LDAP as source
##    - "SCIM"    :    to use the built-in  SCIM as source
##    - "OnLogin" :    to use an on login strategy that only stores users that already
##                     logged in once using SSO.
## Depending on the selected synchronized directory, additional properties may be required:
## ebx.directory.synchronized.* , ebx.directory.ldap.* and ebx.directory.scim.*
#ebx.directory.synchronized.source=SCIM
```

It is also possible to disable the built-in role "ADMINISTRATOR".

```
################################################
## Specifies whether the built-in role ADMINISTRATOR is disabled.
## Default value is false.
################################################
#ebx.directory.disableBuiltInAdministrator=true
```

Synchronized directory

To configure the synchronized directory, set and update the following dedicated properties:

```
################################################
## Synchronized directory
################################################
## Specifies the EBX® external roles mapper.
## Used to map external roles coming from built-in configurations: LDAP, SCIM, OnLogin.
## possible values:
##      - "DEFAULT" or empty:
##             Default behaviour : External roles are used as specific roles without mapping.
##             the built-in roles Administrator and Read Only are mapped through
##             the properties : ebx.directory.synchronized.role.mapper.default.*.
##      - "SCRIPT" : use of scripts to map external roles.
##      - "CUSTOM" : an implementation of com.orchestranetworks.service.directory.api.ExternalRolesMapper
##                   to mention above
ebx.directory.synchronized.role.mapper=
#ebx.directory.synchronized.role.mapper.default.administrator=
#ebx.directory.synchronized.role.mapper.default.read.only=
## Custom implementation of com.orchestranetworks.service.directory.api.ExternalRolesMapper
#ebx.directory.synchronized.role.mapper.custom.class=
## The allowed attributes to get from the LDAP response
## and persist in the synchronized directory user attributes.
ebx.directory.synchronized.mapping.ldap.attributes=
## The name of an attribute that contains a role.
## An SAML 2.0 assertion may contain multiple attributes with the same name,
## if the name corresponds to the role name attribute each value will be considered as a role.
## Required if SCIM or OnLogin directory is enabled and SAML2 is used.
## Default value is Role.
ebx.directory.synchronized.sso.mapping.saml2.attributes.role.name=Role
## the allowed attributes to get from the saml assertion
## and persist in the synchronized directory user attributes.
ebx.directory.synchronized.sso.mapping.saml2.attributes=username
## The name of the custom clam that contains a table of roles.
## Required if synchronized directory is enabled and OIDC is used.
ebx.directory.synchronized.sso.mapping.oidc.claims.role.name=groups
## The allowed claims to get from the OIDC token
## and persisted in the synchronized directory user attributes.
ebx.directory.synchronized.sso.mapping.oidc.claims=sub,preferred_username,iss
## The name of the custom clam that contains a table of roles.
## Required if synchronized directory is enabled and BEARER authentication is used.
## Comma separated values.
ebx.directory.synchronized.sso.mapping.bearer.claims.role.name=groups
## The allowed claims to get from the token
```

```
## and persisted in the synchronized directory user attributes.
## Note: when jwt is not configured, the user should be declared on the
##       synchronized directory to retrieve the claims.
ebx.directory.synchronized.sso.mapping.bearer.claims=sub,preferred_username,iss
## The attribute name that corresponds to the user mail.
ebx.directory.synchronized.sso.mapping.attribute.mail=email
## The user display may be created using an expression from the attributes.
ebx.directory.synchronized.sso.display.user={{preferred_username}}
## These listeners are called when a user logs in or logs out using SSO.
## Implementations of com.orchestranetworks.service.directory.sso.api.OnUserLoginListener
## and com.orchestranetworks.service.directory.sso.api.OnUserLogoutListener
## Comma separated values
ebx.directory.synchronized.sso.listeners=
```

See also *Synchronized directory* *[p 706]*

Built-in LDAP directory

Activate and configure the built-in LDAP (Lightweight Directory Access Protocol) directory implementation through the main EBX® configuration file.

To activate LDAP, update the *ebx.directory.factory* property as follows:

```
ebx.directory.factory=com.orchestranetworks.service.directory.ldap.LdapDirectoryFactory
```

To configure LDAP, set and update the following dedicated properties:

```
#####################################################################
## Built-in LDAP Directory
#####################################################################
## 1- Connection: [ebx.directory.ldap.default]
##      required:               - .hostName
##                              - .port
##      optional:               - .connectionTimeOutInSeconds | default value: '30'   | possible values :
 positive integer
##                              - .bindDnOrUser               | default value is empty
##                              - .bindPassword               | default value is empty
##                              - .authenticationMethod       | default value: 'none' | possible values :
 'simple', 'none'
##                              - .encryptionMethod           | default value: 'none' | possible values : 'ssl',
 'tls', 'none'
##          Note: .bindDnOrUser and .bindPassword are required when using a basic authentication method.
## 2- Mapping: [ebx.directory.ldap.default.mapping]
##      required:   - .role.referenceAttribute      : the ldap attribute that contains the role name
##                  - .role.memberAttribute         : the ldap attribute that contains the users in a group entry
##                  - .role.builtin.administrator   : the ldap group associated with the builtin administrator
 role
##                  - .role.builtin.readOnly        : the ldap group associated with the builtin read-only role
##                  - .user.referenceAttribute      : the ldap attribute that contains the user name to use as a
 login
##                  - .user.referenceComponentInDN  : the ldap dn component that contains the username to use
##      optional:
##                  - .role.mailAttribute         : the ldap attribute that contains the role mail
##                  - .user.mailAttribute         : the ldap attribute that contains the user mail
##          Note: If not defined, the mails will not be handled.
## 3- Display: [ebx.directory.ldap.default.display]
##      optional:   - .role        : an expression using ldap attributes | example: {{cn}}-{{mail}} which gives:
 sales-salesTeam@mail.com
##                  - .user        : an expression using ldap attributes | example: {{salutation}} {{firstName}}
 {{lastName}} Mr John Doe
##                  - .users.all   : If true, all users are returned as profiles with the roles.
##          Note: Display is not locale dependent yet.
## 4- Search request templates: [ebx.directory.ldap.default.request.{requestTemplateName}.{field}]
##   4.1- properties of each request template ({field}):
##      required:   - .baseDN                    : the request base search DN entry
##                  - .filter                    : the request filter
##      optional:   - .scope                     | default value: 'subtree'
##                                               | possible values :  'base','one','subtree'
##                  - .cache.expirationInSeconds | default value: '120'
##                                               | possible values :  positive 32 bit integer
##                  - .cache.refreshInSeconds    | default value: '60'
##                                               | possible values :  positive 32 bit integer
##                  - .cache.maxSize             | default value: '1000'    | possible values :  positive 64 bit
 integer
##                  - .pageSize                  | default value: '100'     | possible values :  positive 32 bit
 integer
##                  - .timeLimitInSeconds        | default value: '30'      | possible values :  positive 32 bit
 integer
##          Note: Caches are used on every request template to enhance performance.
##                Setting a request template's .cache.maxSize and .cache.expirationInSeconds to 0 will
```

```
##                 disable its related cache.
##               Changing the .pageSize of a request template at server runtime will
##                 evict its related cache.
##   4.2- Request templates {requestTemplateName}:
##      required:   - .userSearch         : search a user                  | allowed placeholders : {{login}}
##                  - .userGroupsSearch   : search groups of a user        | allowed placeholders : {{login}},
{{dn}}
##                  - .groupUsersSearch   : search the users under a group  | allowed placeholders : {{role}}
##                  - .allGroupsSearch    : search all groups              | allowed placeholders :
##                  - .allUsersSearch     : search all users               | allowed placeholders :
#########################################################################
## 1- Connection
ebx.directory.ldap.default.hostName=localhost
ebx.directory.ldap.default.port=389
ebx.directory.ldap.default.connectionTimeOutInSeconds=30
ebx.directory.ldap.default.bindDnOrUser=cn=admin,dc=example,dc=org
ebx.directory.ldap.default.bindPassword=admin
ebx.directory.ldap.default.authenticationMethod=simple
#ebx.directory.ldap.default.encryptionMethod=
## 2- Mapping
ebx.directory.ldap.default.mapping.role.referenceAttribute=cn
ebx.directory.ldap.default.mapping.role.memberAttribute=member
ebx.directory.ldap.default.mapping.user.referenceAttribute=uid
ebx.directory.ldap.default.mapping.role.mailAttribute=mail
ebx.directory.ldap.default.mapping.user.mailAttribute=mail
## 3- Display
ebx.directory.ldap.default.display.role={{cn}}
ebx.directory.ldap.default.display.user={{cn}} ({{mail}})
## 4- Request templates
ebx.directory.ldap.default.request.userSearch.baseDN=dc=example,dc=org
ebx.directory.ldap.default.request.userSearch.filter=(uid={{login}})
ebx.directory.ldap.default.request.userSearch.scope=subtree
ebx.directory.ldap.default.request.userSearch.cache.expirationInSeconds=120
ebx.directory.ldap.default.request.userSearch.cache.refreshInSeconds=60
ebx.directory.ldap.default.request.userSearch.cache.maxSize=1000
ebx.directory.ldap.default.request.userSearch.pageSize=100
ebx.directory.ldap.default.request.userSearch.timeLimitInSeconds=30
ebx.directory.ldap.default.request.userGroupsSearch.baseDN=
${ebx.directory.ldap.default.request.userSearch.baseDN}
ebx.directory.ldap.default.request.userGroupsSearch.filter=(&(objectClass=groupOfNames)(member={{dn}}))
ebx.directory.ldap.default.request.userGroupsSearch.scope=subtree
ebx.directory.ldap.default.request.userGroupsSearch.cache.expirationInSeconds=120
ebx.directory.ldap.default.request.userGroupsSearch.cache.refreshInSeconds=60
ebx.directory.ldap.default.request.allGroupsSearch.baseDN=${ebx.directory.ldap.default.request.userSearch.baseDN}
ebx.directory.ldap.default.request.allGroupsSearch.filter=(objectClass=groupOfNames)
ebx.directory.ldap.default.request.allGroupsSearch.scope=subtree
ebx.directory.ldap.default.request.allGroupsSearch.cache.expirationInSeconds=120
ebx.directory.ldap.default.request.allGroupsSearch.cache.refreshInSeconds=60
ebx.directory.ldap.default.request.allGroupsSearch.pageSize=100
ebx.directory.ldap.default.request.allUsersSearch.baseDN=${ebx.directory.ldap.default.request.userSearch.baseDN}
ebx.directory.ldap.default.request.allUsersSearch.filter=(objectClass=inetOrgPerson)
ebx.directory.ldap.default.request.allUsersSearch.scope=subtree
```

See also *LDAP directory* *[p 706]*

SCIM directory

Activate and configure the SCIM (System for Cross-domain Identity Management) feature through the main EBX® configuration file.

To configure SCIM, set and update the following dedicated properties:

```
#################################################
## SCIM properties
## SCIM uses Users and Groups requests that must be configured.
## Authentication info goes in headers which are '%' separated key:value pairs.
## Cache is tuned with max size and expiration.
## Default cache values are 1000 entries and 60 seconds.
#################################################
#ebx.directory.scim.default.request.users.url=\
#  http[s]://<hostname>:<port>/<scim-users-path>
#ebx.directory.scim.default.request.users.headers=\
#  Authorization: Bearer <token> \
#  % Content-Type: application/json
#ebx.directory.scim.default.request.users.page.size=2
#ebx.directory.scim.default.request.users.cache.max.size=1000
#ebx.directory.scim.default.request.users.cacheExpiration=60
#ebx.directory.scim.default.request.groups.url=\
#  http[s]://<hostname>:<port>/<scim-groups-path>
#ebx.directory.scim.default.request.groups.headers=
#ebx.directory.scim.default.request.groups.page.size=2
```

```
#ebx.directory.scim.default.request.groups.cache.max.size=1000
#ebx.directory.scim.default.request.groups.cache.expiration=60
```

See also *SCIM directory [p 708]*

SAML 2.0

SAML 2.0 can be activated and configured through the main configuration file.

```
######################################################################
## SAML 2.0 properties
## The configuration files idp-descriptor.xml should be located under
## {ebx.repository.directory}
## An sp-descriptor.xml would be generated on the system startup and located under
## {ebx.repository.directory}. The final structure:
## {ebx.repository.directory}
##           |- saml2
##                |- idp-descriptor.xml  : to put manually.
##                |- sp-descriptor.xml   : generated on startup and on property update.
######################################################################
## [Required if no servlet configuration is set {ebx.servlet.http(s).xxx}]
## The base uri of the service provider.
#ebx.directory.saml2.default.sp.base.url=http[s]://<hostname>:<port>
## [Required if SAML2.0 is enabled] The service provider entity ID
#ebx.directory.saml2.default.sp.entity.id=
## Keystore configuration
## The keystore type, default value is PKCS12
#ebx.directory.saml2.default.keystore.type=PKCS12
## The keystore path
#ebx.directory.saml2.default.keystore.path=
## [Required if SAML2.0 is enabled] keystore password
#ebx.directory.saml2.default.keystore.password=
## [Required if SAML2.0 is enabled] keystore alias
#ebx.directory.saml2.default.keystore.key.alias=
## [Required if SAML2.0 is enabled] private key password
#ebx.directory.saml2.default.keystore.key.password=
## [Optional] The Name ID attribute to use as userName.
#ebx.directory.saml2.default.name.id.attribute=
## [Optional] The Name ID format to use as userName if the nameId.attribute is not set or found
#ebx.directory.saml2.default.name.id.format=
## [Optional] Specifies whether the service provider signs authentication
## requests it sends to the identity provider
## default value: true
#ebx.directory.saml2.default.authn.requests.signed=true
## [Optional] Specify whether the service provider signs authentication requests
## it sends to the identity provider
## default value: true
#ebx.directory.saml2.default.want.assertions.signed=true
## [Optional] specify whether the service provider (SP) wants the SAML assertions received from
## the identity provider to be signed or not.
## default value: true
#ebx.directory.saml2.default.want.responses.signed=true
## [Optional] specifies whether the Service Provider signs the logout requests it sends to the
## Identity Provider.
## default value: true
#ebx.directory.saml2.default.sp.logout.request.signed=true
```

See also *SAML 2.0 [p 613]*

OpenID Connect

OpenID Connect can be activated and configured through the main configuration file.

```
######################################################################
## OpenID Connect properties
######################################################################
## [Required if no servlet configuration is set {ebx.servlet.http(s).xxx}] The
## base url of the client.
#ebx.directory.oidc.default.client.base.url=http[s]://<hostname>:<port>
## [Required if oidc is enabled] the oidc client id
#ebx.directory.oidc.default.client.id=ebxoidc
## [Required if oidc is enabled] the oidc client secret
#ebx.directory.oidc.default.client.secret=ZXLYr53k8mPBKDQAqSMX7gdQ2iWMxN9u
## [Required if oidc is enabled] the Discovery URI or the file URI that contains the
## configuration.
#ebx.directory.oidc.default.discovery.uri=\
#  http[s]://<hostname>:<port>/<openid-discovery-path>
## [Optional] The OIDC claim to use as a username.
```

```
## default value: preferred_username
#ebx.directory.oidc.default.claim.username=preferred_username
## [Optional] Specifies the maximum time to wait in milliseconds for a connection to
## be established
## with the OIDC Provider
## default value: 500
#ebx.directory.oidc.default.http.connect.timeout=1000
## [Optional] Specifies the maximum time to wait in milliseconds for data to be read
## from the input
## stream once the connection is established
## default value: 5000
#ebx.directory.oidc.default.http.read.timeout=5000
```

See also *OpenID Connect* *[p 614]*

Bearer Authentication

Bearer authentication can be activated and configured through the main configuration file.

> **Note**
>
> Make sure that in case of JWT, the "sub" (subject) claim contains the user name to use in EBX®.

```
######################################################################
## Bearer authentication properties
## ----------------------------------------------------------------
## Enable the Bearer authentication
## default value: false
#ebx.directory.bearer.default.enabled=true
## [Required to enable JWT] the URI of the file or the link that contains the JWT keys.
#ebx.directory.bearer.default.jwt.jwks.uri=\
#  http[s]://<hostname>:<port>/<bearer-certs-path>
## [Required if JWT is enabled] define the issuers to verify on the jwt token
## comma separated values.
#ebx.directory.bearer.default.jwt.issuers=http[s]://<hostname>:<port>/<bearer-issuer-path>
## [Optional] define the scope to verify on the jwt token when authenticating
## in data services.
#ebx.directory.bearer.default.scope.dataservices=ebx-dataservices
## [Optional] The bearer claim to use as a username.
## default value: preferred_username
#ebx.directory.bearer.default.jwt.claim.username=preferred_username
## enables the handling of encrypted JWT.
## default value : false
#ebx.directory.bearer.default.jwt.encryption.enabled=false
## Key store configuration to handle JWT decryption.
## keystore type, default value is PKCS12
#ebx.directory.bearer.default.jwt.encryption.keystore.type=PKCS12
## [Required if encryption is enabled] keystore path
#ebx.directory.bearer.default.jwt.encryption.keystore.path=/home/keystore.p12
## [Required if encryption is enabled] keystore password
#ebx.directory.bearer.default.jwt.encryption.keystore.password=keypass
## [Required if encryption is enabled] key alias
#ebx.directory.bearer.default.jwt.encryption.keystore.key.alias=ebx.jwt
## [Required if encryption is enabled] key password
#ebx.directory.bearer.default.jwt.encryption.keystore.key.password=keypass
## [Optional] an introspection http request pattern to verify a token by the
## Authorization server.
## Http syntax:
##      Arguments: @method @uri @header[*] @content.
##      Encoding: uri and its query params should be encoded.
##      Header: '%' is used as a separator of headers
##      Place holder: {{token}} place holder should be used and will be replaced by the
##      token to verify on the Authorization server.
##      '@' is a reserved character delimiter that should not be used in the request unless
##      encoded If an introspection request is set, no local verification is done on the
##      bearer token even if it is a JWT and consequently it may cause performance issues.
#ebx.directory.bearer.default.introspection.request=\
# @method [POST] \
# @uri [http[s]://<hostname>:<port>/<bearer-introspection-path>] \
# @header [Content-Type: application/x-www-form-urlencoded] \
# @content [token={{token}}&client_id=oauth2& \
# client_secret=rNvm8412z0o9ZZFvOirUZeOJYIyRz8cK]
```

103.7 Configuring EBX® localization

This parameter is used to configure the locales used at runtime. This list must contain all the locales that are exposed to the end-user. EBX® will not be able to display labels and messages in a language that is not declared in this list.

The default locale must be the first one in the list.

```
################################################
## Available locales, separated by a comma.
## The first element in the list is considered as the default locale.
## If not set, available locales are 'en-US, fr-FR'.
##
################################################
#ebx.locales.available=en-US, fr-FR
```

See also *Extending TIBCO EBX® internationalization [p 453]*

103.8 Setting temporary files directories

Temporary files are stored as follows:

```
################################################
## Directories for temporary resources.
################################################
## The property ebx.temp.directory allows to specify a directory for temporary files.
## Default value is java.io.tmpdir
#ebx.temp.directory = /tmp/java
ebx.temp.directory = \\${java.io.tmpdir}

## The property ebx.temp.cache.directory allows to specify the directory containing
## temporary files for cache.
## Default value is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.cache.directory = ${ebx.temp.directory}/ebx.platform

## The property ebx.temp.import.directory allows to specify the directory containing
## temporary files for import.
## Default value is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.import.directory = ${ebx.temp.directory}/ebx.platform
```

103.9 Activating the audit trail

By default, the audit trail logging is deactivated. It can be activated by configuring the "audit" log category to log INFO level messages. For instance the following will log audit trail messages using a specific 'ebxFile' appender [p 584]:

```
ebx.log4j.category.log.audit = INFO, ebxFile:audit
```

See also

Configuring the EBX® logs [p 582]

Audit trail [p 733]

103.10 Activating the legacy XML audit trail (deprecated)

By default, the XML audit trail is deactivated. It can be activated using the following variable:

```
################################################
## The XML history has been replaced by an SQL history.
## This old XML history can be activated using the following variable.
## Default is false.
################################################
```

```
ebx.history.xmlaudittrail.activated = false
```

See also *Legacy XML Audit trail*

103.11 Configuring the EBX® logs

See also *Log configuration* *[p 521]*

EBX® main file mechanism

The most important logging categories are:

ebx.log4j.category.log.kernel	Logs for EBX® main features, processes, exceptions and compilation results of modules and data models.
ebx.log4j.category.log.workflow	Logs for main features, warnings and exceptions about workflow.
ebx.log4j.category.log.persistence	Logs related to communication with the underlying database.
ebx.log4j.category.log.setup	Logs for the compilation results of all EBX® objects, except for modules and data models.
ebx.log4j.category.log.validation	Logs for datasets validation results.
ebx.log4j.category.log.mail	Logs for the activity related to the emails sent by the server (see Activating and configuring SMTP and emails [p 586]). **Note:** This category must not use the Custom SMTP appender [p 585] in order to prevent infinite loops.
ebx.log4j.category.log.d3	Logs for D3 events on EBX®.
ebx.log4j.category.log.dataservices	Logs for data service events in EBX®.
ebx.log4j.category.log.monitoring	Raw logs for memory monitoring [p 615].
ebx.log4j.category.log.request	Logs for `Request`[API] and `Query`[API] events in EBX®: executions having a duration exceeding `ebx.logs.request.durationThreshold` milliseconds, requests / queries where the optimization phase exceeds `ebx.logs.request.optimizationThreshold` milliseconds and executions that internally throw an exception while iterating the results. All requests / queries are logged regardless of their duration, if log level is set to `DEBUG`.

ebx.log4j.category.log.restServices	Logs for REST services events in EBX®, including those from the REST Toolkit [p 1285].
ebx.log4j.category.log.audit	Logs for the audit trail [p 733] feature.
ebx.log4j.category.log.assistant	Logs for the assistant feature.

Some of these categories can also be written to through custom code using the `LoggingCategory`[API] interface.

```
 ################################################
## Log4J properties:
##
## We have some specific syntax extensions:
## - Appender ebxFile:<aFileName>
## Defines a file appender with default settings (threshold=DEBUG)
##
## - property log.defaultConversionPattern is set by Java
##
## Note: Apache Log4j 2 native configuration mechanisms may be used instead, or in combination,
## of these ones. For more information, see 'Logs configuration' section in
## 'Administration overview' chapter from EBX® documentation.
################################################
#ebx.log4j.debug=true
#ebx.log4j.disable=
ebx.log4j.rootCategory= INFO
ebx.log4j.category.log.kernel= INFO, Console, ebxFile:kernel, kernelMail
ebx.log4j.category.log.workflow= INFO, ebxFile:workflow
ebx.log4j.category.log.persistence= INFO, ebxFile:persistence
ebx.log4j.category.log.setup= INFO, Console, ebxFile:kernel
ebx.log4j.category.log.mail= INFO, Console, ebxFile:mail
ebx.log4j.category.log.frontEnd= INFO, Console, ebxFile:kernel
ebx.log4j.category.log.frontEnd.incomingRequest= INFO
ebx.log4j.category.log.frontEnd.requestHistory= INFO
ebx.log4j.category.log.frontEnd.UIComponentInput= INFO
ebx.log4j.category.log.fsm= INFO, Console, ebxFile:fsm
ebx.log4j.category.log.fsm.dispatch= INFO
ebx.log4j.category.log.fsm.pageHistory= INFO
ebx.log4j.category.log.wbp= FATAL, Console
#-------------------------------------------------
ebx.log4j.appender.Console.Threshold = INFO
ebx.log4j.appender.Console=com.onwbp.org.apache.log4j.ConsoleAppender
ebx.log4j.appender.Console.layout=com.onwbp.org.apache.log4j.PatternLayout
ebx.log4j.appender.Console.layout.ConversionPattern=${log.defaultConversionPattern}
#-------------------------------------------------
ebx.log4j.appender.kernelMail.Threshold = ERROR
ebx.log4j.appender.kernelMail = com.onwbp.org.apache.log4j.net.SMTPAppender
ebx.log4j.appender.kernelMail.To = admin@domain.com
ebx.log4j.appender.kernelMail.From=admin${ebx.site.name}
ebx.log4j.appender.kernelMail.Subject=EBX® Error on Site ${ebx.site.name} (VM ${ebx.vm.id})
ebx.log4j.appender.kernelMail.layout.ConversionPattern=**Site ${ebx.site.name} (VM${ebx.vm.id})**%n
${log.defaultConversionPattern}
ebx.log4j.appender.kernelMail.layout=com.onwbp.org.apache.log4j.PatternLayout
#-------------------------------------------------
ebx.log4j.category.log.monitoring=INFO, ebxFile:monitoring
ebx.log4j.category.log.dataServices=INFO, ebxFile:dataServices
ebx.log4j.category.log.d3=INFO, ebxFile:d3
ebx.log4j.category.log.request=INFO, ebxFile:request
ebx.log4j.category.log.restServices=INFO, ebxFile:dataServices
ebx.log4j.category.log.audit=INFO, ebxFile:audit
ebx.log4j.category.log.assistant=INFO, ebxFile:assistant
```

Custom 'ebxFile' appender

The token `ebxFile:` can be used as a shortcut to define a daily rolling file appender with default settings. It must be followed by a file name. It then activates an appender that writes to a file located in the directory `ebx.logs.directory`, with a threshold set to `DEBUG`.

The property `ebx.log4j.appender.ebxFile.backup.Threshold` allows defining the maximum size (in megabytes) of backup files for daily rollover.

```
################################################
```

```
## Directory of log files 'ebxFile:'
## This property is used by special appender prefixed
## by 'ebxFile:' (see log section below)
################################################
ebx.logs.directory=${ebx.home}/ebxLog

################################################
## Daily rollover threshold of log files 'ebxFile:'
## Specifies the maximum size (in megabytes) of backup files for daily rollover
## of 'ebxFile:' appenders.
## When set to a negative value, backup log files are never purged.
## Default value is -1.
################################################
ebx.log4j.appender.ebxFile.backup.Threshold=-1
```

Custom SMTP appender

The appender `com.onwbp.org.apache.log4j.net.SMTPAppender` provides an asynchronous email sender.

See also *Activating and configuring SMTP and emails* *[p 586]*

Custom module log threshold

By default, the log level threshold of the logging category associated with a custom module is set to `INFO`.

This threshold can be customized by setting the property `ebx.log4j.category.log.wbp.xxxxxx` for the custom module `xxxxxx`.

Example: `ebx.log4j.category.log.wbp.mycompany-module=DEBUG`.

See also *ModuleContextOnRepositoryStartup.getLoggingCategory*[API]

Add-on module log threshold

By default, the log level threshold of any add-on module is set to `INFO`.

The log level threshold can be customized by setting the property `ebx.log4j.category.log.addon.xxxxxx` for the add-on module `ebx-addon-xxxxxx`.

Example: `ebx.log4j.category.log.addon.daqa=DEBUG`

Migrate from EBX® main file mechanism to Apache Log4j™ 2 and reverse

Supposing that Apache Log4j™ 2 is not configured programmatically, then moving from one approach to another is as simple as cleaning the previously used file, converting every logger configuration into the new format and installing the target file at the proper place.

To define EBX® loggers configuration, refer to Configuring the EBX® logs [p 582].

To define Apache Log4j™ 2 loggers configuration, remove the *ebx.log4j.category* prefix from EBX® loggers' name, replace the *ebx.log4j.rootCategory* configuration by the root logger one and refer to Apache Log4j™ 2 configuration documentation for the remaining.

Considering the underlying EBX® logs configuration:

```
ebx.log4j.appender.Console=       com.onwbp.org.apache.log4j.ConsoleAppender
ebx.log4j.appender.Console.layout=com.onwbp.org.apache.log4j.PatternLayout
ebx.log4j.appender.Console.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss.SSS} %-5p %-6pid{[0]} --- [%15.15t]
 %-40.40c{39} : %m%n

ebx.log4j.category.log.kernel=  INFO, Console
ebx.log4j.rootCategory=         INFO, Console
```

The Apache Log4j™ 2 equivalent configuration will be as the following:

```
{
  "configuration": {
    "appenders": {
      "Console": {
        "name": "Console",
        "PatternLayout": {
          "pattern": "%d{yyyy-MM-dd HH:mm:ss.SSS} %-5p %-6pid{[0]} --- [%15.15t] %-40.40c{39} : %m%n"
        }
      }
    },
    "loggers": {
      "logger": [
        {
          "name": "log.kernel",
          "level": "INFO",
          "AppenderRef": {
            "ref": "Console"
          }
        }
      ],
      "root": {
        "level": "INFO",
        "AppenderRef": {
          "ref": "Console"
        }
      }
    }
  }
}
```

Note

The Custom 'ebxFile' appender [p 584] does not exist natively in Apache Log4j™ 2, but its behavior can be mimic.

Note

Ensure to migrate every logger from one mechanism to another to get as close functional coverage as possible.

103.12 Activating and configuring SMTP and emails

The internal mail manager sends emails asynchronously. It is used by the workflow engine and the custom SMTP appender `com.onwbp.org.apache.log4j.net.SMTPAppender`.

See also *Mail sessions [p 540]*

```
################################################
## SMTP and emails
################################################

## Activate emails (true or false, default is false).
## If activated, the deployer must ensure that the entry 'mail/EBX_MAIL_SESSION' is bound
## in the operational environment of the application server (except if a specific email
## configuration is used by setting the property ebx.mail.smtp.host below).
#ebx.mail.activate=false

## Polling interval is in seconds (default is 10).
#ebx.mail.polling.interval=10

## Specific properties to be set only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.mail.smtp.host = smtp.domain.com
## SMTP port default is 25.
#ebx.mail.smtp.port= 25
#ebx.mail.smtp.login=
#ebx.mail.smtp.password=
## SMTP socket connection timeout value in milliseconds (default is 600000).
#ebx.mail.smtp.connectionTimeout=600000
## SMTP socket read timeout value in milliseconds (default is 600000).
```

```
#ebx.mail.smtp.timeout=600000
## SMTP socket write timeout value in milliseconds (default is 600000).
#ebx.mail.smtp.writeTimeout=600000

## Activate SSL (true or false, default is false).
#ebx.mail.smtp.ssl.activate=true
```

103.13 **Configuring data services**

```
##################################################
## Data services
##################################################

## Specifies the default value of the data services parameter
## 'disableRedirectionToLastBroadcast'.
## Default is false.
#ebx.dataservices.disableRedirectionToLastBroadcast.default=false

## Specifies the default value for deletion at the end of close and
## merge operations.
## If the parameter is set in the request operation, it overrides
## this default setting.
## If unspecified, default is false.
#ebx.dataservices.dataDeletionOnCloseOrMerge.default=false
#ebx.dataservices.historyDeletionOnCloseOrMerge.default=false

## Specifies the default maximum pagination size value for the select
## operations. This configuration is used by SOAP and REST connectors.
## Default value is 10000, maximum recommended value is 100000
#ebx.dataservices.pagination.maxSize.default= 10000

## Specifies the default pagination size value for the select
## operations. This configuration is used by the SOAP connector.
## Default value is 10.
#ebx.dataservices.pagination.pageSize.default=10

## Upon WSDL generation, specifies if the target namespace value
## corresponds to the content before 5.5.0 'ebx-services'
## or 'urn:ebx:ebx-services' in conformity with the URI syntax.
## If the parameter is set to true, there is no check of the target
## namespace as URI at the WSDL generation.
## If unspecified, default is false.
#ebx.dataservices.wsdlTargetNamespace.disabledCheck=false

##################################################
## REST configuration
##################################################

## If activated, the HTTP request header 'Accept' is used to specify
## the accepted content type. If none is supported, an error is
## returned to the client with the HTTP code 406 'Not acceptable'.
## If deactivated, the header is ignored therefore the best content
## type is used.
## Default is false.
#ebx.dataservices.rest.request.checkAccept=false

## If activated, when a REST data service authentication negotiate fails,
## EBX response includes fallback to 'Basic' authentication method by setting
## the HTTP header 'WWW-Authenticate' to 'Basic'.
## Note: This property only activate/deactivate
## the authentication fallback.
## Default is false.
#ebx.dataservices.rest.auth.tryBasicAuthentication=false

## Authorization token timeout is seconds.
## Default value is 1800 seconds (30 minutes)
## This value is ignored if 'Token Authentication Scheme' is not activated.
#ebx.dataservices.rest.auth.token.timeout=1800

# Specifies the maximum size in KB of a response in BO select
# operations. This configuration is used by the REST connector.
# If exceeded, a partial response is returned with a limited number
# of business objects and the HTTP status code 206 Partial Content.
# Default value is 1024 (1MB)
# The minimum value is 56 which is the default buffer size of the
# JSON serializer
# The maximum value is 10MB.
#ebx.dataservices.rest.bo.maxResponseSizeInKB=1024
```

103.14 Activating and configuring JMS

See also *JMS for data services [p 541]*

```
################################################
## JMS configuration for Data Services
################################################

## Activates JMS (true or false, default is false).
## If activated, the deployer must ensure that the entry 'jms/EBX_JMSConnectionFactory'
## are bound in the operational environment of the application server.
## The entry 'jms/EBX_QueueIn' should also be bound to enable handling Data Services
## request using JMS.
#ebx.jms.activate=false

## Activates JMS queue for failures (true or false, default is false).
## If activated, the deployer must ensure that the entry 'jms/EBX_QueueFailure' is bound
## in the operational environment of the application server.
#ebx.jms.activate.queueFailure=false

## Number of concurrent listener(s)
## Default is 3.
## Property is used if ebx.jms.activate is set to true.
#ebx.jms.listeners.count=3
```

103.15 Configuring distributed data delivery (D3)

See Configuring D3 nodes [p 769] for the main configuration file properties pertaining to D3.

See also

JMS for distributed data delivery (D3) [p 759]

Introduction to D3 [p 750]

103.16 Configuring REST toolkit services

```
################################################
## REST configuration
################################################

## Defines the maximum number of bytes that will be extracted
## from the REST request body to build some DEBUG log messages.
## Default value is 8192 bytes.
## This value is ignored if DEBUG level is not activated on the restServices logger.
#ebx.restservices.log.body.content.extract.size=8192
```

103.17 Keystore sensitive property file location

Sensitive properties specify the location of the keystore file and other various information such as the secret key aliases and passwords.

```
#####################################################################
## Keystore sensitive property file.
#####################################################################
# Specifies the path of the file keystore.
# If the specified path is not absolute, it must be relative to ${ebx.repository.directory}.
# The default file keystore path is 'ebx-keystore-sensitive.properties'.
ebx.keystore.sensitive.properties.path=${ebx.repository.directory}/ebx-keystore-sensitive.properties
```

See also *Keystore file [p 716]*

103.18 **Activating and configuring staging**

Staging is enabled/disabled via configuration. Some options can be set so as to optimize memory and disk usage. The properties are configured as follows:

```
################################################
## Staging configuration
################################################

## Activates staging (true or false, default is true).
#ebx.staging.activated=true

## Defines the max size on disc
## dedicated to staging temporary folders.
## 0 defines an infinite size.
## If unset, the default value is 0.
#ebx.staging.maxTemporaryFolderSizeInBytes=0

## Max size for canonicalization of attachments in bytes.
## Canonicalization consumes memory and CPU on large xml and json for attachments.
## If not set, default value is 10 Mo (10485760).
#ebx.staging.maxCanonicalizationSizeInBytes=10485760

## Defines the maximum number of staging element per list when browsing a repository
## If not defined, the default value is 1000.
#ebx.staging.maxElementPerList=1000

## Defines the maximum number of staging element per selection operation when browsing a repository
## If not defined, the default value is 1000.
#ebx.staging.maxElementSelection=1000

# Defines a set of folder relative paths to ${ebx.repository.directory}/startup, separated by commas,
# used to configure the import of staging archives on startup.
# The folders must be strictly located in the startup folder with no deviations or shortcuts
# Each folder is a configuration of an import which must contain a file
# named "instructions.json" and the staging archives to import.
# Note that the instructions file can only mention archives strictly under the same folder
# and can not be on the root of (${ebx.repository.directory}/startup).
# In case of errors, an error.json report is added to the folder.
# In case of success, an output.json report is added to the folder
# and the folder would be ignored in the next startup.
# The content of the file is a json in the following format:
# {
#  "imports": [
#    {
#      "archiveFilePath": "myArchive1.zip"
#    },
#    {
#      "archiveFilePath": "myArchive2.zip",
#      "advanced_options": {
#        "forceUpdateOnIdenticalComponent": false,
#        "importDefinition": true
#      }
#    }
#  ]
#}
# [forceUpdateOnIdenticalComponent] boolean, when true, forces the update of a component even
# if it is the same in the repository.
# [importDefinition] boolean, when true, imports the domain definition in the staging domain
# configurations to be able to re-export the same domain.
#ebx.staging.importOnStartup=hr,catalog
```

See also *Staging* *[p 777]*

103.19 Configuring Web access from end-user browsers

HTTP Authorization header policy

EBX® natively offers three policies to send and receive credentials using HTTP headers:

`standard`	It corresponds to the authentication scheme, using the HTTP Authorization header, described in the RFC 2617.
`ebx`	To prevent HTTP Authorization header override issues, this policy acts the same as the `standard` but the credentials are stored in an EBX® specific HTTP header.
`both`	It is the combination of the two previously described policies.

```
################################################
## EBX® authorization header policy for HTTP requests
##
## Possible values are: standard, ebx, both.
##  standard:
##    the standard HTTP Authorization header holds the credentials
##  ebx:
##    an EBX® specific HTTP header holds the credentials
##  both:
##    both (standard and specific) HTTP headers hold the credentials
##
## Default value is: both.
################################################
#ebx.http.authorization.header.policy=both
```

URLs computing

By default, EBX® runs in "standalone" mode, where external resources (images, JavaScript, etc.) are provided by the application server.

Also by default, URL-related parameters in the main configuration file do not have to be set.

In this case, the server name and the port are obtained from the initial request sent to EBX®.

See also *URL policy (deprecated)* *[p 675]*

```
################################################
## EBX® FrontServlet: default properties for computing servlet address
##
## {useLocalUrl}:
## If set to true, servlet address is a "local absolute" URL.
## (that is, a relative URL consisting of an absolute path: "/path")
## See RFC 2396, http://www.ietf.org/rfc/rfc2396.txt).
## This property is defined once for HTTP and HTTPS.
## Default value is false.
##
## {host}:
## If neither defined nor adapted, retrieves initial request host
## {port}:
## If neither defined nor adapted, retrieves initial request host
## {path}:
## Mandatory, may be empty
## {ui.path}:
## If not defined, defaults to ebx-ui/
## {http.useHttpsSettings}:
## If true, force the use of SSL security even if the incoming requests do not
## {authentication.redirectToHttps}
## If set to true, an HTTP request to the login form is automatically redirected to force it
```

```
## to HTTPS.
## Default is false.
## Usually, this property should be set to false if EBX® is behind a reverse proxy or a firewall
## that takes care of HTTPS encryption.
##
## Resulting address will be:
## EBX®: protocol://{host}:{port}/{path}
## UI: protocol://{host}:{port}/{ui.path}
##
## Each property for HTTP (except {port}) may be inherited from HTTPS property,
## and reciprocally.
################################################

ebx.servlet.useLocalUrl=true

#ebx.servlet.http.host=
#ebx.servlet.http.port=
ebx.servlet.http.path=ebx/
#ebx.servlet.http.ui.path=ebx-ui/
#ebx.servlet.http.authentication.path=ebx-authentication/
#ebx.servlet.http.authentication.redirectToHttps=false
#ebx.servlet.http.useHttpsSettings=false

#ebx.servlet.https.host=
#ebx.servlet.https.port=
ebx.servlet.https.path=ebx/
#ebx.servlet.https.ui.path=ebx-ui/
#ebx.servlet.https.authentication.path=ebx-authentication/

################################################
## External resources: default properties for computing external resources address
##
## The same rules apply as EBX® FrontServlet properties (see comments).
##
## Each property may be inherited from EBX® FrontServlet.
################################################

ebx.externalResources.useLocalUrl=true

#ebx.externalResources.http.host=
#ebx.externalResources.http.port=
#ebx.externalResources.http.path=
#ebx.externalResources.http.useHttpsSettings=false

#ebx.externalResources.https.host=
#ebx.externalResources.https.port=
#ebx.externalResources.https.path=
```

Proxy mode

Proxy mode allows using a front-end HTTP server to provide static resources (images, CSS, JavaScript, etc.). This architecture reduces the load on the application server for static HTTP requests. This configuration also allows using SSL security on the front-end server.

The web server sends requests to the application server according to a path in the URL. The `servletAlias` and `uiServletAlias` paths are specified in the main configuration file.

The web server provides all external resources. These resources are stored in a dedicated directory, accessible using the `resourcesAlias` path.

EBX® must also be able to access external resources from the file system. To do so, the property `ebx.webapps.directory.externalResources` must be specified.

The incoming requests are always assumed to be HTTPS if `ebx.servlet.http.useHttpsSettings` and / or `ebx.externalResources.http.useHttpsSettings` properties are set to true. Their default values are `false`. These properties are useful if EBX® is behind a reverse proxy or a firewall that takes care of HTTPS encryption.

The main configuration file may be configured as follows:

```
################################################
## Path for external resources if they are not
## delivered within web applications
## This field is mandatory if in proxy mode.
################################################
ebx.webapps.directory.externalResources=D:/http/resourcesFolder
```

```
################################################

ebx.servlet.useLocalUrl=true

#ebx.servlet.http.host=
#ebx.servlet.http.port=
ebx.servlet.http.path=servletAlias
ebx.servlet.http.ui.path=uiServletAlias
ebx.servlet.http.authentication.path=authenticationServletAlias
#ebx.servlet.http.authentication.redirectToHttps=false
#ebx.servlet.http.useHttpsSettings=false

#ebx.servlet.https.host=
#ebx.servlet.https.port=
ebx.servlet.https.path=servletAlias
ebx.servlet.https.ui.path=uiServletAlias
ebx.servlet.https.authentication.path=authenticationServletAlias

################################################

ebx.externalResources.useLocalUrl=true

#ebx.externalResources.http.host=
#ebx.externalResources.http.port=
ebx.externalResources.http.path=resourcesAlias
#ebx.externalResources.http.useHttpsSettings=false

#ebx.externalResources.https.host=
#ebx.externalResources.https.port=
ebx.externalResources.https.path=resourcesAlias
```

Attention

When proxy mode is used, the URL to the `ebx-dataservices` module must be configured through the lineage administration panel. Note that the provided URL must end its path with `/ebx-dataservices`.

Reverse-proxy mode

If URLs generated by EBX®, for requests and external resources, must contain a different protocol than the one from the incoming request, a specific server name, a specific port number or a specific path prefix, properties may be configured as follows:

```
################################################
#ebx.servlet.useLocalUrl=false

ebx.servlet.http.host=reverseDomain
#ebx.servlet.http.port=
ebx.servlet.http.path=ebx/
#ebx.servlet.http.ui.path=ebx-ui/
#ebx.servlet.http.authentication.path=ebx-authentication/
#ebx.servlet.http.authentication.redirectToHttps=false
#ebx.servlet.http.useHttpsSettings=false

ebx.servlet.https.host=reverseDomain
#ebx.servlet.https.port=
ebx.servlet.https.path=ebx/
#ebx.servlet.https.ui.path=ebx-ui/
#ebx.servlet.https.authentication.path=ebx-authentication/

################################################
## Web parameters (for external resources)
## if nothing is set, values are taken from servlet.
################################################
#ebx.externalResources.useLocalUrl=false

#ebx.externalResources.http.host=
#ebx.externalResources.http.port=
#ebx.externalResources.http.path=
ebx.externalResources.http.useHttpsSettings=true

ebx.externalResources.https.host=reverseDomain
#ebx.externalResources.https.port=
```

```
ebx.externalResources.https.path=
```

Attention

When reverse-proxy mode is used, the URL to the `ebx-dataservices` module must be configured through the lineage administration panel. Note that the provided URL must end its path with `/ebx-dataservices`.

103.20 **Security configuration**

These parameters are used to configure the external pages that EBX® can redirect the user to, as well as some options regarding the authorization cookie.

```
# Custom login page URL.
# The login page is displayed when an unidentified user accesses EBX.
# If no url is specified, a built-in page will be displayed.
#ebx.security.loginPage.url=

# When using a custom login page, should a 'resume' parameter be appended to the URL?
# Default value is: true
#ebx.security.loginPage.appendResumeUrl=true

# Custom exit page URL.
# The exit page is displayed when a user exits EBX normally (when logging out for instance)
# and, if the 'exit error' page is not defined, when the user exits EBX because of an error.
# If no url is specified, a built-in page will be displayed.
#ebx.security.exitPage.url=

# When using a custom exit page, should a 'resume' parameter be appended to the URL?
# Default value is: true
#ebx.security.exitPage.appendResumeUrl=true

# Custom 'exit error' page URL.
# The 'exit error' page is displayed when a user exits EBX because of an error.
# If no url is specified, the exit page will be displayed.
#ebx.security.exitErrorPage.url=

# When using a custom 'exit error' page, should a 'resume' parameter be appended to the URL?
# Default value is: true
#ebx.security.exitErrorPage.appendResumeUrl=true

# Custom 'access denied' page URL.
# The 'access denied' page is displayed when an exception is thrown when authenticating a user.
# If no url is specified, a built-in page will be displayed.
#ebx.security.accessDeniedPage.url=

# Custom timeout page URL.
# The timeout page is displayed when a user accesses EBX with an outdated session.
# If no url is specified, the login page will be displayed.
#ebx.security.timeoutPage.url=

# List of authorized domains to redirect to.
# This is a list of comma-separated URLs which has as many items as 'ebx.security.exitRestrictions.httpsOnly'.
# For each item in this list, there should be a related item in 'ebx.security.exitRestrictions.httpsOnly'.
# Example: ebx.security.exitRestrictions.domains=my-domain.com,safe-domain.com,trusted-domain.com
#ebx.security.exitRestrictions.domains=

# For each domain declared in 'ebx.security.exitRestrictions.domains', indicates if only the https protocol is
 allowed.
# This is a list of comma-separated booleans which has as many items as 'ebx.security.exitRestrictions.domains'.
# For each item in this list, there should be a related item in 'ebx.security.exitRestrictions.domains'.
# Example: ebx.security.exitRestrictions.httpsOnly=true,false,false
#ebx.security.exitRestrictions.httpsOnly=

# Indicates if the authorization cookie should have the 'Secure' attribute.
# Accepted values are:
#   - true
#   - false
#   - auto: depends on the of value of 'ebx.https.support' and the protocol of the incoming request
# Default value is: auto
#ebx.security.authorizationCookie.attribute.secure=auto

# Indicates the value of the 'SameSite' attribute of the authorization cookie.
# Accepted values are:
#   - Strict
#   - Lax
```

```
#   - None
# Default value is: Strict
#ebx.security.authorizationCookie.attribute.sameSite=Strict
```

These parameters are used to add rules to add to the CSP exclusions.

```
# Raw value to add at the end of the 'script-src' CSP value.
# Example: ebx.security.csp.script-src=https://my-domain.com
#ebx.security.csp.script-src=

# Raw value to add at the end of the 'style-src' CSP value.
# Example: ebx.security.csp.style-src=https://my-domain.com
#ebx.security.csp.style-src=

# Raw value to add at the end of the 'img-src' CSP value.
# Example: ebx.security.csp.img-src=https://my-domain.com
#ebx.security.csp.img-src=
```

103.21 Configuring failover

These parameters are used to configure the failover mode and activation key, as well as heartbeat logging in `DEBUG` mode.

See also *Failover with hot-standby* *[p 661]*

```
################################################
## Mode used to qualify the way in which a server accesses the repository.
## Possible values are: unique, failovermain, failoverstandby.
## Default value is: unique.
################################################
#ebx.repository.ownership.mode=unique

## Activation key used in case of failover. The backup server must include this
## key in the HTTP request used to transfer exclusive ownership of the repository.
## The activation key must be an alphanumeric ASCII string longer than 8 characters.
#ebx.repository.ownership.activationkey=

## Specifies whether to hide heartbeat logging in DEBUG mode.
## Default value is true.
#ebx.repository.ownership.hideHeartBeatLogForDebug=true
```

103.22 Tuning the EBX® repository

Some options can be set so as to optimize memory usage.

The properties are configured as follows:

```
################################################
## Technical parameters for memory and performance tuning
################################################
## Import commit threshold allows to specify the commit threshold
## exclusively for the archive import launched directly from Manager.
##
## For more details about the commit threshold,
## see the JavaDoc ProcedureContext.setCommitThreshold().
## Default value is 0.
ebx.manager.import.commit.threshold=100
```

See also *Validation report page* *[p 601]*

103.23 Advanced index configuration

Load index at startup

Indexes can load during EBX® startup to speed up initial table access. Indexes are rebuilt if they are missing (`open` rebuild strategy), or if the index does not match the latest model (`update` rebuild strategy).

The properties are configured as follows:

```
# load index at startup. set level to 1 to preload Reference dataspace, 2 to preload Reference dataspace and all
 it's direct
# children, 3 to preload Reference dataspace and all their children and grandchildren.
# Use higher values to go deeper in dataspace tree but this could be time-consuming.
ebx.index.loader.level=0
# Index load strategy
# open: opens the index as it is stored; the index cannot match the latest model
# update: rebuilds the index to match the latest model
# rebuild: always rebuilds the index. This option is only usable during boot and will block the boot until all
 indexes are rebuilt
ebx.index.loader.strategy=open
# Block the boot until all indexes are loaded
ebx.index.loader.waitOnBoot=false
# Include the technical dataspace in preload. Unset this only if something goes wrong during boot with loading
 the technical dataspace.
# This value has no effect if ebx.index.loader.level is set to less than 2
ebx.index.loader.includeTechnical=true
# Include / exclude a rule for index selection
# Each rule is a JSON object, or an array of JSON objects defining regular expressions on a dataspace, dataset,
 and table.
# Include rules are tested before exclude rules.
# Default rules for include are accept all. The exclude default rule is exclude nothing.
ebx.index.loader.include={"dataspace":".*","dataset":".*","table":".*"}
ebx.index.loader.exclude=
```

Attention

The `rebuild` strategy must be used with extreme caution. It forces a full rebuild of each selected index at startup and wait for the completion of all rebuild. This mode should only be used when instructed by customer support and reverted to `open` or `update` after restart.

103.24 Miscellaneous

Activating data workflows

This parameter specifies whether data workflows are activated. This parameter is not taken into account on the fly. The server must be restarted whenever the value changes.

```
################################################
## Workflow activation.
## Default is true.
################################################
ebx.workflow.activation = true
```

Disabling user task legacy mode

This parameter specifies whether the creation service of a user task in legacy mode should be offered in the workflow modeling. The default value is `false`.

See `UserTask.UserTaskMode.LEGACY_MODE`[API] for more information.

```
################################################
## Disables legacy work item mode (default is false)
## Specify if the creation service of user task in legacy mode must be offered
```

```
## in workflow modeling.
################################################
#ebx.manager.workflow.legacy.userTaskMode=true
```

Disabling hierarchy plan view

This parameter specifies whether the hierarchy plan view is hidden. The default value is `true`.

```
################################################
## Activate or deactivate Workflow hierarchy plan view
################################################
ebx.manager.workflow.hierarchyPlanView.hidden=false
```

Activating delegation to launch a workflow by API

This parameter activates delegation to launch a workflow by API. The default value is `false` which means that permissions will be always checked unless this parameter is activated.

```
################################################
## Activate delegation to launch workflows by API
################################################
ebx.workflow.api.processLauncher.permissionCheckDelegated=true
```

Log procedure starts

This parameter specifies whether starts of the procedure execution are logged.

```
################################################
## Specifies whether transaction starts are logged. Default is false.
################################################
ebx.logs.logTransactionStart = true
```

Log validation starts

This parameter specifies whether starts of datasets validation are logged.

```
################################################
## Specifies whether validation starts are logged. Default is false.
################################################
ebx.logs.logValidationStart = true
```

Request cache activation

This parameter specifies whether or not to activate the cache for the internal request-to-query conversions that are required by the query engine when a `Request`[API] is executed. The activation of this cache will improve the performance on certain types of requests, especially if they are executed multiple times.

```
# Activates the request cache, which can improve the performance of certain requests. Possible values are
 DISABLED,
# WITH_CHECK (ensures that a cached request executes the same query plan as its non-cached version, recommended),
# EXCEPTION (throws an exception if the query plan for a request differs from its non-cached version), and
 ENABLED.
# The default value is WITH_CHECK.
#ebx.cache.request.mode=WITH_CHECK
```

Request duration threshold for logs

This parameter specifies in milliseconds the threshold of duration of `Request`[API] and `Query`[API] to be logged. Logs are generated if logging category `ebx.log4j.category.log.request` level is not higher than `INFO`. If the level is `DEBUG`, all `Request`[API] and `Query`[API] are logged.

```
################################################
## Specifies in milliseconds the threshold of duration of Requests and Queries
## to be logged
```

```
## Default value is 1000 ms.
## If unset, the default value is used.
##################################################
#ebx.logs.request.durationThreshold=1000
```

Request repetition threshold for logs

This parameter specifies in milliseconds the delay between 2 logs for `Request`[API] and `Query`[API] that goes beyond the threshold of duration. If this value is greater than 0, and the query duration goes beyond the threshold of duration, it will be logged again repeatedly with at least this delay between each log. As log messages include duration, this is useful to track long queries duration.

```
##################################################
## Specifies in milliseconds the delay between two log entries for Requests and
## Queries that goes beyond the threshold of duration. If this value is greater
## than 0, and the query duration goes beyond the threshold of duration, it will
## be logged again repeatedly with at least this delay between each log.
## Default value is 30000 ms.
##################################################
#ebx.logs.request.logAgainEvery=30000
```

Request optimization threshold for logs

This parameter specifies in milliseconds the threshold of the optimization phase of `Request`[API] and `Query`[API] to be logged. Logs are generated if logging category `ebx.log4j.category.log.request` level is not higher than `INFO`.

```
##################################################
## Specifies in milliseconds the threshold of optimization of Requests and Queries
## to be logged. Default value is 1000 ms.
##################################################
#ebx.logs.request.optimizationThreshold=1000
```

Request SQL representation for logs

This parameter specifies if the logs for a certain `Request`[API] shall include the corresponding SQL representation of the request. Notice that this conversion takes a toll in terms of performance, and not all requests can be converted into SQL. The default value is `false`.

```
##################################################
## Specifies if the logs of requests shall include the SQL representation.
## Default is false.
##################################################
#ebx.log4j.category.log.request.enableSQLConversion=false
```

Request optimization for sorting a table by foreign key labels

This parameter specifies whether the query optimizer rewrites certain combinations of sort and join operations to a specialized operator that carries out the join(s) in a sorted manner.

```
##################################################
## Properties to activate/deactivate EnumerableSortedNestedLoopLeft/Right operators,
## which can improve the performance of requests sorted by foreign key labels.
## Default value is true.
##################################################
#ebx.query.sortedNestedLoopLeft.enabled=true
#ebx.query.sortedNestedLoopRight.enabled=true
```

Deployment site identification

This parameter allows specifying the email address to which technical log emails are sent.

```
##################################################
## Unique Site Name
## --> used by monitoring emails and by the repository
##################################################
```

```
ebx.site.name= name@domain.com
```

Dynamically reloading the main configuration

Some parameters can be dynamically reloaded, without restarting EBX®. The parameter `thisfile.checks.intervalInSeconds` indicates how frequently the main configuration file is checked.

```
################################################
### Checks if this file has been updated
### If value <= 0, no more checks will be done
################################################
thisfile.checks.intervalInSeconds=1
```

In development mode, this parameter can be set to as low as one second. On production systems, where changes are expected to be less frequent, the value can be greater, or set to '0' to disable hot reloading entirely.

This property is not always supported when the module is deployed as a WAR, as it would then depend on the application server.

Declaring modules as undeployed

On application server startup, the initialization of deployed web applications / EBX® modules and the initialization of the EBX® repository are performed asynchronously. In order to properly initialize the EBX® repository, it is necessary to compile all the data models used by at least a dataset, hence EBX® will wait endlessly for referenced modules to be registered.

If a module is referenced by a data model but is not deployed (or no longer deployed), it is necessary to declare this module as undeployed to unlock the wait and continue the startup process.

> **Note**
>
> The `kernel` logging category indicates which modules are awaited.

> **Note**
>
> A module declared as undeployed cannot be registered into EBX® until it is removed from the property `ebx.module.undeployedModules`.

> **Note**
>
> Any data model based on an unregistered module will have an "undeployed module" compilation error.

See also

Module registration *[p 880]*

Dynamically reloading the main configuration *[p 598]*

```
################################################
## Comma-separated list of EBX® modules declared
## as undeployed.
## If a module is expected by the EBX® repository but is
## not deployed, it must be declared in this property.
## Caution:
## if the "thisfile.checks.intervalInSeconds" property is deactivated,
## a restart is mandatory, otherwise it will be hot-reloaded.
################################################
ebx.module.undeployedModules=
```

Module public path prefix

EBX® modules' public paths are declared in the 'module.xml' file of each module. A context prefix can be declared for all modules, without having to modify the 'module.xml' content, by specifying the property that follows.

This prefix will apply to any EBX® module, including core, add-on and specific modules.

When proxy and / or reverse-proxy mode are used, the `ebx.servlet.http[s].path` and `ebx.servlet.http[s].ui.path` properties must take into account this module public path prefix setting. Conversely, the `ebx.externalResources.http[s].path` property must end its path just before a potential prefix.

```
################################################
ebx.servlet.useLocalUrl=true

#ebx.servlet.http.host=
#ebx.servlet.http.port=
ebx.servlet.http.path=reverse-proxy/prefix/ebx/
ebx.servlet.http.ui.path=reverse-proxy/prefix/ebx-ui/
ebx.servlet.http.authentication.path=reverse-proxy/prefix/ebx-authentication/
#ebx.servlet.http.authentication.redirectToHttps=false
#ebx.servlet.http.useHttpsSettings=false

#ebx.servlet.https.host=
#ebx.servlet.https.port=
ebx.servlet.https.path=reverse-proxy/prefix/ebx/
ebx.servlet.https.ui.path=reverse-proxy/prefix/ebx-ui/
ebx.servlet.https.authentication.path=reverse-proxy/prefix/ebx-authentication/

################################################
## Web parameters (for external resources)
## if nothing is set, values are taken from servlet.
################################################
ebx.externalResources.useLocalUrl=true

#ebx.externalResources.http.host=
#ebx.externalResources.http.port=
ebx.externalResources.http.path=reverse-proxy/
#ebx.externalResources.http.useHttpsSettings=false

#ebx.externalResources.https.host=
#ebx.externalResources.https.port=
ebx.externalResources.https.path=reverse-proxy/

################################################
## EBX® Module context path prefix
##
## If defined, applies to all EBX® modules public paths declared in
## any module.xml file (core, add-on and specific).
################################################
ebx.module.publicPath.prefix=prefix/
```

See URLs computing [p 590] for more information.

EBX® run mode

This property defines how EBX® runs. Three run modes are available: *development*, *integration* and *production*.

When running in *development* mode, the development tools [p 891] are activated in EBX®, some features thus become fully accessible and more technical information is displayed.

> **Note**
>
> The administrator can always access this information regardless of the mode used.

The additional features accessible when running in *development* mode include the following (non-exhaustive list):

Documentation pane	In the case of a computed value, the Java class name is displayed. A button is displayed giving access to the path to a node.
Compilation information	Module and schema compilation information is displayed in the dataset validation report.
Java bindings	The generation of Java bindings is available if the schema of the dataset mentions at least one binding.
Web component link generator	The Web component link generator is available on datasets and dataspaces.
Data model assistant	Data model configuration and additional options, such as Services, Business Objects and Rules, Java Bindings, Toolbars and some advanced properties.
Workflow modeling	Declare specific script tasks.
Log	The logs include additional technical information intended for the developer. For example, a warning is written to logs if a drop-down list is defined on a node which is not an enumeration in a UI Bean.
Product documentation	The product documentation is always the most complete one (i.e "advanced"), including administration and development chapters.

```
################################################
## Server Mode
## Value must be one of: development, integration, production
## Default is production.
################################################
backend.mode=integration
```

> **Note**
>
> There is no difference between the *integration* and *production* modes.

Resource filtering

This property allows the filtering of certain files and directories in the resource directory contents (resource type node, with an associated facet that indicates the directory that contains usable resources).

```
################################################
## list (separated by comma) of regexps excluding resource
## the regexp can be of type [pattern] or "m:[pattern]:".
## the list can be void
################################################
```

```
ebx.resource.exclude=CVS/*
```

HTML headers

This property allows to declare additional javascript files in the HTML pages.

```
################################################
## HTML header properties
##
################################################
## Specifies the javascript files to add in the HTML header.
##
## The first property, 'ebx.html.header.javascript', is a declaration of names, which will be used in the details
 properties.
## Example: ebx.html.header.javascript=example,other
##
## The details properties are formed as follow: 'ebx.html.header.javascript.[name].[property]'.
## Example: ebx.html.header.javascript.example.src=/my-module/www/common/my-script.js
##
## Detail property: 'src'.
## Contains the URL to the javascript file.
## Relative URLs are accepted, start by a "/".
## This property is mandatory.
## Example: ebx.html.header.javascript.example.src=/my-module/www/common/my-script.js
##
## Detail property: 'async'.
## Indicates if the associated javascript file should be loaded asynchronously.
## Accepted values: 'true', 'false'.
## This property is optional.
## Default value is 'false'.
## Example: ebx.html.header.javascript.example.async=true
##
## Full example:
## ebx.html.header.javascript=example,other
## ebx.html.header.javascript.example.src=/my-module/www/common/my-script.js
## ebx.html.header.javascript.example.async=true
## ebx.html.header.javascript.other.src=http://my-domain.com/other-script.js
#ebx.html.header.javascript=
```

Validation report page

The validation report page can display a finite number of items for each severity. This number can be tuned with this property.

```
################################################
## Defines the maximum item displayed for each severity in the validation report page.
## Default value is 100.
################################################
ebx.validation.report.maxItemDisplayed=200
```

See also *Tuning the EBX® repository* *[p 594]*

Validation report logs

This property allows to specify the number of validation messages to display in the logs when validating a dataset or a table.

```
################################################
## Defines the maximum number of messages displayed in the logs.
## Default value is 100.
## When set to 0 or a negative value, the limit is not considered.
################################################
ebx.validation.report.maxItemDisplayedInLogs=500
```

By default, the content of the validation of a dataset or a table is logged. Logging the content of the validation reports can be deactivated using the following property:

```
################################################
## Specifies whether the content of validation reports is logged when validating
## a dataset or a table.
## Default is true.
################################################
ebx.validation.report.logContent=true
```

See also *Tuning the EBX® repository* *[p 594]*

Validation mode of child datasets

This property influences how the algorithm for validating a child dataset behaves. It only affects the performance. The contents of the resulting validation reports are independent of the property.

```
# Changes the way how a child dataset is validated. This affects only the performance of the validation,
# but not the content of the resulting validation report. In the FULL mode, the report of the tables
# will be cleared, then the tables will be validated as if performing an initial validation.
# In the DIFF mode, the algorithm calculates the differences since the last validation
# and processes them to perform the necessary updates on the validation report.
# Possible values are: FULL, DIFF.
# The default is DIFF.
#ebx.validation.childDataset.mode=DIFF
```

APV should store invalid input in DataContext

Specifies the behavior of APV when the user enters a value that has validation errors. If true, the value will be stored in the ValueContext. If false, the value will not be stored in the ValueContext. The previous value will be kept. The default value is false.

```
# Specifies the behavior of APV when the user enters a value that has validation errors.
# If true, the value will be stored in the ValueContext.
# If false, the value will not be stored in the ValueContext. The previous value will be kept.
# The default value is false.
#ebx.manager.apv.storeInvalidInputInDataContext=false
```

CHAPTER 104

Initialization and first-launch assistant

Deliverables can be found on TIBCO eDelivery (an account is mandatory in order to access eDelivery, please contact the support team to request one).

The TIBCO EBX® Configuration Assistant helps with the initial configuration of the EBX® repository. If EBX® does not have a repository installed upon startup and if the automatic installation [p 573] is not enabled, the configuration assistant is launched automatically.

Before starting the configuration of the repository, make sure that EBX® is correctly deployed on the application server. See Jakarta EE deployment [p 533].

> **Note**
>
> The EBX® main configuration file must also be properly configured. See TIBCO EBX® main configuration file [p 571].

This chapter contains the following topics:

1. License key
2. Configuration steps

104.1 License key

When launching EBX®, the license key page displays automatically if no valid license key is available, that is, if there is no license key entered in the EBX® main configuration file, or if the current license key has expired.

The license key can be retrieved from the TIBCO eDelivery site (an account is mandatory in order to access eDelivery, please contact https://support.tibco.com to request one).

104.2 Configuration steps

The EBX® configuration assistant guides you through the following steps:

1. Validating the license agreement.
2. Configuring the repository.
3. Defining users in the default user and roles directory (if a custom directory is not defined).

4. Validating the information entered.
5. Installing the EBX® repository.

Validating the license agreement

In order to proceed with the configuration, you must read and accept the product license agreement.

Configuring the repository

This page displays some of the properties defined in the EBX® main configuration file. You also define several basic properties of the repository in this step.

Id of the repository (`repositoryId`)	Must uniquely identify the repository (in the scope of the enterprise). The identifier is 48 bits (6 bytes) long and is usually represented as 12 hexadecimal digits. This information is used for generating the Universally Unique Identifiers (UUIDs) of entities created in the repository, and also of transactions logged in the history. This identifier acts as the "UUID node", as specified by RFC 4122.
Repository label	Defines a user-friendly label that indicates the purpose and context of the repository.

See also *TIBCO EBX® main configuration file* *[p 571]*

Defining users in the default directory

If a custom user and roles directory is not defined in the EBX® main configuration file, the configuration assistant allows to define default users for the default user and roles directory.

An administrator user must be defined. You may optionally create a second user.

See also *Users and roles directory* *[p 703]*

Validating the information entered

Before proceeding with the installation of the repository, you can review the configuration of the repository and the information entered on the 'Configuration Summary' page. If you need to modify information, you can return to the previous pages using the configuration assistant **< Back** button.

Once you have verified the configuration, click the button **Install the repository >** to proceed with the installation.

Installing the EBX® repository

The repository installation is performed using the provided information. When the installation is complete, you are redirected to the repository login page.

CHAPTER 105

Managing TIBCO EBX® add-ons

This chapter contains the following topics:

1. Overview
2. Add-on compatibility
3. Deploying an add-on module
4. Registering an add-on module
5. Activating an add-on module
6. Deleting an add-on module

105.1 Overview

The following sections cover:

- Add-on compatibility [p 605]
- Deploying an add-on module [p 606]
- Registering an add-on module [p 606]
- Activating an add-on module [p 606]
- Deleting an add-on module [p 607]

Refer to each add-on's documentation for instructions on configuring and using the add-on.

105.2 Add-on compatibility

Each Add-on Bundle is compatible with specific versions and fix releases of EBX®. However, beginning with version 6.1.0, the major and minor versions of add-ons and EBX® must be the same. For example, EBX® version 6.1.0 is only compatible with add-ons that are also version 6.1.0. If this is not the case, add-on registration is aborted. For other releases, the *TIBCO EBX® Add-ons Versioning and Packaging Guide* includes a version compatibility matrix. Go to the official documentation site and use the **Bundle version** menu to view the version of this guide that corresponds with your Add-on Bundle.

105.3 Deploying an add-on module

Copy the add-on common JAR file (named `lib/ebx-addons.jar`) in the EBX® class-path; it must be accessible from the ebx.jar class-loader.

Copy the add-on common WAR file (named `wars/ebx-addon-common.war`) in the application server webapps folder or EAR as EBX® built-in web applications.

If an add-on is used, copy the EBX® add-on WAR file (named `wars/ebx-addon-<name>.war`) in the same folder or EAR as EBX® built-in web applications.

See Web applications [p 537], Deployment details [p 539] and Installation notes [p 542] for more information.

> **Note**
>
> The add-on log level can be managed in the main configuration file [p 585].

105.4 Registering an add-on module

Registering an add-on makes its configuration available in the **Administration** panel. Add-on features are only available to end-users when the add-on is also activated [p 606].

EBX® add-ons are automatically registered and enabled in the repository. To configure whether an add-on is active:

1. Navigate to the **Administration** panel.
2. Click the down-arrow in the navigation pane and select **Technical configuration** > **Add-ons registration**.
3. On the **Registered add-ons** page, click the **add-on** record you want to configure, and make any desired changes.
4. Select the add-on you are registering.
5. Click **Save**.

> **Note**
>
> Unregistering an add-on will not delete any existing configuration, but will make it available in the UI until the add-on is registered again.

105.5 Activating an add-on module

Activating an add-on makes its features available to the end-users. Only registered add-ons can be activated.

To activate an EBX® add-on in the repository:

1. Navigate to the 'Administration' area.
2. Click the down-arrow in the navigation pane and select **Technical configuration** > **Add-ons registration**.
3. Select the registered add-on you are activating and enable the 'Activation' field.
4. Click on **Save**.

105.6 Deleting an add-on module

To delete an add-on module from the EBX® repository:

1. Navigate to the **Administration** panel.
2. Click the down-arrow in the navigation pane and select *Technical configuration* > *Add-ons registration*.
3. On the **Registered add-ons** page, tick the box corresponding to the add-on you want to delete.
4. In the **Actions** menu, select **Delete**.
5. Stop the application server.
6. Add the deleted add-on's module name (ebx-addon-<name>) to the `ebx.module.undeployedModules` property in your `ebx.properties` file. See declaring modules as undeployed [p 598] for more information.
7. Remove the add-on's WAR file (ebx-addon-<name>.war) from the application server's `webapps` folder, or its EAR file if used as an EBX® built-in web application.
8. Restart the application server.
9. Close and purge the Administration datasets and dataspaces related to the deleted add-on.

   When an add-on is no longer deployed, a dataspace corresponding to the Administration dataset displays in the list of Reference children under the dataspaces. This makes it necessary to close/delete and manually purge all data/dataspaces related to the add-on.

CHAPTER 106

User authentication

This chapter contains the following topics:

1. The /ebx-authentication servlet
2. The authorization token
3. Customizing the authentication process
4. Single Sign-On (SSO)

106.1 The /ebx-authentication servlet

The /ebx-authentication servlet is the central point handling the user authentication in EBX®. Its role is to:

- Create an authorization token [p 609] when the user logs in for the first time.
- Revoke the authorization token [p 609] when the user logs out.
- Redirect the authorized user to the appropriate page.

Furthermore, depending on the configuration, it can also:

- Handle the login process by displaying a login screen.
- Redirect the unauthorized user to a custom login screen [p 610].
- Redirect the user logging out to a custom exit page.

106.2 The authorization token

When the /ebx-authentication servlet successfully authenticates a user, it creates a token containing the authentication information and stores it in a session cookie.

This means that the authorization is shared between all the browser tabs:

- If the user is already authenticated on a browser tab and opens EBX® on another tab, it will not be asked to authenticate again.
- It is not possible to be logged with different users in different tabs of the same browser.

Cookie properties

The cookie containing the token has the following properties:

`HttpOnly`	Not customizable. The cookie can't be read by javascript code in the browser.
`Secure`	Customizable via the property `ebx.security.authorizationCookie.attribute.secure` in `ebx.properties`. Defines if the cookie can be sent over HTTPS only, or if it can also be sent over HTTP.
`SameSite`	Customizable via the property `ebx.security.authorizationCookie.attribute.sameSite` in `ebx.properties`. Defines if the cookie can be sent when browsing from an external site.

See also *Security configuration [p 593]*

106.3 Customizing the authentication process

By default, EBX® uses a built-in directory and built-in login page. It is strongly recommended to replace the built-in directory by a custom one.

Customizing the directory

While EBX® provides a built-in directory, it is strongly recommended to replace it:

1. Create a class overriding `Directory`[API].
2. Create a class overriding `DirectoryFactory`[API].
3. Use the property `ebx.directory.factory` in `ebx.properties` to declare the factory.

See also *Security configuration [p 593]*

Customizing the login page

While EBX® provides a built-in login page, it is possible to replace it. To do so, use the property `ebx.security.loginPage.url` in `ebx.properties` to declare the custom login page.

It is up to the custom login page to authenticate the user. Once the user is authenticated, he should be redirected to the `/ebx-authentication/login` page. Then, EBX® will call the `Directory.authenticateUserFromHttpRequest`[API] method before creating an authorization token. The following points are required for this to work:

- The request pointing to `/ebx-authentication/login` should contain enough information to authenticate the user, for instance in a cookie, a HTTP header or a query parameter.

- The directory must be overridden (see Customizing the directory [p 610]) and the implementation of the `Directory.authenticateUserFromHttpRequest`[API] method should read the information from the incoming request in order to return the appropriate `UserReference`[API].

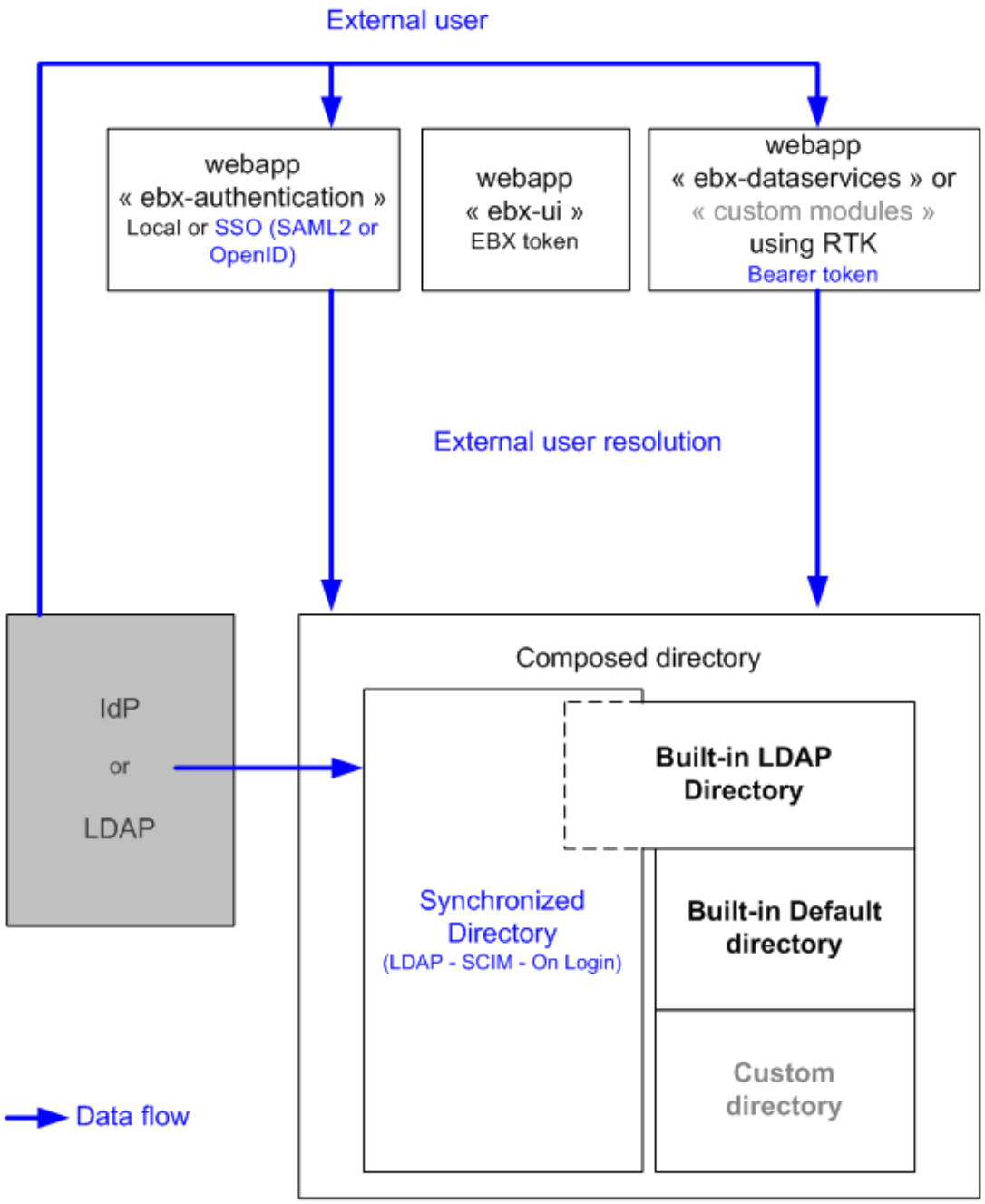
> **Note**
>
> When EBX® redirects the user to the custom login page, it also adds a `resume` query parameter to the URL. This `resume` query parameter contains an URL pointing to `/ebx-authentication/login` with some additional parameters and should be used in priority to redirect the user to `/ebx-authentication/login` after the authentication succeeds.

See also *Security configuration [p 593]*

106.4 Single Sign-On (SSO)

Single sign-on allows users to authenticate once and gain access to multiple applications and services through a centralized Identity Provider (IdP) rather than managing separate login credentials. This

integration streamlines the login process, reduces the risk of password-related security breaches, and enhances user experience by enabling seamless access.



See also *Synchronized directory* *[p 706]*

EBX® supports the SAML 2.0 and OpenID Connect protocols. To enable these features, you must declare and configure EBX® as an application in the Identity Provider. Configuration requirements depend on the enabled protocol. When using SAML 2.0, an *sp-descriptor.xml* is generated on startup and should be imported in the Identity Provider to ensure a seamless configuration.

In the case of a *reverse-proxy* configuration, the base URL of the server should be configured in the properties of both protocols.

> **Note**
>
> Only one protocol can be configured at a time.

> **Note**
>
> The logout process initiated from the IdP will only apply to the browser used to log out.

> **Note**
>
> The default security setting prevents cookies from being sent during cross-site POST requests, which blocks SSO authentication. To avoid this issue, set the authorization cookie to `Lax`: `ebx.security.authorizationCookie.attribute.sameSite=Lax`.

The endpoints used to initiate a login/logout via the configured protocol are:

```
http[s]://<host>[:<port>]/ebx-authentication/rest/sso/v1/login
```

> **Note**
>
> A *resume* query parameter can be used to hold an *encoded URL* to which the client should be redirected once authentication is completed.

```
http[s]://<host>[:<port>]/ebx-authentication/rest/sso/v1/logout
```

If there is a need to access or redirect to the built-in or custom login page that uses the internal directory, the following url may be used:

```
http[s]://<host>[:<port>]/ebx-authentication/login?method=loginPage
```

SAML 2.0

To enable SAML 2.0, configure the service provider properties [p 579] or EBX® Container Edition environment variables [p 644] and have a keystore, idp-descriptor.xml and sp-descriptor.xml in the repository folder.

- **Keystore:** A keystore must be configured for all signature and encryption operations. Use the following java keytool command to generate a new one:

  ```
  keytool -genkeypair -alias ebx.saml -keypass keypass -keystore keystore.p12 -
  storepass keypass -keyalg RSA -keysize 2048 -validity 3650
  ```

- **idp-descriptor.xml:** The *IDPSSODescriptor* XML file exported from the Identity Provider.
- **sp-descriptor.xml:** The *SPSSODescriptor XML* file generated by EBX® on startup using the configured properties.

The configured `nameId` is extracted from the assertion and matched with the EBX® directory to authenticate the user. Therefore, it's crucial to precisely configure the `nameId` attribute or the format.

The *urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST* binding type is used for all operations.

The same endpoint is used for both assertion consumer and single logout services:

```
http[s]://<host>[:<port>]/ebx-authentication/rest/sso/v1/saml2/callback
```

> **Note**
>
> A property change triggers the generation of a new *sp-descriptor.xml* and the loading of the existing *keystore.jks* and *idp-descriptor.xml*.

OpenID Connect

To enable OpenID connect, configure the client properties [p 579] or for EBX® Container Edition the environment variables [p 645]. When using OpenID:

- The client authentication method used is *Client Secret Post*.
- The response mode is *form_post* and the response type is *code*.

The same endpoint is used for both redirect and logout operations:

```
http[s]://<host>[:<port>]/ebx-authentication/rest/sso/v1/oidc/callback
```

CHAPTER 107

Performance and tuning

This chapter contains the following topics:

1. Environment
2. Database
3. Data modeling
4. Data validation
5. Accessing tables
6. Performance checklist for other Java customizations

107.1 Environment

Memory management

Memory monitoring	Indications of EBX® load activity are provided by monitoring the underlying database, and also by the 'monitoring' logging category [p 582]. If the numbers for *cleared* and *built* objects remain high for a long time, this is an indication that EBX® is swapping on the application server. In that case, the memory allocated to the application server should be increased.
Garbage collector	Tuning the garbage collector can also benefit overall performance. This tuning should be adapted to the use case and specific Java Runtime Environment used.

CPU

The number of CPUs available for the application server must be defined considering the number of concurrent HTTP requests to be served, the complexity (CPU cost) of the implied tasks, and the background activities, including the Java garbage collection.

Large imports and more generally large transactions involving many creates, updates and deletes will be completed faster if the difference between the server load and the number of available processors

allows the indexing to be efficiently run in parallel within the transaction. The `persistence` log category contains the following entries:

- *'Setting forced synchronous indexing to false (...)'* to indicate that the indexing will be performed concurrently;
- *'Setting forced synchronous indexing to true (...)'* to indicate that the indexing will not be performed concurrently.

The difference mentioned above is assessed every ten seconds and is computed using the methods `getSystemLoadAverage()` and `getAvailableProcessors()` in the Java class `java.lang.management.OperatingSystemMXBean`. Both numbers are written at the end of the log entry above.

Scanning on server startup

To speed up the web applications server startup, the JAR files scanner [p 542] should be configured.

107.2 Database

Reorganizing database tables

As with any database, inserting and deleting large volumes of data may lead to fragmented data, which can deteriorate performance over time. To resolve the issue, reorganizing the impacted database tables is necessary. See Monitoring and cleanup of the relational database [p 666].

A specificity of EBX® is that creating dataspaces and snapshots adds new entries to tables `GRS_DTR` and `GRS_SHR`. When poor performance is experienced, it may be necessary to schedule a reorganization of these tables, for large repositories in which many dataspaces are created and deleted.

See also *Monitoring and cleanup of the relational database [p 666]*

107.3 Data modeling

Aggregated lists

In a data model, when an element's cardinality constraint `maxOccurs` is greater than 1 and no `osd:table` is declared on this element, it is implemented as a Java `List`. This type of element is called an aggregated list [p 910], as opposed to a table.

It is important to consider that there is no specific optimization when accessing aggregated lists, in terms of iterations, user interface display, etc. Besides performance concerns, aggregated lists are limited with regard to many functionalities that are supported by tables. See tables introduction [p 913] for a list of these features.

Attention

For the reasons stated above, aggregated lists should be used only for small volumes of simple data (one or two dozen records), with no advanced requirements for their identification, lookups, permissions, etc. For larger volumes of data (or more advanced functionalities), it is recommended to use `osd:table` declarations.

Inherited fields

It is possible in a data model to use inherited fields [p 480] for an advanced inheritance based on relationships opposed to dataset inheritance.

As inherited fields cannot benefit from index optimizations they should be used with caution or avoided since they behave as computed fields to resolve their values. As a consequence all operations (querying, validation, data comparison in resolved mode, etc.) that will be performed on inherited fields won't be optimized and their performances can be strongly impacted.

107.4 Data validation

The internal validation framework will optimize the work required during successive requests to update the validation report of a dataset or a table. The incremental validation process behaves as follows:

- The first call to a dataset or table validation report performs a full validation of the dataset or the table.
- The next call to the validation report will compute the changes performed since the last validation. The validation report will be updated according to these changes.
- Validation reports are stored persistently in the TIBCO EBX® repository. This reduces the amount of memory dedicated to validation reports when datasets have a large amount of validation messages. Also, validation reports are not lost when the application server restarts.
- Validation reports can be reset manually in the user interface by an administrator user (this option is available from the validation report section in EBX®). As a consequence, resetting validation reports must be used with caution since associated datasets or tables will be fully revalidated during the next call to their validation reports.

  See `Adaptation.resetValidationReport`[API] for more information.

Certain constraints are systematically re-validated, even if no updates have occurred since the last validation. These are the constraints with *unknown dependencies*. An element has unknown dependencies if:

- It specifies a **programmatic constraint** `Constraint`[API] in the default *unknown dependencies* mode.
- It declares a **computed value** `ValueFunction`[API], or it declares a dynamic facet that depends on an element that is itself a **computed value** `ValueFunction`[API].
- It is an inherited field [p 482] or it declares a dynamic facet that depends on a node that is itself an inherited field [p 482].

> **Note**
>
> It is possible to disable the validation of inherited fields to avoid performance issues when validating large tables. See inherited field [p 482] for more information.

Consequently, on large tables, it is recommended to:

- Avoid constraints with unknown dependencies (or at least to minimize the number of such constraints). For programmatic constraints, the developer is able to specify two alternative modes that drastically reduce incremental validation cost: *local dependency* mode and *explicit dependencies*. For more information, see **Dependencies and validation** `DependenciesDefinitionContext.dependencies`[API].

- To use **constraints on tables** `ConstraintOnTable`[API] instead of **programmatic constraints** `Constraint`[API] defined at field level. Indeed, if a table defines constraints at field level, then the validation process will iterate over all the records to check if the value of the associated field complies with the constraint. Using **constraints on tables** `ConstraintOnTable`[API] gives the opportunity to execute optimized queries on the whole table. See below [p 615] for recommendations on how to optimize such queries. However, there is a trade-off: A **constraint on tables** `ConstraintOnTable`[API] does not benefit from the incremental validation and will always be validated completely. If in general only a few records are created or deleted between validations, a **programmatic constraint** `Constraint`[API] might be more performant. Benchmarks or a sampling profiler can help to make the decision.

The following properties can be used to minimize the impact on performance when logging a validation report:

- Set the `ebx.validation.report.logContent` property to `false` to avoid logging the individual validation messages. The validation report summary is still logged, including the message count. You can also avoid logging the validations started by the Java API completely. See `ValidationSpec.setResultLogged`[API] for more information.
- Though less effective than the previous recommendation, setting the `ebx.validation.report.maxItemDisplayed` property to a lower value than the default 100 reduces the amount of validation messages to log, thereby reducing the related computational work.

107.5 Accessing tables

Functionalities

Tables are commonly accessed through EBX® UI, data services and also through the `Request`[API] and `Query`[API] APIs. This access involves a unique set of functions, including a *dynamic resolution* process. This process behaves as follows:

- **Inheritance**: Inheritance in the dataset tree takes into account records and values that are defined in the parent dataset, using a recursive process. Also, in a root dataset, a record can inherit some of its values from the data model default values, defined by the `xs:default` attribute.
- **Value computation**: A node declared as an `osd:function` is always computed on the fly when the value is accessed. See `ValueFunction.getValue`[API].
- **Filtering**: An XPath predicate [p 441], a **programmatic filter** `AdaptationFilter`[API], or a record-level **permission rule** `SchemaExtensionsContext.setAccessRuleOnOccurrence`[API] requires a selection of records.
- **Sort**: A sort of the resulting records can be performed.

Query on tables

Architecture and design

The EBX® engine manages persistent Lucene indexes to improve operation speeds on tables. This is done by:

- **Ensuring the index rebuild chain is optimized:** Think of the index rebuild process as a chain. The chain is made up of a sequence of table revisions, or links, that comprise the table's history. Each of these links is the result of a snapshot or child dataspace creation. Where possible, an

index rebuild skips links in the rebuild chain, reducing it to a maximum number of 5 links. This improves the rebuild duration and index structure through effective reuse of base indexes. If some links are skipped in the rebuild chain, the following message is logged in the `persistence.log` file: `"Index rebuild chain size is" + <fullSize> + ", reducing to 5.`

Please note that this process reduces rebuild times for intermediate snapshots; however, accessing skipped snapshots might take slightly longer, as their indexes must be rebuild incrementally from the nearest available point.

- **Dynamically assessing incremental rebuild viability:** EBX® estimates the "distance" between table revisions to determine whether to perform an incremental, or a full index rebuild. A full rebuild is initiated when the difference between the base and target index is too large. This leads to faster rebuilds and more efficient indexing. Accessing a snapshot older than this breaking point can take longer, as a new rebuild chain must be computed from scratch. If the rebuild chain requires a full rebuild, the following message is logged in the `persistence.log` file: `"Breaking revision chain at table revision " + aRevision.getId() + ": too different from the rebuild target."`

  > **Note**
  >
  > The above message is not logged when the first revision in the chain is rebuilt from scratch, or if table revisions are too minor to make the complexity of an incremental rebuild worth it.

- **Batching the processing for index rebuilds:** Index rebuild operations group deletion and creation actions to optimize efficiency.
- **Multi-threading rebuilds:** Multi-threading enhances rebuild performance through parallelism. You can control the number of threads used during index rebuilds. See Configuring multi-thread index builds [p 620] for more details.

> **Attention**
>
> Faster access to tables is ensured if indexes are ready and maintained in the OS memory cache. As mentioned above [p 615], it is important for the OS to have enough space allocated.

Performance considerations

The query optimizer favors the use of indexes when computing a request result. If a query cannot take advantage of the indexes, it will be resolved in Java memory, and experience poor performance on large volumes. The following guidelines apply:

> **Attention**
>
> - Only XPath predicates and SQL queries can benefit from index optimization.
> - Some fields and some datasets cannot be indexed, as described in section Limitations [p 512].
> - XPath predicates using the `osd:label` function cannot benefit from index optimization

If indexes have not yet been built, additional time is required to build and persist the indexes, on the first access to the table.

Accessing the table data blocks is required when the query cannot be computed against any index (whether for resolving a rule, filter or sort), as well as for building the index. If the table blocks are not present in memory, additional time is needed to fetch them from the database.

Apart from that, there are other considerations that can impact requests / queries performance:

- If the same request / query is going to be executed multiple times with just a variation on the filter value, it is recommendable to use parameters in said filter:
  - `request.setXPathFilter("./field=$param");`
  - `SELECT * FROM myTable WHERE field=?`
- If possible, use `RequestResult.isSizeGreaterOrEqual`[API] rather than `RequestResult.getSize`[API]
- If possible, avoid `RequestResult` size-related calls. Especially avoid the unnecessary code pattern "check isEmpty before traversing" on a `RequestResult`:

```
try (RequestResult result = ...) {
 if (result.isEmpty())
  return; // unnecessary and inefficient!
 for (Adaptation record : result) {
  ...
 }
}

// simply do:

try (RequestResult result = ...) {
 for (Adaptation record : result) {
  ...
 }
}
```

It is possible to get information through the memory monitoring [p 615] and request logging categories.

Accessing and modifying a table

The following access lead to poor performance, and must be avoided:

- Access a table after a few modifications, repeatedly. It implies the index state to be refreshed after each modification. The cost of refreshing makes this pattern ineffective. Instead, perform a single query and apply the modification when browsing the results.
- If there is an ongoing access to the same table, concurrently to the previous case, it prevents outdated index files to be deleted. As a consequence, the size of the index on disk increases, and the server may run out of disk space in extreme cases. When the concurrent access is closed, the index size is back to normal. This is usually a sign that a Request or a Query is not properly closed.

See also

`RequestResult.close`[API]

`QueryResult.close`[API]

Other operations on tables

The new records creations or record insertions depend on the primary key index. Thus, a creation becomes almost immediate if this index is already loaded.

Configuring multi-thread index builds

Use the `ebx.index.threads` property in the `ebx.properties` file to adjust the number of threads used during index rebuilds. This property defaults to a value of '2', which should be optimal for most

production scenarios. Since increasing the thread count can be detrimental to concurrent activity on a server, you might want to restrict this practice to a test environment. However, if a product upgrade, migration, or significant changes to data models require extensive index rebuilds, consider the following during the deployment phase:

- If you have a high-performance hardware configuration that allows a high concurrency level, you can target setting `ebx.index.threads` to a value of `CPU count/2` up to a maximum of 10.
- Additional threads are only started for large enough tables. The exact value is not fixed, but is generally larger than 200,000 records.

REST built-in and business objects

When using select operations with business objects, EBX® utilizes a temporary folder to handle large response contents. Administrators can set a size limit through the `ebx.dataservices.rest.bo.maxResponseSizeInKB` configuration parameter. Adjust this setting based on the server's available filesystem space dedicated to temporary content and considering the maximum number of request processing threads.

See ebx.dataservices.rest.bo.maxResponseSizeInKB [p 587] and Setting temporary files directories [p 581] for more information.

REST access to history table

The merge information in history table (the `merge_info` field) has a potentially high access cost. To improve performance and if the client code does not need this field, the `includeMergeInfo` [p 1142] parameter must be set to `false`.

See History [p 461] for more information.

Setting a fetch size

In order to improve performance, a fetch size should be set according to the expected size of the result of the request on a table. If no fetch size is set, the default value will be used.

- On a history table, the default value is assigned by the JDBC driver: 10 for Oracle and 0 for PostgreSQL.

> **Attention**
>
> On PostgreSQL, the default value of 0 instructs the JDBC driver to fetch the whole result set at once, which could lead to an `OutOfMemoryError` when retrieving large amounts of data. On the other hand, using fetchSize on PostgreSQL will invalidate server-side cursors at the end of the transaction. If, in the same thread, you first fetch a result set with a fetchsize, then execute a procedure that commits the transaction, then, accessing the next result will raise an exception.

See also

`Request.setFetchSize`[API]

`RequestResult`[API]

107.6 Performance checklist for other Java customizations

While TIBCO EBX® is designed to support large volumes of data, several common factors can lead to poor performance. Addressing the key points discussed in this section will solve the usual performance bottlenecks.

Expensive programmatic extensions

For reference, the table below details the programmatic extensions that can be implemented.

Use case	Programmatic extensions that can be involved
Validation	• **programmatic constraints** `Constraint`[API] • **computed values** `ValueFunction`[API]
Table access	• **record-level permission rules** `SchemaExtensionsContext.setAccessRuleOnOccurrence`[API] • **programmatic filters** `AdaptationFilter`[API]
EBX® content display	• **computed values** `ValueFunction`[API] • **UI Components** `UIBeanEditor`[API] • **node-level permission rules** `SchemaExtensionsContext.setAccessRuleOnNode`[API]
Data update	• **triggers** `Package com.orchestranetworks.schema.trigger`[API]

For large volumes of data, using algorithms of high computational complexity has a serious impact on performance. For example, the complexity of a constraint's algorithm is $O(n^2)$. If the data size is 100, the resulting cost is proportional to 10 000 (this generally produces an immediate result). However, if the data size is 10 000, the resulting cost will be proportional to 10 000 000.

Another reason for slow performance is calling external resources. Local caching usually solves this type of problem.

If one of the use cases above displays poor performance, it is recommended to track the problem, either by code analysis or by using a Java profiling tool.

Unnecessary index refresh

Refreshing a Lucene index takes time. It should be avoided whenever possible.

When does a refresh happen?	In the context of a transaction, an index refresh occurs when the table has been modified and one of the conditions below occurs: 1. For a lookup by primary key, the refresh is always triggered if the searched key has been "touched" (created, modified or deleted) in the current `Procedure` (or `TableTrigger`). 2. For a standard `Query` (or `Request`), an index refresh is always performed if the table has been modified in the current `Procedure` (or `TableTrigger`).
Coding recommendations	1. To avoid triggering a refresh through a lookup by primary key, the developer must register the `Adaptation` object returned from the last call to `doCreateOccurrence` or `doModifyContent`, and reuse this object instead of performing the lookup. 2. Avoid any lookup by primary key on a record that has been deleted in the current procedure. 3. In the case of a query triggering the refresh, the developer must ask the following question: can this query be avoided in my procedure?

Transaction threshold for mass updates

It is generally not advised to use a single transaction when the number of atomic updates in the transaction is beyond the order of 10^5. Large transactions require a lot of resources, in particular, memory, from EBX® and from the underlying database.

To reduce the transaction size, it is possible to:

- Specify the property ebx.manager.import.commit.threshold [p 594]. However, this property is only used for interactive archive imports performed from the EBX® user interface.
- Explicitly specify a **commit threshold** `ProcedureContext.setCommitThreshold`[API] inside the batch procedure.
- Structurally limit the transaction scope by implementing `Procedure`[API] for a part of the task and executing it as many times as necessary.

On the other hand, specifying a very small transaction size can also hinder performance, due to the persistent tasks that need to be done for each commit.

> **Note**
>
> If intermediate commits are a problem because transactional atomicity is no longer guaranteed, it is recommended to execute the mass update inside a dedicated dataspace. This dataspace will be created just before the mass update. If the update does not complete successfully, the dataspace must be closed, and the update reattempted after correcting the reason for the initial failure. If it succeeds, the dataspace can be safely merged into the original dataspace.

Triggers

If required, triggers can be deactivated using the method `ProcedureContext.setTriggerActivation`[API].

Directory integration

Authentication and permissions management involve the user and roles directory [p 703].

If a specific directory implementation is deployed and accesses an external directory, it can be useful to ensure that local caching is performed. In particular, one of the most frequently called methods is `Directory.isUserInRole`[API].

CHAPTER 108

Configuration notes

This chapter contains the following topics:

1. Memory allocated to the app server
2. Memory allocated to the OS

108.1 Memory allocated to the app server

Since the query engine retrieves the necessary information from persistent storage, the memory allocated to the Java Virtual Machine (usually specified by the -Xmx parameter) can be kept low. We recommend setting this figure to less than 1/3rd of the total memory available on the OS. We also advise to stay below 32 GB, which should fit all reasonable use cases, and allow benefiting from the Compressed Ordinary Object Pointer feature.

108.2 Memory allocated to the OS

On the OS running the application server, it is important to leave sufficient room to the OS cache, letting it optimize access to the persistent Lucene indexes. Indeed, once these have been loaded from the file system, the OS uses its memory cache to speed up subsequent accesses to this same data, and avoid reloading it every time from the disk. This is only possible if sufficient RAM has been left for this purpose.

It is also necessary to configure the OS so that the JVM process can reserve the resources required by numerous memory-mapped files (see this article for details). On a Linux OS, this can be done by issuing the following commands:

```
ulimit -n 512000
sysctl vm.max_map_count=262144
```

EBX® Container Edition

CHAPTER 109

Building the image

This chapter contains the following topics:

1. Overview
2. Requirement
3. Building the image

109.1 Overview

TIBCO EBX® Container Edition image is Linux OS based for several architectures and includes the following:

- Application Server Apache Tomcat® 10.1
- Java JDK 25 LTS

109.2 Requirement

The image can be built on a POSIX native system or using a compatible layer:

- **Operating system** : Linux, macOS, Windows (10 version 2004, Server 2022 or greater) using WSL2,
- **Processor type** : AMD64 (Intel 64 bits), ARM64,
- **Docker Engine** installed and running,
- Access to **Internet** .

109.3 Building the image

Download installers

To build an EBX® image, one needs to download file **TIB_ebx_6.2.3_addon_6.2.X_container_edition.zip** from TIBCO eDelivery.

Running the interactive installer

To start installer on **Linux** or **macOS** :

- Unzip the **TIB_ebx_6.2.3_addon_6.2.X_container_edition.zip** ,

- Open a terminal in folder where file **ebx-ce-installer.sh** file is located,
- Execute command **./ebx-ce-installer.sh** .

To start installer on **Windows** :

- Unzip the **TIB_ebx_6.2.3_addon_6.2.X_container_edition.zip** ,
- Open a Windows PowerShell in folder where file **ebx-ce-installer.bat** file is located,
- Execute command **ebx-ce-installer.bat** .

Follow the instructions and select the optional components to be added to the image. If Metadata is selected, the interactive installer automatically adds the following addons: DAMA, DINT, DMDV, DPRA and TESE.

The installer will then build the image and print a summary similar to:

```
********************************************************************************************
The TIBCO EBX Container Edition image was successfully created with following names:
 ebx:latest
 ebx:6.2.3-dama-dint-dmdv-dpra-mame-metadata-tese-6.2.X
This image includes the following optional component(s):
 Metadata
 Digital Asset Manager (DAMA)
 Data Exchange (New) (DINT)
 Data Model and Data Visualization (DMDV)
 Insight (New) (DPRA)
 Match and Merge (MAME)
 EBX GO (MODA)
 Information Search (TESE)
 Data Exchange (ADIX, legacy)
 Insight (DQID, legacy)
To run this image with the default configuration and an embedded database, use command:
 docker run -p 8080:8080 -d ebx:latest
You can test EBX by visiting http://localhost:8080 in a browser.
To run this image with other configurations, for example with an external database, see
documentation.
********************************************************************************************
```

If you select the **Generate image only for Linux on current architecture?** option, the install prompts for additional architectures that you want to build EBX® Container Edition for. In that case, Docker image tags will contain the architecture it was built for. For instance, **ebx:6.2.3-dama-dint-dmdv-dpra-mame-metadata-tese-6.2.X-amd64** .

Running the batch installer

To start batch installer on **Linux** or **macOS** :

- Unzip the **TIB_ebx_6.2.3_addon_6.2.X_container_edition.zip** .
- Open a terminal in the folder where the **ebx-ce-installer-batch.sh** file is located.
- Execute the following command: **./ebx-ce-installer-batch.sh [options] [metadata] [<addon(s)>]**

```
Batch script to build image with optionally Metadata and/or addons.

Options:
  -a             add Metadata and all addons excluding legacy addons
  -al            add Metadata and all addons including legacy addons
  -p <arch,...>  architecture(s) to build on (i.e. \"amd64,arm64/v8,ppc64le\")
                 possible values: amd64, arm64/v8, ppc64le, s390x
                 using this parameter the default image tag is suffixed
  -t <tag_file>  create a file named tag_file including the image tag built
  --help         display this help and exit

  [metadata]     install Metadata
  [<addon(s)>]   install one or more addons (in lowercase and separated by a white space)
                 for example: mame tese
```

To start the batch installer on **Windows** :

- Unzip the **TIB_ebx_6.2.3_addon_6.2.X_container_edition.zip** .
- Open a Windows PowerShell in the folder where the **ebx-ce-installer-batch.bat** file is located.
- Execute the following command: **ebx-ce-installer-batch.bat [options] [metadata] [<addon(s)>]**

The batch installer will then build the image and print a summary identical to the interactive installer.

Testing the image

The image can be run locally using command:

```
docker run -p 8080:8080 -d ebx:latest
```

In **production** , it is recommended to **not** use the **latest** tag. The installer always generates another tag, depending on the selected optional components.

Using the previous example, the tag is **6.2.3-dama-dint-dmdv-dpra-mame-metadata-tese-6.2.X** because optional components where selected:

- Metadata,
- Digital Asset Manager (DAMA),
- Data Exchange (New) (DINT),
- Data Model and Data Visualization (DMDV),
- Insight (New) (DPRA),
- Match and Merge (MAME),
- Information Search (TESE).

The image can then be run using the following command:

```
docker run -p 8080:8080 -d ebx:6.2.3-dama-dint-dmdv-dpra-mame-metadata-tese-6.2.X
```

If no optional components are selected, the tag is **6.2.3** .

The image can be run using the following command:

```
docker run -p 8080:8080 -d ebx:6.2.3
```

Sharing the image

The steps to share an image depend on your company's infrastructure.

In the following example, the image is pushed to a Docker private registry named **myregistry** :

```
docker tag ebx:6.2.3 myregistry:5000/ebx:6.2.3
docker push myregistry:5000/ebx:6.2.3
```

In the event that the image is built for several architectures, detailed instructions are provided in the **README-MULTIARCH.md** file located in the **files/** subdirectory.

Other architectures than AMD64

The installers are tested on AMD64 (Intel 64 bits). On these architectures the generated image's architecture will be **linux/amd64** .

The installers may succeed when running on a system using another architecture. In that case the generated image platform will be linux for that architecture type.

For example, if the installer runs on a macOS M1 workstation, the generated image's platform will be **linux/arm64** .

Attention

Generating an Linux image for AMD64 (Intel 64 bits) or ARM64 architecture is fully supported. The other platforms are **experimental** and are **not supported in production** .

CHAPTER 110

Running the image

This chapter contains the following topics:

1. Support Policy
2. Starting EBX®
3. Container access
4. Environment variables
5. Configuration files
6. Volumes
7. Linux user and group
8. Host configuration
9. Logs access

110.1 Support Policy

Docker

TIBCO EBX® Container Edition is tested with the Docker Engine version 20.10.

Kubernetes

Current EBX® Container Edition release was tested with a certified Kubernetes implementation version 1.26 and Red Hat® OpenShift® 4.13.1.

We provides Helm chart samples for EBX® Container Edition that are available on GitHub.

See https://github.com/TIBCOSoftware/ebx-container-edition for more information.

110.2 Starting EBX®

First-launch assistant

To start EBX® with default configuration that includes an embedded H2 database, execute command:

```
docker run -p 8080:8080 -d ebx:6.2.3
```

Using a browser, you can connect to EBX® with URL http://localhost:8080 . This will display the **first-launch assistant** that will help you configure EBX®.

For more information on the first launch assistant, see chapter Initialization and first-launch assistant [p 603] .

Automatic initialization

To start EBX® with automatic initialisation on first startup and an embedded H2 database, execute command:

```
docker run -d -p 8080:8080 \
 -e "EBX_FLA_DISABLED=true" \
 -e "EBX_INSTALL_ADMIN_PASSWORD=<password>" \
 -e "EBX_INSTALL_LICENSE=XXXXX-XXXXX-XXXXX-XXXXX" \
 ebx:6.2.3
```

The EBX® repository will be automatically created on first startup.

Using a browser, you can connect to EBX® with URL http://localhost:8080 . This will display the EBX® login screen. The username for administrator is **admin** and the password is the one specified in previous command.

Note

It’s possible to specify another username for the administrator. For more information see Automatic repository installation on first launch [p 635] .

Supported browsers

For details a supported browsers see: Supported Web Browsers [p 524] .

110.3 Container access

The following command will start a bash shell inside the EBX® container using default user:

```
docker exec -it <container-id> bash
```

To connect as root, use command:

```
docker exec -it --user root <container-id> bash
```

110.4 Environment variables

This chapter describes the environment variables supported by **EBX® Container Edition** .

All are optional.

Disabling First-launch assistant

For security reasons, one might want to disable the first-launch assistant in all circumstances.

This is achieved by setting environment variable to **EBX_FLA_DISABLED** to **true** .

Automatic repository installation on first launch

If the repository is not yet initialized and first-launch assistant is disabled, EBX® will automatically trigger its installation if following mandatory variables are provided:

Name	Default	Description
EBX_INSTALL_ADMIN_LOGIN	admin	Sets the EBX® administrator login name. This parameter is ignored if repository variable **EBX_FLA_DISABLED** value is not **true** or if repository is already initialized.
EBX_INSTALL_ADMIN_PASSWORD		Sets the EBX® administrator password. This parameter is mandatory if **EBX_FLA_DISABLED** value is **true** and is ignored if repository variable **EBX_FLA_DISABLED** value is not **true** or if repository is already initialized.
EBX_INSTALL_LICENSE		Sets the EBX® license key if not already defined.

Note

If mandatory variables are not provided, EBX® will display an error message.

Example

To automatically install repository launch EBX® using following command:

```
docker run -d -p 8080:8080 \
 -e "EBX_FLA_DISABLED=true" \
 -e "EBX_INSTALL_ADMIN_LOGIN=<login-name>" \
 -e "EBX_INSTALL_ADMIN_PASSWORD=<password>" \
 -e "EBX_INSTALL_LICENSE=XXXXX-XXXXX-XXXXX-XXXXX" \
 ebx:6.2.3
```

Define EBX® license key

To start EBX® with a dedicated license key, execute command:

```
docker run -d -p 8080:8080 \
 -e "EBX_INSTALL_LICENSE=XXXXX-XXXXX-XXXXX-XXXXX" \
 ebx:6.2.3
```

Note: **EBX_INSTALL_LICENSE** is ignored if license key is already defined at persisted location.

The license key is persisted at location **/ebx/data/ebx-license.properties** .

See Volumes [p 650] for more information.

EBX® run mode

This property defines how EBX® runs.

Name	Default	Description
EBX_BACKEND_MODE	development	Three run modes are available: *development* , *integration* and *production* .

```
docker run -d -p 8080:8080 \
```

```
 -e "EBX_BACKEND_MODE=production" \
 ebx:6.2.3
```

See EBX® run mode [p 599] for more information.

URL configuration

Some EBX® features require generating URLs. Specific configuration may be required to achieve this, for example if EBX® is running behind a reverse proxy or on a Kubernetes cluster.

Name	Default	Description
EBX_IS_SECURED	If incoming request is HTTPS, "true" is assumed or else "false" is assumed.	If "true", the protocol "HTTPS" is always assumed. This value can be useful if the container is behind a reverse proxy, a firewall or an ingress that takes care of HTTPS encryption. This will also set the EBX® authentication cookie to be **secure** . This means that a user using a modern browser will not be able to log in using HTTP. If "false", the protocol "HTTP" is assumed.
EBX_AUTHENTICATION_REDIRECT_TO_HTTPS	Default is "false".	If "true" and the incoming request to the login form is HTTP, a redirect occurs to to force HTTPS. This property it should be set to false if the container is behind a reverse proxy, a firewall or an ingress that takes care of HTTPS encryption.
EBX_HOSTNAME	The host name specified by the incoming HTTP(S) request.	The EBX® server host name.
EBX_PORT	The port number specified by the incoming HTTP request.	The EBX® server HTTP port number.
EBX_PORT_SECURED	The port number specified by the incoming HTTPS request.	The EBX® server HTTPS port number. **Note** HTTPS support must be provided by a reverse proxy or an ingress that takes care of encryption.
EBX_ROOT_PATH	By default, the context path is empty.	If set, all EBX® urls will be prefixed by this value. The value must have a leading **/** and must not have a trailing **/** except if value is **/** . For example a valid value is **/mdm/sales** . Setting this variable is useful when running more than one instance of EBX® with the same host name.

Name	Default	Description
EBX_URL_DEFAULT		This environment variable is used when a background task needs to calculate a URL to EBX®. It should be set to a full URL without the path component (EBX_ROOT_PATH applies for the path component). For example a valid value is: https://host_name . If EBX_URL_DEFAULT is not specified and EBX_HOSTNAME is specified, a default is calculated with following assumptions: - If EBX_IS_SECURED is **true** , then HTTPS is assumed with port number equal to EBX_PORT_SECURED (443 is the default). - If EBX_IS_SECURED is **false** or not set, then HTTP is assumed with port number equal to EBX_PORT (80 is the default).

EBX® Database connectivity

For information on supported databases see chapter Supported databases [p 527] .

By default, an embedded H2 database is used. Data for this H2 database is persisted at location **/ebx/data/h2** .

An external database may be configured using the following variables:

Name	TIBCO EBX® main configuration file equivalent	Description
EBX_DB_FACTORY	ebx.persistence.factory	Specifies the type of database server.
EBX_DB_URL	ebx.persistence.url	The JDBC URL. Its format is: **jdbc:<dialect>://<database_host>:<database_port_number>/<database_name>** .
EBX_DB_USER	ebx.persistence.user	The database user id.
EBX_DB_PASSWORD	ebx.persistence.password	The database user password.

For more information on these variables see their TIBCO EBX® main configuration file equivalent in chapter Configuring the EBX® repository [p 574] .

Note

The container includes JDBC drivers only for H2, PostgreSQL and Microsoft SQL Server. Using other databases that are supported by EBX® requires adding the driver.

For instructions on how to add a driver, see Adding a new JDBC driver [p 655] for more information.

You can configure EBX® to use a JNDI data source. See Configuring the database in JNDI [p 656] for more information.

Example

To start an instance of EBX® that connects to a PostgreSQL database, execute the following command:

```
docker run -d -p 8080:8080 \
 -e "EBX_DB_FACTORY=postgresql" \
 -e "EBX_DB_URL=jdbc:postgresql://<database_host>:<database_port>/<database_name>" \
 -e "EBX_DB_USER=<user_name>" \
 -e "EBX_DB_PASSWORD=<user_password>" \
 ebx:6.2.3
```

Metadata classifier database connectivity

Metadata classifier uses the SQLAlchemy toolkit to connect to its database. The SQLAlchemy toolkit supports most common databases.

By default, an embedded SQLite database is used. Data for this SQLite database is persisted at this location: **/ebx/data/sqlite/ebx-metadata-classifier.db** .

An external database may be configured using the following variables:

Name	Description
EBX_METADATA_CLASSIFIER_DB_URL	The SQLAlchemy URL without credentials. Its format is: **<dialect>://<database_host>:<database_port>/<database_name>** .
EBX_METADATA_CLASSIFIER_DB_USER	The database user id.
EBX_METADATA_CLASSIFIER_DB_PASSWORD	The database user password.

The container includes only the driver for the default database (SQLite). Using other databases that are supported by SQLAlchemy requires adding the driver.

For instructions on how to add a driver, see Adding a driver for the Metadata classifier database [p 657] .

Example

To start an instance of Metadata that connects to a PostgreSQL database, execute the following command:

```
docker run -d -p 8080:8080 \
 -e "EBX_METADATA_CLASSIFIER_DB_URL=postgresql://<database_host>:<database_port>/<database_name>" \
 -e "EBX_METADATA_CLASSIFIER_DB_USER=<user_name>" \
 -e "EBX_METADATA_CLASSIFIER_DB_PASSWORD=<user_password>" \
 ebx:6.2.3
```

Notes

- The Metadata classifier database name (**database_name**) can be the same as Database connectivity [p 638] .
- For information about how to use a specific database driver and how to escape special characters in the connection URL, please refer to the SQLAlchemy's documentation.
- The H2 database is not supported by SQLAlchemy.

Directory connectivity

The EBX® directory can be configured through the following environment variables:

Name	TIBCO EBX® main configuration file equivalent	Description
EBX_DIRECTORY_FACTORY	ebx.directory.factory	Specifies the Java directory factory class name (optional)
EBX_DIRECTORY_SYNCHRONIZED_SOURCE	ebx.directory.synchronized.source	Enables and specifies the synchronized directory implementation (LDAP, SCIM or OnLogin)
EBX_SSO_PROTOCOL	ebx.directory.sso.protocol	Enables and specifies the SSO protocol (OIDC or SAML2)

See Configuring the user and roles directory [p 575] for more information.

Synchronized directory

The EBX® synchronized directory can be configured through the following environment variables:

Name	TIBCO EBX® main configuration file equivalent (spaces must be removed)
EBX_DIRECTORY_SYNCHRONIZED_ROLE_MAPPER	ebx.directory.synchronized.role.mapper
EBX_DIRECTORY_SYNCHRONIZED_ROLE_MAPPER_DEFAULT_ADMINISTRATOR	ebx.directory.synchronized.role.mapper.default.administrator
EBX_DIRECTORY_SYNCHRONIZED_ROLE_MAPPER_DEFAULT_READ_ONLY	ebx.directory.synchronized.role.mapper.default.read_oly
EBX_DIRECTORY_SYNCHRONIZED_MAPPING_LDAP_ATTRIBUTES	ebx.directory.synchronized.mapping.ldap.attributes
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAPPING_SAML2_ATTRIBUTES_ROLE_NAME	ebx.directory.synchronized.sso.mapping.saml2.attributes.role.name
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAPPING_SAML2_ATTRIBUTES	ebx.directory.synchronized.sso.mapping.saml2.attributes
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAPPING_OIDC_CLAIMS_ROLE_NAME	ebx.directory.synchronized.sso.mapping.oidc.claims.role.name
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAPPING_OIDC_CLAIMS	ebx.directory.synchronized.sso.mapping.oidc. claims
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAPPING_BEARER_CLAIMS_ROLE_NAME	ebx.directory.synchronized.sso.mapping.bearer.claims.role.name
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAPPING_BEARER_CLAIMS	ebx.directory.synchronized.sso.mapping.bearer. claims
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAPPING_ATTRIBUTE.MAIL	ebx.directory.synchronized.sso.mapping.attribute. mail
EBX_DIRECTORY_SYNCHRONIZED_SSO_DISPLAY_USER	ebx.directory.synchronized.sso.display.user
EBX_DIRECTORY_SYNCHRONIZED_SSO_LISTENERS	ebx.directory.synchronized.sso.listeners

LDAP connectivity

The EBX® integration to an existing LDAP directory can be configured through the following environment variables:

Name	TIBCO EBX® main configuration file equivalent (spaces must be removed)
EBX_LDAP_DEFAULT_HOSTNAME	ebx.directory.ldap.default.hostName
EBX_LDAP_DEFAULT_PORT	ebx.directory.ldap.default.port
EBX_LDAP_DEFAULT_CONNECTION_TIMEOUT	ebx.directory.ldap.default. connectionTimeOutInSeconds
EBX_LDAP_DEFAULT_BIND_DN_OR_USER	ebx.directory.ldap.default.bindDnOrUser
EBX_LDAP_DEFAULT_BIND_PASSWORD	ebx.directory.ldap.default.bindPassword
EBX_LDAP_DEFAULT_AUTHENTICATION_METHOD	ebx.directory.ldap.default. authenticationMethod
EBX_LDAP_DEFAULT_ENCRYPTION_METHOD	ebx.directory.ldap.default. encryptionMethod
EBX_LDAP_DEFAULT_MAPPING_ROLE_REFERENCE_ATTRIBUTE	ebx.directory.ldap.default.mapping. role.referenceAttribute
EBX_LDAP_DEFAULT_MAPPING_ROLE_MEMBER_ATTRIBUTE	ebx.directory.ldap.default.mapping. role.memberAttribute
EBX_LDAP_DEFAULT_MAPPING_USER_REFERENCE_ATTRIBUTE	ebx.directory.ldap.default.mapping. user.referenceAttribute
EBX_LDAP_DEFAULT_MAPPING_USER_REFERENCE_COMPONENT_DN	ebx.directory.ldap.default.mapping. user.referenceComponentInDN
EBX_LDAP_DEFAULT_MAPPING_ROLE_MAIL_ATTRIBUTE	ebx.directory.ldap.default.mapping. role.mailAttribute
EBX_LDAP_DEFAULT_MAPPING_USER_MAIL_ATTRIBUTE	ebx.directory.ldap.default.mapping. user.mailAttribute
EBX_LDAP_DEFAULT_DISPLAY_ROLE	ebx.directory.ldap.default.display.role
EBX_LDAP_DEFAULT_DISPLAY_USER	ebx.directory.ldap.default.display.user
EBX_LDAP_DEFAULT_DISPLAY_USER_ALL	ebx.directory.ldap.default.display.user.all

Each request from USER_SEARCH, USER_GROUPS_SEARCH, ALL_USERS_SEARCH, ALL_GROUPS_SEARCH, can be configured through the following environment variables:

For example, the USER_SEARCH request would be:

Name	TIBCO EBX® main configuration file equivalent (spaces must be removed)
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_BASEDN	ebx.directory.ldap.default.request.userSearch.baseDN
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_FILTER	ebx.directory.ldap.default.request.userSearch.filter
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_SCOPE	ebx.directory.ldap.default.request.userSearch.scope
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_CACHE_EXPIRATION	ebx.directory.ldap.default.request.userSearch.cache.expirationInSeconds
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_CACHE_REFRESH	ebx.directory.ldap.default.request.userSearch.cache.refreshInSeconds
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_CACHE_MAX_SIZE	ebx.directory.ldap.default.request.userSearch.cache.maxSize
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_PAGE_SIZE	ebx.directory.ldap.default.request.userSearch.pageSize
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_TIME_LIMIT	ebx.directory.ldap.default.request.userSearch.timeLimitInSeconds

See Built-in LDAP directory [p 577] for more information.

Example

To start an instance with LDAP connectivity, execute the following command:

```
docker run -d -p 8080:8080 \
 -e "EBX_DIRECTORY_FACTORY=com.orchestranetworks.service.directory.ldap.LdapDirectoryFactory" \
 -e "EBX_LDAP_DEFAULT_HOSTNAME=localhost" \
 -e "EBX_LDAP_DEFAULT_PORT=389" \
 -e "EBX_LDAP_DEFAULT_MAPPING_ROLE_REFERENCE_ATTRIBUTE=cn" \
 -e "EBX_LDAP_DEFAULT_MAPPING_ROLE_MEMBER_ATTRIBUTE=member" \
 -e "EBX_LDAP_DEFAULT_MAPPING_USER_REFERENCE_ATTRIBUTE=uid" \
 -e "EBX_LDAP_DEFAULT_DISPLAY_ROLE={{cn}}" \
 -e "EBX_LDAP_DEFAULT_DISPLAY_USER={{cn}} ({{mail}})" \
 -e "EBX_LDAP_DEFAULT_DISPLAY_USER_ALL=true
 -e "EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_BASEDN=dc=example,dc=org" \
 -e "EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_FILTER=(uid={{login}})" \
 -e "EBX_LDAP_DEFAULT_REQUEST_USER_GROUPS_SEARCH_BASEDN=dc=example,dc=org" \
 -e "EBX_LDAP_DEFAULT_REQUEST_USER_GROUPS_SEARCH_FILTER=(&(objectClass=groupOfNames)(member={{dn}}))" \
 -e "EBX_LDAP_DEFAULT_REQUEST_ALL_GROUPS_SEARCH_BASEDN=dc=example,dc=org" \
 -e "EBX_LDAP_DEFAULT_REQUEST_ALL_GROUPS_SEARCH_FILTER=(objectClass=groupOfNames)" \
 -e "EBX_LDAP_DEFAULT_REQUEST_ALL_USERS_SEARCH_BASEDN=dc=example,dc=org" \
 -e "EBX_LDAP_DEFAULT_REQUEST_ALL_USERS_SEARCH_FILTER=(objectClass=inetOrgPerson)" \
 ebx:6.2.3
```

Note

To enable LDAP directory, **EBX_DIRECTORY_FACTORY** must be defined to **com.orchestranetworks.service.directory.ldap.LdapDirectoryFactory** .

SSO SAML 2.0 connectivity

The EBX® Single Sign-On (SSO) SAML 2.0 built-in can be configured through the following environment variables:

Name	TIBCO EBX® main configuration file equivalent
EBX_SAML2_DEFAULT_SP_BASE_URL	ebx.directory.saml2.default.sp.base.url
EBX_SAML2_DEFAULT_SP_ENTITY_ID	ebx.directory.saml2.default.sp.entity.id
EBX_SAML2_DEFAULT_KEYSTORE_URI	ebx.directory.saml2.default.keystore.uri
EBX_SAML2_DEFAULT_KEYSTORE_TYPE	ebx.directory.saml2.default.keystore.type
EBX_SAML2_DEFAULT_KEYSTORE_PATH	ebx.directory.saml2.default.keystore.path
EBX_SAML2_DEFAULT_KEYSTORE_PASSWORD	ebx.directory.saml2.default.keystore.password
EBX_SAML2_DEFAULT_KEYSTORE_KEY_ALIAS	ebx.directory.saml2.default.keystore.key.alias
EBX_SAML2_DEFAULT_KEYSTORE_KEY_PASSWORD	ebx.directory.saml2.default.keystore.key.password
EBX_SAML2_DEFAULT_NAME_ID_ATTRIBUTE	ebx.directory.saml2.default.name.id.attribute
EBX_SAML2_DEFAULT_NAME_ID_FORMAT	ebx.directory.saml2.default.name.id.format
EBX_SAML2_DEFAULT_AUTHN_REQUESTS_SIGNED	ebx.directory.saml2.default.authn.requests.signed
EBX_SAML2_DEFAULT_WANT_ASSERTIONS_SIGNED	ebx.directory.saml2.default.want.assertions.signed
EBX_SAML2_DEFAULT_WANT_RESPONSES_SIGNED	ebx.directory.saml2.default.want.responses.signed
EBX_SAML2_DEFAULT_SP_LOGOUT_REQUEST_SIGNED	ebx.directory.saml2.default.sp.logout.request.signed

See SAML 2.0 [p 613] for more information.

SSO OpenID Connect connectivity

The EBX® Single Sign-On (SSO) OpenID Connect built-in can be configured through the following environment variables:

Name	TIBCO EBX® main configuration file equivalent
EBX_OIDC_DEFAULT_CLIENT_BASE_URL	ebx.directory.oidc.default.client.base.url
EBX_OIDC_DEFAULT_CLIENT_ID	ebx.directory.oidc.default.client.id
EBX_OIDC_DEFAULT_CLIENT_SECRET	ebx.directory.oidc.default.client.secret
EBX_OIDC_DEFAULT_DISCOVERY_URI	ebx.directory.oidc.default.discovery.uri
EBX_OIDC_DEFAULT_CLAIM_USERNAME	ebx.directory.oidc.default.claim.username
EBX_OIDC_DEFAULT_HTTP_CONNECT_TIMEOUT	ebx.directory.oidc.default.http.connect.timeout
EBX_OIDC_DEFAULT_HTTP_READ_TIMEOUT	ebx.directory.oidc.default.http.read.timeout

See OpenID Connect [p 614] for more information.

Bearer Authentication

The EBX® Bearer Authentication can be configured through the following environment variables:

Name	TIBCO EBX® main configuration file equivalent (spaces must be removed)
EBX_BEARER_DEFAULT_ENABLED	ebx.directory.bearer.default.enabled
EBX_BEARER_DEFAULT_JWT_JWKS_URI	ebx.directory.bearer.default.jwt.jwks.uri
EBX_BEARER_DEFAULT_JWT_ISSUERS	ebx.directory.bearer.default.jwt.issuers
EBX_BEARER_DEFAULT_SCOPE_DATASERVICES	ebx.directory.bearer.default.scope.dataservices
EBX_BEARER_DEFAULT_JWT_CLAIM_USERNAME	ebx.directory.bearer.default.jwt.claim.username
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_ENABLED	ebx.directory.bearer.default.jwt.encryption.enabled
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEY_URI	ebx.directory.bearer.default.jwt.encryption.key.uri
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEYSTORE_TYPE	ebx.directory.bearer.default.jwt.encryption.keystore.type
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEYSTORE_PATH	ebx.directory.bearer.default.jwt.encryption.keystore.path
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEYSTORE_PASSWORD	ebx.directory.bearer.default.jwt.encryption.keystore.password
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEYSTORE_KEY_ALIAS	ebx.directory.bearer.default.jwt.encryption.keystore.key.alias
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEYSTORE_KEY_PASSWORD	ebx.directory.bearer.default.jwt.encryption.keystore.key.password
EBX_BEARER_DEFAULT_INTROSPECTION_REQUEST	ebx.directory.bearer.default.introspection.request

See Bearer Authentication [p 580] properties for more information.

SCIM directory

The EBX® SCIM directory can be configured through the following environment variables:

Name	TIBCO EBX® main configuration file equivalent (spaces must be removed)
EBX_SCIM_DEFAULT_REQUEST_USERS_URL	ebx.directory.scim.default.request.users.url
EBX_SCIM_DEFAULT_REQUEST_USERS_HEADERS	ebx.directory.scim.default.request.users.headers
EBX_SCIM_DEFAULT_REQUEST_USERS_PAGE_SIZE	ebx.directory.scim.default.request.users.page.size
EBX_SCIM_DEFAULT_REQUEST_USERS_CACHE_MAX_SIZE	ebx.directory.scim.default.request.users.cache.max.size
EBX_SCIM_DEFAULT_REQUEST_USERS_CACHE_EXPIRATION	ebx.directory.scim.default.request.users.cache.expiration
EBX_SCIM_DEFAULT_REQUEST_GROUPS_URL	ebx.directory.scim.default.request.groups.url
EBX_SCIM_DEFAULT_REQUEST_GROUPS_HEADERS	ebx.directory.scim.default.request.groups.headers
EBX_SCIM_DEFAULT_REQUEST_GROUPS_PAGE_SIZE	ebx.directory.scim.default.request.groups.page.size
EBX_SCIM_DEFAULT_REQUEST_GROUPS_CACHE_MAX_SIZE	ebx.directory.scim.default.request.groups.cache.max.size
EBX_SCIM_DEFAULT_REQUEST_GROUPS_CACHE_EXPIRATION	ebx.directory.scim.default.request.groups.cache.expiration

See SCIM feature [p 578] properties for more information.

Email connectivity

The EBX® Mail service can be configured through the following environment variables :

Name	TIBCO EBX® main configuration file equivalent	Default	Description
EBX_SMTP_HOST	ebx.mail.smtp.host		SMTP server host name.
EBX_SMTP_PORT	ebx.mail.smtp.port		SMTP server port number.
EBX_SMTP_CONNECTION_TIMEOUT	ebx.mail.smtp.connectionTimeout	600000	SMTP socket connection timeout value in milliseconds.
EBX_SMTP_TIMEOUT	ebx.mail.smtp.timeout	600000	SMTP socket read timeout value in milliseconds.
EBX_SMTP_WRITE_TIMEOUT	ebx.mail.smtp.writeTimeout	600000	SMTP socket write timeout value in milliseconds.
EBX_SMTP_LOGIN	ebx.mail.smtp.login		SMTP server login id.
EBX_SMTP_PASSWORD	ebx.mail.smtp.password		SMTP server login password.
EBX_SMTP_SSL_ENABLED	ebx.mail.smtp.ssl.activate	true	Enables SSL. Value can be 'true' or 'false'.
EBX_WORKFLOW_MAIL_SENDER	ebx.manager.workflow.mail.sender		The workflow sender email. If not set, Workflows cannot send notifications.

More information on the used properties can be found in chapter Activating and configuring SMTP and emails [p 586] .

Memory configuration

Environment variables **JAVA_MEMORY_PERCENT** may be used to configure the percentage of the container memory that is assigned to the JVM the runs EBX®. It must be an integer value between 0 and 100.

If not set, a default value is used at startup.

Note

This variable is for advanced usage. Setting it too low or too high may cause runtime issues.

Authentication for REST services

Basic authentication for REST services is not enabled by default.

To enable this feature, set environment variable **EBX_REST_AUTHENTICATION_BASIC** to **true** .

Staging

The environment variable **EBX_STAGING_ACTIVATED** activates, if value is **true** , or deactivates, if value is **false** , the staging feature.

By default, staging is activated.

110.5 Configuration files

Two Java property files are currently used to configure EBX®.

On startup EBX® reads property files in the following order:

- **/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties**
- **/my_custom/conf/ebx-container.properties**

File ebx-default.properties

The file **/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties** sets default EBX® configuration properties for the container.

It should **never** be modified at runtime as this may prevent easily updating EBX® to a next version, instead use **/opt/ebx/conf/ebx-container.properties** .

File ebx-container.properties

The file **/opt/ebx/conf/ebx-container.properties** is by default empty. Any property value specified here will override the value set by **ebx-default.properties** .

This file is useful to change a property at runtime. To change a property at run time, create a new file, for example **/my_custom/conf/ebx-container.properties** , containing the new property values and mount de parent folder from the host to the container:

```
docker run -v /my-custom/conf:/opt/ebx/conf -p 8080:8080 -d ebx:6.2.3
```

For the list of properties supported by EBX® see chapter TIBCO EBX® main configuration file [p 571] .

110.6 **Volumes**

This image defines the following volumes:

Location	Description
/ebx/data	The EBX® root directory is located in this volume. It contains EBX® indexes and, when H2 embedded database is used, persisted data. It is recommended to use a high performance volume, for example an SSD. Using a shared network volume, for example a NFS mount, is not recommended and may cause performance and stability issues.
/ebx/logs	This volume is used for log files.
/ebx/temp	This volume is used for temporary files.

To start EBX® with all dedicated volumes, execute command:

```
docker run -d -p 8080:8080 \
 -v /my_custom/data:/ebx/data \
 -v /my_custom/logs:/ebx/logs \
 -v /my_custom/temp:/ebx/temp \
 ebx:6.2.3
```

Note

The volume **/ebx/data** should be mapped to a persistent volume even when an external database is used. If not, EBX® will have to rebuild its indexes on startup which may considerably increase boot time.

110.7 **Linux user and group**

The container is started using user **ebx** (uid 1500). User ebx's primary group is **root** (guid 0).

Note

Red Hat® OpenShift® may use another UID than 1500 when starting the container. For details see Red Hat® OpenShift® documentation.

110.8 **Host configuration**

It may be necessary to configure the Host so that the EBX® Container can reserve the resources required by numerous memory-mapped files.

On a Linux OS:

- Command **ulimit -n** should return a value equal or greater than **512000** .
- Command **sysctl vm.max_map_count** should return a value equal or greater than **262144** .

110.9 **Logs access**

Logs are sent to the stdout and stderr output streams and can be viewed using the following command:

```
docker logs <container-id>
```

Logs for both EBX® and Tomcat will be displayed.

Log files are also available under folder **/ebx/logs** :

- EBX® logs files are in folder **/ebx/logs/ebx**
- Tomcat logs files are in folder **/ebx/logs/tomcat** .

CHAPTER 111

Customizing the image

The EBX® Container Edition image can be used as a parent image to create a customized image.

This chapter contains the following topics:

1. User specified in Docker file
2. Setting default configuration
3. Adding a custom module
4. Adding a new locale
5. Adding a new JDBC driver
6. Configuring the database in JNDI
7. Configuring the mail in JNDI
8. Adding a driver for the Metadata Classifier database

111.1 User specified in Docker file

The EBX® Container Edition image's docker file specifies user **ebx** .

A Docker file extending the image may need to set current user to root and later switch back to **ebx** as in following sample:

```
FROM ebx:6.2.3
USER root
# Do something requiring being root...
USER ebx
```

111.2 Setting default configuration

Setting default EBX® configuration should not be based on **/opt/ebx/conf/ebx-container.properties** as this file may be overridden at runtime.

Instead, proceed as following in the Docker file:

- Rename file **/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties** , for example to **ebx-default-original.properties** .
- Create a new file **/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties** container new property values. This file must define property **ebx.file.previous** set to the original property file new name, for example **ebx-default-original.properties** .

For the list of properties supported by EBX®, see TIBCO EBX® main configuration file [p 571] .

Here is a sample Docker file that set the locale to "en-US"

```
FROM ebx:6.2.3
USER root

RUN mv /opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties \
       /opt/ebx/webapps/ebx/WEB-INF/ebx-default-original.properties

RUN echo "ebx.file.previous=ebx-default-original.properties" >> \
         /opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties
RUN echo "ebx.locales.available=en-US" >> /opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties

USER ebx
```

111.3 Adding a custom module

One can extend EBX® by developing custom modules. An EBX® module is a standard Jakarta EE web application, packaging various resources such as XML Schema documents, Java classes and static resources.

For more information on EBX® module, see Packaging TIBCO EBX® modules [p 879] .

With EBX® Container Edition, it is recommended to deploy modules as "unpacked" (exploded) WARs. This allows a faster startup and avoids unnecessarily increasing container size because Tomcat will not need to unpack WAR files.

The recommended way to add a module to an image is to:

- Copy the WAR to folder **/opt/ebx/webapps** . As stated previously, exploded format is recommended.
- Create an associated **Tomcat context XML file** named **module_war_name.xml** and copy it to folder **/opt/ebx/contexts.**
- Optionally, copy shared JARs to folder **/opt/ebx/lib** .

The **Tomcat context XML file** should have the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context docBase="${ebx.container.base}/webapps/module_war_name"/>
```

Using variable **${ebx.container.base}** is required for correct support of environment variable **EBX_ROOT_PATH** .

For more information on Tomcat contexts see https://tomcat.apache.org/tomcat-10.1-doc/config/context.html#Defining_a_context .

Here is a sample Docker file:

```
FROM ebx:6.2.3
USER root
COPY "./module_name.xml" "/opt/ebx/contexts"
COPY "./module_name" "/opt/ebx/webapps"
USER ebx
```

111.4 Adding a new locale

To add one or more locales, you need the language packs software [p 453] zip file(s). Here is a shell script that creates a Docker file and builds the target image. This script can be modified to suit your needs.

```
#!/bin/bash
#
# Build a docker image tag including language packs for
# a specified version available from the current folder.
#
# Usage: build.sh <lp-version> <from-tag> <build-tag>
#
```

```
set -e

LP_VERSION="$1"
FROM_TAG="$2"
BUILD_TAG="$3"

if (( $# != 3 )); then
  echo "Syntax $0 <lp-version> <from-tag> <build-tag>" 1>&2
  exit 1
fi

EBX_LOCALES=""

if [ -d files ]; then
  rm -rf files
fi

mkdir -p "files/lib"

for zip in TIB_ebx-lp_${LP_VERSION}_addon_*_languagepack*.zip ; do
  locale=$(echo ${zip} | sed -e 's/.*languagepack-\([a-zA-Z-]*\).zip/\1/g')
  EBX_LOCALES="${EBX_LOCALES},${locale}"
  echo unzip locale=${locale}
  unzip -oq ${zip}

  if [ -d ${locale} ] ; then
    mkdir -p "files/webapps/ebx-manager/www/${locale}"
    mv "${locale}/" "files/webapps/ebx-manager/www/"
  fi
done

mv *.jar "files/lib"

cat > Dockerfile << EOF
FROM ${FROM_TAG}

COPY "files" "/opt/ebx"

USER root

RUN mv "/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties" \
       "/opt/ebx/webapps/ebx/WEB-INF/ebx-default-original.properties" &&\
  echo "ebx.file.previous=ebx-default-original.properties" >> \
       "/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties" &&\
  echo "ebx.locales.available=en-US,fr-FR${EBX_LOCALES}" >> \
       "/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties"

USER ebx
EOF

docker build -t ${BUILD_TAG} .

echo "To run a container using this image, use command:"
echo " EBX_LOCALES=${EBX_LOCALES}"
echo " docker run -p 8080:8080 -d ${BUILD_TAG}"
```

In this example, libs and optional documentation are copied, then locales are set using the method described in the setting default configuration [p 653] section.

111.5 Adding a new JDBC driver

Adding a new JDBC driver is similar to adding a new library. You simply have to copy the jar file in the "/opt/ebx/lib" folder with the correct permission. Here is an example with the Oracle JDBC driver:

```
FROM ebx:6.2.3
USER root

ADD \
 https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc11/23.8.0.25.04/ojdbc11-23.8.0.25.04.jar \
 "/opt/ebx/lib/"

RUN chmod +r "/opt/ebx/lib/ojdbc11-23.8.0.25.04.jar"

USER ebx
```

See Database drivers [p 535] for more information.

111.6 Configuring the database in JNDI

To configure a JNDI data source for EBX®, you'll need to customize your Docker image. The following sample Dockerfile demonstrates the necessary steps to integrate your **ebx.xml** configuration.

```
FROM ebx:6.2.3
USER root

RUN mv "/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties" \
       "/opt/ebx/webapps/ebx/WEB-INF/ebx-default-original.properties" ; \
    mv "/opt/ebx/contexts/ebx.xml" \
       "/opt/ebx/contexts/ebx.original" ; \
  echo "ebx.file.previous=ebx-default-original.properties" >> \
       "/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties" ; \
  echo "ebx.persistence.url=" >> /opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties

COPY "./ebx.xml" "/opt/ebx/contexts"

USER ebx
```

The **ebx.xml** file manages the web application context and includes essential configuration information for the JNDI data source used by EBX®.

Critical point : each property listed below must be carefully adjusted, added, or removed depending on your database (e.g., Oracle, PostgreSQL, SQL Server, etc.) or your cloud provider.

Important : This sample file must be adapted to your specific environment before being integrated into your deployment image. UTF-8 encoding is required for the **ebx.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context docBase="${ebx.container.base}/webapps/ebx" crossContext="true">
    <Resource
        name="jdbc/EBX_REPOSITORY"
        auth="Container"
        type="javax.sql.DataSource"
        factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
        username="<username>"
        password="<password>"
        driverClassName="<driverClassName>"
        url="<url>"
        maxTotal="100"
        minIdle="10"
        maxIdle="25"
        initialSize="10"
        validationQuery="<validationQuery>"
        rollbackOnReturn="false"
        enableAutoCommitOnReturn="false"
    />
</Context>
```

Note

By configuring the JNDI data source, you're instructing EBX® to use that JDBC connection method. Consequently, EBX® will no longer consider environment variables beginning with 'EBX_DB_'.

111.7 Configuring the mail in JNDI

To configure a JNDI Mail session for EBX®, you'll need to customize your Docker image. The following sample Dockerfile demonstrates the necessary steps to integrate your **ebx.xml** configuration.

```
FROM ebx:6.2.3
USER root

RUN mv "/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties" \
       "/opt/ebx/webapps/ebx/WEB-INF/ebx-default-original.properties" ; \
    mv "/opt/ebx/contexts/ebx.xml" \
       "/opt/ebx/contexts/ebx.original" ; \
  echo "ebx.file.previous=ebx-default-original.properties" >> \
```

```
       "/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties" ; \
  echo "ebx.mail.activate=true" >> /opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties ; \
  echo "ebx.mail.smtp.host=" >> /opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties

COPY "./ebx.xml" "/opt/ebx/contexts"

USER ebx
```

The **ebx.xml** file manages the web application context and includes essential configuration information for the JNDI mail session used by EBX®.

Critical point : each property listed below must be carefully adjusted, added, or removed depending on your mail manager or your cloud provider.

Important : This sample file must be adapted to your specific environment before being integrated into your deployment image. UTF-8 encoding is required for the **ebx.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context docBase="${ebx.container.base}/webapps/ebx" crossContext="true">
    <Resource
        name="mail/EBX_MAIL_SESSION"
        auth="Container"
        type="jakarta.mail.Session"
        factory="org.apache.naming.factory.MailSessionFactory"
        mail.debug="false"
        mail.smtp.host="<host>"
        mail.smtp.port="<port>"
        mail.smtp.auth="true"
        mail.smtp.user="<user>"
        password="<password>"
        mail.smtp.ssl.enable="true"
        mail.smtp.starttls.enable="false"
        mail.smtp.connectiontimeout="600000"
        mail.smtp.timeout="600000"
        mail.smtp.writetimeout="600000"
        mail.transport.protocol="smtp"
        description="Mail session for Tomcat 10.x resource" />
    />
</Context>
```

Note

By configuring the JNDI mail session, you're instructing EBX® to use that mail connection method. Consequently, EBX® will no longer consider environment variables beginning with 'EBX_SMTP_'.

111.8 Adding a driver for the Metadata Classifier database

Adding a driver for the Metadata Classifier database requires installing the corresponding Python module. The Python module must be installed within the Python virtual environment. The following examples show how to customize the Docker image to add the driver.

For **PostgreSQL database** :

```
FROM ebx:6.2.3

RUN pipenv install psycopg2-binary
```

For **Oracle database** :

```
FROM ebx:6.2.3

RUN pipenv install oracledb
```

For **Microsoft SQL Server database** :

```
FROM ebx:6.2.3

USER root

# Install Microsoft SQL ODBC driver
RUN UBUNTU_VERSION=$(grep VERSION_ID /etc/os-release | cut -d '"' -f 2) && \
    echo "Detected Ubuntu version: $UBUNTU_VERSION" && \
    curl -sSL -O https://packages.microsoft.com/config/ubuntu/$UBUNTU_VERSION/packages-microsoft-prod.deb && \
    dpkg -i packages-microsoft-prod.deb && \
```

```
    rm packages-microsoft-prod.deb && \
    apt-get update && \
    ACCEPT_EULA=Y apt-get install -y msodbcsql18 && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

USER ebx
RUN pipenv install pyodbc
```

See SQLAlchemy dialects for more information about the database drivers that can be used with SQLAlchemy.

Technical administration

CHAPTER 112

Repository administration

This chapter contains the following topics:

1. Technical architecture
2. Auto-increments
3. Repository management
4. Monitoring management
5. Dataspaces

112.1 Technical architecture

Overview

The main principles of the TIBCO EBX® technical architecture are the following:

- A Java process (JVM) that runs EBX® is limited to a single EBX® repository. This repository is physically persisted in a supported relational database instance [p 527], accessed through a configured data source [p 574].
- A repository cannot be shared by multiple JVMs at any given time. However, a failover architecture may be used. These aspects are detailed in the sections Single JVM per repository [p 661] and Failover with hot-standby [p 661]. Furthermore, to achieve horizontal scalability, an alternative is to deploy a distributed data delivery (D3) [p 750] environment.
- A single relational database instance can support multiple EBX® repositories (used by distinct JVMs). It is then required that they specify distinct table prefixes using the property `ebx.persistence.table.prefix`.

See also

Configuring the EBX® repository [p 574]

Supported databases [p 527]

Data source of the EBX® repository [p 540]

Rules for the database access and user privileges

Attention

In order to guarantee the integrity of persisted master data, **it is strictly forbidden to perform direct SQL writes to the database**.

It is required for the database user specified by the configured data source [p 574] to have the 'create/alter' privileges on tables, indexes and sequences. This allows for automatic repository installation and upgrades [p 663].

See also

SQL access to history [p 464]

Accessing a replica table using SQL [p 471]

Data source of the EBX® repository [p 540]

Single JVM per repository

A repository cannot be shared by multiple JVMs. If such a situation was to occur, it would lead to unpredictable behavior and potentially even corruption of data in the repository.

EBX® performs checks to enforce this restriction. Before the repository becomes available, the repository must first acquire exclusive ownership of the relational database. After starting the repository, the JVM periodically checks that it still holds ownership of the repository.

These checks are performed by repeatedly tagging a technical table in the relational database. The shutdown command for the application server ensures that the tag on this technical table is removed. If the server shuts down unexpectedly, the tag may be left in the table. If this occurs, the server must wait several additional seconds upon restart to ensure that the table is not being updated by another live process.

Attention

To avoid an additional wait period at the next start up, it is recommended to always properly shut down the application server.

Failover with hot-standby

The exclusion mechanism described above is compatible with failover architectures, where only one server is active at any given time in an active/passive cluster. To ensure that this is the case, the main server must declare the property `ebx.repository.ownership.mode=failovermain`. The main server claims ownership of the repository database, as in the case of a single server.

A backup server can still start up, but it will not have access to the repository. It must declare the property `ebx.repository.ownership.mode=failoverstandby` to act as the backup server. It is not recommended to share the EBX® root directory, defined by the `ebx.repository.directory` property, with the active EBX® and standby EBX®. This means that, when the standby EBX® takes control, it starts on an empty directory and indexes have to be rebuilt, slowing down the first access to table data. To avoid this, you can replicate the EBX® repository folders. On Linux, running a timer on the failover node that executes `rsync` every few minutes to copy EBX® repository folder from the

active to the failover node has been successfully tested. It's important to stop synchronization before the failover node is started to prevent any concurrent modification of indexes.

> **Note**
>
> On Kubernetes, it is recommended to not use EBX® failover with hot standby and to rely instead on Kubernetes itself, which can automatically replace a failing POD with a new one that takes over the disks of the previous POD. Sample HELM charts are available on GitHub.

Once started, the backup server is registered in the connection log. Its status can be retrieved using the Java API or through an HTTP request, as described in the section Repository status information and logs [p 662] below.

In order to activate the backup server and transfer exclusive ownership of the repository to it, a specific request must be issued by an HTTP request, or using the Java API:

- Using HTTP, the request must include the parameter `activationKeyFromStandbyMode`, and the value of this parameter must be equal to the value declared for the entry `ebx.repository.ownership.activationkey` in the EBX® main configuration file. See Configuring failover [p 594].

  The format of the request URL must be:

  ```
  http[s]://<host>[:<port>]/ebx?activationKeyFromStandbyMode={value}
  ```

- Using the Java API, call the method `RepositoryStatus.wakeFromStandby`[API].

If the main server is still up and accessing the database, the following applies: the backup server marks the ownership table in the database, requesting a clean shutdown for the main server (yet allowing any running transactions to finish). Only after the main server has returned ownership can the backup server start using the repository.

Repository status information and logs

A log of all attempted Java process connections to the repository is available in the Administration area under 'History and logs [p 461]' > 'Repository connection log'.

The status of the repository may be retrieved using the methods in the `RepositoryStatus`[API] API.

It is also possible to get the repository status information using an HTTP request that includes the parameter **`repositoryInformationRequest`** with one of following values:

`state`	The state of the repository in terms of ownership registration. • `D`: Java process is stopped. • `O`: Java process has exclusive ownership of the database. • `S`: Java process is started in failover standby mode, but is not yet allowed to interact with the repository. • `N`: Java process has tried to take ownership of the database but failed because another process is holding it.
`heart_beat_count`	The number of times that the repository has made contact since associating with the database.
`info`	Detailed information for the end-user regarding the repository's registration status. The format of this information may be subject to modifications in the future without explicit warning.

112.2 Auto-increments

Several technical tables can be accessed in the 'Administration' area of the EBX® user interface. These tables are for internal use only and their content should not be edited manually, unless removing obsolete or erroneous data. Among these technical tables are:

Auto-increments	Lists all auto-increment fields in the repository.

112.3 Repository management

Installation and upgrades

Automatic installation and upgrades

By complying with the Rules for the database access and user privileges [p 661], the repository installation or upgrade is done automatically.

Inter-database migration

EBX® provides a way to export the full content of a repository to another database. The export includes all dataspaces, configuration datasets, and mapped tables. To operate this migration, the following guidelines must be respected:

- The source repository **must be shut down**: no EBX® server process must be accessing it; **not strictly complying with this requirement can lead to a corrupted target repository**;
- A new EBX® server process must be launched on the target repository, which must be empty. In addition to the classic Java system property `-Debx.properties`, this process must also specify `ebx.migration.source.properties`: the location of an EBX® properties file specifying the source repository. (It is allowed to provide distinct table prefixes between target and source.)
- The migration process will then take place automatically. Please note, however, that this process is not transactional: should it fail halfway, it will be necessary to delete the created objects in the target database, before starting over.
- After the migration is complete, an exception will be thrown, to force restarting the EBX® server process accessing the target repository.

Limitations:

- The names of the database objects representing the mapped tables (history, replication) may have to be altered when migrated to the target database, to comply with the limitations of its database engine (maximum length, reserved words, ...). Such alterations will be logged during the migration process.
- As a consequence, the names specified for replicated tables in the data model will not be consistent with the adapted name in the database. The first recompilation of this data model will force to correct this inconsistency.
- Due to different representations of numeric types, values for `xs:decimal` types might get rounded if the target database engine offers a lesser precision than the source. For example, a value of `10000000.1234567890123456789` in Oracle will get rounded to `10000000.12345678901234567` in SQL Server.

Repository backup

A global backup of the EBX® repository must be delegated to the underlying RDBMS. The database administrator must use the standard backup procedures of the underlying database.

Archives directory

Archives are stored in a sub-directory called `archives` within the `ebx.repository.directory` (see configuration [p 571]). This directory is automatically created during the first export from EBX®.

Attention

As specified in the security best practices [p 874], access to this directory must be carefully protected. Also, if manually creating this directory, make sure that the EBX® process has read-write access to it. Furthermore, the administrator is responsible for cleaning this directory, as EBX® does not maintain it.

Note

The transfer of files between two EBX® environments must be performed using tools such as FTP or simple file copies by network sharing.

Repository attributes

A repository has the following attributes:

repositoryId	Uniquely identifies a repository within the scope of the company. It is 48 bits (6 bytes) and is usually represented as 12 hexadecimal digits. This information is used for generating UUIDs (Universally Unique Identifiers) for entities created in the repository, as well as transactions logged in history tables or in the XML audit trail. This identifier acts as the 'UUID node' part, as specified by *RFC 4122*.
repository label	Provides a user-friendly label that identifies the purpose and context of the repository. For example: "Production environment".
store format	Identifies the underlying persistence system, including the current version of its structure.

Record deduplication

An issue with indices can occur where records in the same table are duplicated. EBX® provides a built-in service to resolve this issue. See the steps to run the service below.

Attention

While the service executes, you cannot access the EBX® repository through the UI, REST, or SOAP requests. Once completed, the service automatically shuts down the repository.

To remove duplicate records:

1. In the `ebx.properties` configuration file, set the `ebx.persistence.boot.checkRecordDuplicates` property to `true`.
2. Start the repository and wait for the service to automatically shut down the repository.
3. Set the `ebx.persistence.boot.checkRecordDuplicates` property to `false`.
4. Start your repository.

112.4 Monitoring management

Monitoring and cleanup of the relational database

Some entities accumulate during the execution of EBX®.

> **Attention**
> It is the *administrator's responsibility* to monitor and clean up these entities.

Database monitoring and organization

The persistence data source of the repository must be monitored through RDBMS monitoring.

The EBX® following tables allow data persistence in database:

- The table `{ebx.persistence.table.prefix}G_DSP`, in which each record represents a dataspace or a snapshot (its name is `EBXG_DSP` if the property `ebx.persistence.table.prefix` is unset).
- The table `${ebx.persistence.table.prefix}G_DST` holds dataset references.
- The table `{ebx.persistence.table.prefix}G_BLK`, where the data of EBX® tables are segmented into blocks sized up to 256 EBX® records.
- The tables {ebx.persistence.table.prefix}G_DTR, {ebx.persistence.table.prefix}G_TRV and {ebx.persistence.table.prefix}G_SHR, defining which blocks belong to a given EBX® table in a given dataspace or snapshot.
- The tables {ebx.persistence.table.prefix}G_TLR and {ebx.persistence.table.prefix}G_SCH, holds information about schemas and the tables they define.

Database statistics

The performance of requests executed by EBX® requires that the database has computed up-to-date statistics on its tables. Since database engines regularly schedule statistics updates, this is usually not an issue. Yet, it could be necessary to explicitly update the statistics in cases where tables are heavily modified over a short period of time (e.g. by an import creating many records).

History tables: impact on UI

For history tables, some UI components use statistics to adapt their behavior in order to prevent users from executing costly requests unwillingly.

For example, the combo box will not automatically search on user input if the table contains a large volume of records. This behavior may also occur if the database's statistics are not up to date, because a table may be considered as containing a large volume of records even if it is not actually the case.

Cleaning up dataspaces, snapshots, and history

A full cleanup of dataspaces, snapshots, and history from the repository involves several stages:

1. Closing unused dataspaces and snapshots to keep the cache to a minimal size.
2. Deleting dataspaces, snapshots, and history.
3. Purging the remaining entities associated with the deleted dataspaces, snapshots, and history from the repository.

Closing unused dataspaces and snapshots

In order to keep the cache and the repository to a reasonable size, it is recommended to close any dataspaces and snapshots that are no longer required. This can be done in the following ways:

- Through the user interface, in the 'Dataspaces' area.
- From the 'Dataspaces / Snapshots' table under 'Dataspaces' in the 'Administration' area, using the **Actions** menu in the workspace. The action can be used on a filtered view of the table.
- Through the Java API, using the method `Repository.closeHome`[API].
- Using the data service "close dataspace" and "close snapshot" operations. See Closing a dataspace or snapshot [p 1105] for more information.

Once the dataspaces and snapshots have been closed, the data can be safely removed from the repository.

> **Note**
>
> Closed dataspaces and snapshots can be reopened in the 'Administration' area, under 'Dataspaces'.

Deleting dataspaces, snapshots, and history

Dataspaces, associated history and snapshots can be permanently deleted from the repository. However, the deletion of a dataspace does not necessarily imply the deletion of its history. The two operations are independent and can be performed at different times.

> **Note**
>
> The deletion of a dataspace, a snapshot, or of the history associated with them is recursive. The deletion operation will be performed on every descendant of the selected dataspace.

After the deletion of a dataspace or snapshot, some entities will remain until a repository-wide purge of obsolete data is performed.

In particular, the complete history of a dataspace remains visible until a repository-wide purge is performed. Both steps, the deletion and the repository-wide purge, must be completed in order to totally remove the data and history. The process has been divided into two steps for performance issues. As the total clean-up of the repository can be time-intensive, this allows the purge execution to be initiated during off-peak periods on the server.

The process of deleting the history of a dataspace takes into account all history transactions recorded up until the deletion is submitted or until a date specified by the user. Any subsequent historized operations will not be included when the purge operation is executed. To delete new transactions, the history of the dataspace must be deleted again.

> **Note**
>
> It is not possible to set a deletion date in the future. The specified date will thus be ignored and the current date will be used instead.

The deletion of dataspaces, snapshots, and history can be performed in a number of different ways:

- From the 'Dataspaces/Snapshots' table under 'Dataspaces' in the 'Administration' area, using the **Actions** menu button in the workspace. The action can be used on a filtered view of the table.

- Using the Java API, and more specifically the methods `Repository.deleteHome`[API] and `RepositoryPurge.markHomeForHistoryPurge`[API].
- At the end of the data service "close dataspace" operation, using the parameters `deleteDataOnClose` and `deleteHistoryOnClose`, or at the end of a "merge dataspace" operation, using the parameters `deleteDataOnMerge` and `deleteHistoryOnMerge`.

Purging remaining entities after a dataspace, snapshot, or history deletion

Once items have been deleted, a purge can be executed to clean up remaining data from *all* deletions performed until that point. A purge can be initiated in the following ways:

- Through the user interface, by selecting in the 'Administration' area **Actions > Execute purge** in the navigation pane.
- Using the Java API, specifically the method `RepositoryPurge.purgeAll`[API].
- Using the task scheduler. See Task scheduler [p 723] for more information.

The purge process is logged in the directory `${ebx.repository.directory}/db.purge/`.

Cleaning up other repository entities

It is the *administrator's responsibility* to monitor and regularly cleanup the following entities.

Purge

A purge can be executed to clean up the remaining data from *all* deletions, that is, deleted dataspaces, snapshots and history performed up until that point. This includes the dataspaces and snapshots created for the persistent validation reports that have become obsolete. A purge can be initiated by selecting in the 'Administration' area **Actions > Execute purge** in the navigation pane.

Task scheduler execution reports

Task scheduler execution reports are persisted in the 'executions report' table, in the 'Task scheduler' section of the 'Administration' area. Scheduled tasks constantly add to this table as they are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

User interactions

User interactions are used by the EBX® component as a reliable means for an application to initiate and get the result of a service execution. They are persisted in the *ebx-interactions* administration section. It is recommended to regularly monitor the user interactions table, as well as to clean it, if needed.

Workflow history

The workflow events are persisted in the workflow history table, in the 'Workflow' section of the 'Administration' area. Data workflows constantly add to this table as they are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

The steps to clean history are the following

- Make sure the process executions are removed (it can be done by selecting in the 'Administration' area of Workflows **Actions > Terminate and clean this workflow** or **Actions > Clean from a date** in the navigation pane).

- Clean main processes in history (it can be done by selecting in the 'Administration' area of Workflows history **Actions > Clear from a date** or **Actions > Clean from selected workflows** in the navigation pane).
- Purge remaining entities in workflow history using 'standard EBX® purge'

See also *the standard EBX® purge* *[p 667]*

Monitoring and clean up of the file system

Attention

In order to guarantee the correct operation of EBX®, the disk usage and disk availability of the following directories must be supervised by the administrator, as EBX® does not perform any clean up, except for Lucene indexes:

- **Lucene indexes**: `${ebx.repository.directory}/indexes-(...)/`

  Lucene indexes: Indexes can require a lot of disk space; they are critical to the correct functioning of EBX®. In nominal usage, they must not be deleted or modified directly. However, there are cases where deleting these indexes might be needed:

  - If the repository is recreated from scratch, whereas the directory `${ebx.repository.directory}/` is preserved; to ensure consistency of data, it is then required to delete the root directory of the indexes.
  - More generally, if the indexes have become inconsistent with the repository data (this could happen in rare cases of bugs).

  After deletion, the content of the indexes will be lazily recomputed per table, derived from the content of the repository. The deletion must happen at the root folder of the indexes: if a single directory is deleted at a lower level, the global structure of the index will become inconsistent. This operation, however, has a cost, and should generally be avoided.
- **XML audit trail**: `${ebx.repository.directory}/History/`
- **Archives**: `${ebx.repository.directory}/archives/`
- **Logs**: ebx.logs.directory [p 582]
- **Temporary directory**: ebx.temp.directory [p 581]

Attention

For **XML audit trail**, if large transactions are executed with full update details activated (contrary to the default setting), the required disk space can increase.

Attention

For pagination in the data services `getChanges` operation, a persistent store is used in the **Temporary directory**. Large changes may require a large amount of disk space.

See also

XML audit Trail

Tuning the EBX® repository *[p 594]*

112.5 Dataspaces

Some dataspace administrative tasks can be performed from the 'Administration' area of EBX® by selecting 'Dataspaces'.

Dataspaces/snapshots

This table lists all the existing dataspaces and snapshots in the repository, whether open or closed. You can view and modify the information of dataspaces included in this table.

See also *Dataspace information* *[p 110]*

From this section, it is also possible to close open dataspaces, reopen previously closed dataspaces, as well as delete and purge open or closed dataspaces, associated history, and snapshots.

See also *Cleaning up dataspaces, snapshots, and history* *[p 666]*

Dataspace permissions

This table lists all the existing permission rules defined on all the dataspaces in the repository. You can view the permission rules and modify their information.

See also *Dataspace permissions* *[p 111]*

Repository history

The table 'Deleted dataspaces/snapshots' lists all the dataspaces that have already been purged from the repository.

From this section, it is also possible to delete the history of purged dataspaces.

CHAPTER 113

UI administration

TIBCO EBX® comes with a full user interface called Advanced perspective [p 672] that includes all available features. The interface is fully customizable [p 676] (custom logo, colors, field size, default values, etc.) and available to built-in administrators.

Access to the advanced perspective can be restricted in order to simplify the end-user experience, through global permissions [p 671], giving the possibility to grant or restrict access to functional categories. Administrators can create simplified perspectives called recommended perspectives [p 684] for end-users, containing only the features and menus they need for their daily tasks.

This chapter contains the following topics:

1. Global permissions
2. Advanced perspective
3. Recommended perspectives
4. Custom views
5. User session management
6. Accessibility mode

113.1 Global permissions

Global permission rules can be created in EBX®.

The 'Display area' property allows restricting access to areas of the user interface. To define the access rules, select 'Global permissions' in the 'Administration' area.

Profile	Indicates on which profile the rule will be applied.
Restriction policy	Indicates if the permissions defined here restrict the ones defined for other profiles. See the Restriction policy concept [p 501] for more information.
Dataspaces	Defines permissions for the Dataspaces area.
Data Models	Defines permissions for the Data Models area.
Workflow Models	Defines permissions for the Workflow Models area.
Data Workflows	Defines permissions for the Data Workflows area.
Data Services	Defines permissions for the Data Services area. Independently, it is also possible to: • Defines the access permissions for the REST built-in connector HTTP(S). This setting does not impact the REST Toolkit applications. • Defines the access permissions for the REST OpenAPI services. • Defines the access permissions for the SOAP connector HTTP(S) and JMS. • Defines the access permissions for the WSDL connector HTTP(S).
Administration	Defines permissions for the Administration area.

> **Note**
>
> Permissions can be defined by administrators and by the dataspace or dataset owner.

113.2 Advanced perspective

The advanced perspective and its parameterization are unique. It is the parent perspective from which any new perspective [p 684] will inherit.

Children perspectives can be created from that main perspective in order to offer a customized, simplified menu to the end-users. Thanks to dataset inheritance, these simplified perspectives will receive their parameters from the advanced perspective (the root dataset). These parameters can then be overridden on the newly created simplified perspectives. Simplified perspectives can be created underneath an existing simplified perspective, thus inheriting from the parent's parameters.

See also *Inheritance* *[p 37]*

The advanced perspective is available by default to all end-users but access can be restricted.

Note: Administrators can always access the advanced perspective even when it is deactivated.

It is possible to configure which perspective is applied by default when users log in. This 'default perspective' is based on two criteria: 'recommended perspectives', defined by administrators and 'favorite perspectives', defined by users.

See also

Recommended perspectives *[p 684]*

Favorite perspectives *[p 23]*

Perspective creation

To create a perspective, open the 'Select an administration feature' drop-down menu and click on the + sign to create a child dataset.

See also *Creating an inheriting child dataset* *[p 126]*

User interface

Options are available in the Administration area for configuring the web interface, in the 'User interface' section.

Attention

Be careful when configuring the URL policy (deprecated) [p 675]. If the web interface configuration is invalid, it can lead to the unusability of EBX®. If this occurs, use the "rescue mode" by setting `frontEnd.rescueMode.enable=true` in EBX® main configuration file [p 571], and accessing the following URL in your browser as a built-in administrator user: `http://.../ebx/?onwbpID=iebx-manager-rescue`.

Session configuration

These parameters configure the user session options:

User session default locale	Default session locale
Session time-out (in seconds)	Maximum duration of user inactivity before the session is considered inactive and is terminated. A negative value indicates that the session should never timeout.

Interface configuration

Entry policy (deprecated)

> **Attention**
> This screen is deprecated. It is now recommended to configure those options in the `ebx.properties` file.

See also *Security configuration* *[p 593]*

Describes the URL to access the application.

Login URL	If the user is not authenticated, the session is forwarded to this URL.

The entry policy defines an EBX® login page, replacing the default one.

If defined,

- it replaces an authentication URL that may have been defined using a specific user `Directory`[API],
- it is used to build the permalinks in the user interface,
- if the URL is full, that is, starting with `http://` or `https://`, it replaces the URL of the workflow email configuration.

URL policy (deprecated)

Describes the URL and proxy policy. Both dynamic (servlet) and static (resources) URLs can be configured.

Attention

This configuration method is deprecated. Navigate to *Administration > User interface > User interface configuration > Advanced perspective > Interface configuration > URL Policy* and revert any existing settings. To update the URL computing configuration, you can edit the `ebx.properties` file. See URLs computing [p 590] for more information.

HTTP servlet policy	Header content of the servlet HTTP request: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.
HTTPS servlet policy	Header content of the servlet HTTPS request: • if a field is not set, the default value is chosen (in an environment configuration), • if a default value is not set, the value in the initial request is used.
HTTP external resources policy	Header content of the external resources URL in HTTP: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.
HTTPS external resources policy	Header content of the external resources URL in HTTPS: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.

Exit policy (deprecated)

Attention

This screen is deprecated. It is now recommended to configure those options in the `ebx.properties` file.

See also *Security configuration [p 593]*

Describes how the application is exited.

Normal redirection	Specifies the redirection URL used when exiting the session normally.
Error redirection	Specifies the redirection URL used when exiting the session due to an error. This feature is now deprecated and may be ignored by EBX®.
Redirection restrictions	Specifies the list of authorized domains and whether HTTPS is mandatory for each domain.

Graphical interface configuration

Activation & Allowed profiles

The 'Activated' radio button allows to activate or deactivate the perspective. When deactivated, the perspective will only be made available to the administrator.

The 'Allowed profiles' feature is used to give access to the perspective to a given profile. Several profiles can be added to the list of authorized profiles by clicking on the + icon below the numbered list.

The available perspective properties are:

Activated	Indicates if the perspective is visible to authorized users.
Allowed profiles	The list of authorized user profiles for the perspective.
Allowed devices	The list of authorized devices for the perspective. If not specified, only "EBX® Web Application" can display this perspective.
Default selection	The menu item that is selected by default. This property is not available for the advanced perspective.

Application locking

EBX® availability status:

Availability status	This application can be closed to users during maintenance (but still remain open to administrators). Takes effect immediately.
Unavailability message	Message displayed to users when access is restricted to administrator profiles.

Security policy

EBX® access security policy. These parameters only apply to new HTTP sessions.

IP access restriction	Restricts access to designated IP addresses (see IP pattern below).
IP restriction pattern	Regular expression representation of IP addresses authorized to access EBX®. For example, `((127\.0\.0\.1) | (192\.168\.*\.*))` grants access to the local machine and the network IP range `192.168.*.*`.
Unique session control	Specifies whether EBX® should control the uniqueness of user sessions. When set to 'Yes', if a user does not log out before closing the browser, it will not be possible for that user to log in again until the previous session has timed out.

Ergonomics and layout

EBX® ergonomics parameters:

Max table columns to display	According to network and browser performance, adjusts the maximum number of columns to display in a table. This property is not used when a view is applied on a table.
Maximum auto-width for table columns	Defines the maximum width to which a table column can auto-size during table initialization. This is to prevent columns from being too wide, which could occur for very long values, such as URLs. Users will still be able to manually resize columns beyond this value.
Max expanded elements for a hierarchy	Defines the maximum number of elements that can be expanded in a hierarchy when using the action "Expand all". A value less than or equal to '0' disables this parameter.
Default table filter	Defines the default table filter to display in the filters list in tabular views. If modified, users must log out and log in for the new value to take effect.
Display the message box automatically	Defines the message severity threshold for displaying the messages pop-up.
Forms: width of labels	The width of labels in forms.
Forms: width of inputs	The width of form input fields in forms.
Forms: height of text areas	The height of text entry fields in forms.
Forms: aggregated lists	The number of hidden candidate lines to be generated, available to create new instances in the list.
Forms: minimum width of HTML editor	The minimum width of HTML editors in forms. To have an editor with a fixed width, set the same value than the maximum width.
Forms: maximum width of HTML editor	The maximum width of HTML editors in forms.
Forms: minimum height of HTML editor	The minimum height of HTML editors in forms. To have an editor with a fixed height, set the same value than the maximum height.

Forms: maximum height of HTML editor	The maximum height of HTML editors in forms. In forms, this value won't exceed workspace height to avoid scrolling issues.
Searchable list selection page size	Maximum number of rows downloaded at each request of the searchable list selection (used for selecting foreign keys, enumerations, etc.).
Record form: rendering mode for nodes	Specifies how to display non-terminal nodes in record forms. If this property is modified, users are required to log out and log in for the new value to take effect.
Record form: display of selection and association nodes in creation mode	If enabled, the selection and association nodes will be displayed in record creation forms.
Display density	Defines the default display density mode for all users. If no density has been selected by the user yet, this value will be applied. Conversely, if the user already chose a density, their choice will prevail.
Avatar displayed in the header	This property defines the display mode of avatars in the header. For example, it is possible to enable or disable the use of avatars in the header by updating this property. If no value is defined, the default value is 'Avatar only'. If it is a relative path, prefix it with "../" to get back to the application root URL.
Avatar displayed in the history	This property defines the display mode of avatars in the history. For example, it is possible to enable or disable the use of avatars in the history by updating this property. If no value is defined, the default value is 'Avatar only'. If it is a relative path, prefix it with "../" to get back to the application root URL.
Avatar displayed in the workflow	This property defines the display mode of avatars in the workflow. For example, it is possible to enable or disable the use of avatars in the workflow by updating this property. If no value is defined, the default value is 'Avatar only'. If it is a relative path, prefix it with "../" to get back to the application root URL.

Default option values

Defines default values for options in the user interface.

Import/Export

CSV file encoding	Specifies the default character encoding to use for CSV file import and export.
CSV : field separator	Specifies the default separator character to use for CSV file import and export.
CSV : list separator	Specifies the default list separator character to use for CSV file import and export.
Import mode	Specifies the default import mode.
Missing XML values as 'null'	If 'Yes', when updating existing records, if a node is missing or empty in the imported file, the value is considered as 'null'. If 'No', the value is not modified.

Colors and themes

Customizes EBX® colors and themes.

Web site icon URL (favicon)	Sets a custom favicon.
Logo URL (SVG)	Specifies the SVG image used for compatible browsers. Leave this field blank to use the PNG image, if specified. The user interface will attempt to use the specified PNG image if the browser is not SVG-compatible. If no PNG image is specified, the GIF/JPG image will be used. The logo must have a maximum height of 40px. If the height exceeds 40px, it will be cropped, not scaled, to a height of 40px. The width of the logo will determine the position of the buttons in the header. If it is a relative path, prefix it with "../" to get back to the application root URL.
Logo URL (PNG)	Specifies the PNG image used for compatible browsers. Leave both this field and the SVG image URL field blank to use the GIF/JPG image. The user interface will use the GIF/JPG image if the browser is not PNG-compatible. The logo must have a maximum height of 40px. If the height exceeds 40px, it will be cropped, not scaled, to a height of 40px. The width of the logo will determine the position of the buttons in the header. If it is a relative path, prefix it with "../" to get back to the application root URL.
Logo URL (GIF/JPG)	Specifies the GIF/JPG image to use when neither the PNG nor SVG are defined. The recommended formats are GIF and JPG. The logo must have a maximum height of 40px. If the height exceeds 40px, it will be cropped, not scaled, to a height of 40px. The width of the logo will determine the position of the buttons in the header. If it is a relative path, prefix it with "../" to get back to the application root URL.
Main	Main user interface theme color, used for selections and highlights.
Header	Background color of the user interface header. By default, set to the same value as the Main color.
Workflow badge	Background and text/outline colors of new workflow task counters.
Primary buttons	Color of buttons selected by default. By default, set to the same value as the Main color.

Text of link style buttons	Text color of some buttons having a link style (the text is not dark or light, but colored). By default, set to the same value as the main color.
Selected tab border	Border color of the selected tab. By default, set to the same value as the Main color.
Table history view: technical data	Background color of technical data cells in the table history view.
Table history view: creation	Background color of cells with the state 'creation' in the table history view.
Table history view: deletion	Background color of cells with the state 'deletion' in the table history view.
Table history view: update	Background color of cells with the state 'update' in the table history view.

Child perspective menu

An unlimited number of child perspectives can be created. Child perspectives inherit from the parameters of the 'Advanced perspective'. Some of these parameters can be overridden as detailed hereafter.

Activation & Allowed profiles

See Activation and Allowed profiles for the Advanced perspective [p 676] for more information.

> **Note**
>
> Any specific parameter set for this perspective will override the default parameters that have been set in the 'Advanced perspective' configuration.

Perspective Menu

This view displays the perspective menu. It is a hierarchical table view.

From this view, a user can create, delete or reorder menu item records.

See also *Hierarchical table view [p 37]*

Section Menu Item	This is a top level menu item. It contains other menu items.
Menu group	This is a container for other menu items.
Action Menu Item	This menu item displays a user service in the workspace area.

Menu item properties

When creating a record in the 'Perspective' Menu, the available perspective properties are:

Type	The menu item type. **See also** *Menu item types* *[p 682]*
Parent	The parent of the menu item. This property is not available for section menu items.
Label	The menu item label. The label is optional for action menu items. If not specified, the label will be dynamically generated by EBX® when the menu item is displayed.
Allowed devices	The list of authorized devices for this item. If not specified, all devices can display this menu item. Currently only two devices are supported: "EBX® Web Application" and "EBX® GO".
Icon	The icon for the menu item. Icon can be either "standard" (provided by EBX®) or an image, specified by a URL, that can be hosted on any web server. Icons size should be 16x16 pixels. This property is not available for section menu items.
Top separator	Indicates that the menu item section has a top separator. This property is only available for section menu items.
Action	The user service to execute when the user clicks on the menu item. **See also** *User interface services* *[p 1027]* If an end-user is allowed to view the perspective but not to execute the user service, an "access denied" message will be displayed when the user clicks on the menu item. This property is only available for action menu items.
Selection on close	The menu item that will be selected when the service terminates. Built-in services use this property when the user clicks on the 'Close' button.

This property is only available for action menu items.

Ergonomics and layout

See Ergonomics and layout for the Advanced perspective [p 678] for more information.

> **Note**
>
> Any specific parameter set for this perspective will override the default parameters that have been set in the 'Advanced perspective' configuration.

Colors and themes

See Colors and themes for the Advanced perspective [p 681] for more information.

> **Note**
>
> Any specific parameter set for this perspective will override the default parameters that have been set in the 'Advanced perspective' configuration.

113.3 Recommended perspectives

It is possible for a perspective administrator to configure recommended perspectives dedicated to a specific audience. These recommended perspectives are a way to choose which perspective is applied by default when a user logs in, based on their role.

However, users always have the possibility to switch between the various perspectives that are available to them and to set one as their favorite. See Perspective modes [p 23] for more information.

To configure recommended perspectives, go to *User interface* > *Recommended perspectives* > *Manage recommended perspectives*.

Managing recommended perspectives

The main screen shows an ordered list of records associating a profile with a perspective. Note that the order here is important since a user can match more than one record (see Resolution [p 684] for more information).

- To add an entry, use the 'Create' action.
- To edit an entry, first select it in the list by clicking on it, then click on the 'Edit' action, or simply double-click on it.
- To remove an entry, first select it in the list, then click on the 'Delete' action.
- To move an entry, first select it in the list, then use the actions in the toolbar to the right of the list.

Resolution

When a user logs in, the following algorithm determines which perspective is selected by default:

```
// 1) favorite perspective
IF the user has a favorite perspective
AND this perspective is active
AND the user is authorized for this perspective
    SELECT this perspective
    DONE

// 2) recommended perspective
FOR EACH association in the recommended perspectives list, in the declared order
    IF the user is in the declared profile
```

```
    AND the associated perspective is active
    AND the user is authorized for the associated perspective
        SELECT this perspective
        DONE

// 3) advanced perspective
IF the advanced perspective is active
AND the user is authorized for this perspective
    SELECT this perspective
    DONE

// 4) any perspective
SELECT any active perspective for which the user is authorized
DONE
```

113.4 Custom views

Users can create and manage custom views directly from the 'View' menu on tables. This administration section is the central point to manage these custom views.

Views

This table contains all custom views defined on any table. Only a subset of fields is editable:

Documentation	Localized labels and descriptions.
Owner	Defines the user(s) owning and authoring this view definition.
View group	Indicates the menu group in which this view is displayed in the 'View' menu.
Share with	Defines the users allowed to select this view from their 'View' menu.

Views permissions

This table allows to manage permissions relative to custom views, by data model and profile. The following permissions can be configured (the default value is applied when no permission is set for a given user):

Permission	Description	Default value
Recommend views	Allows the user to manage recommended views.	If the user is the dataset owner, the default value is 'Yes', otherwise it is 'No'.
Manage views	Defines the views the user can modify and delete.	If the user is a built-in administrator, the default value is 'Owned + shared', otherwise it is 'Owned'.
Share views	Defines the views for which the user can edit the 'Share with' field.	If the user is a built-in administrator, the default value is 'Owned + shared', else if the user is the dataset owner, it is 'Owned', otherwise it is 'None'.
Publish views	Allows the user to publish views to make them available to all users using Web components, workflow user tasks, or data services.	If the user is a built-in administrator, the default value is 'Yes', otherwise it is 'No'.

113.5 User session management

This tool lists all user sessions and allows terminating active sessions when necessary.

For example: it is possible to invalidate and terminate all currently open and active sessions for maintenance purposes. The access to the user interface can be temporarily closed, with an unavailability message being displayed, through Application locking [p 676]. After active sessions are terminated, users will not be able to reconnect and will see the unavailability message. The maintenance operation can then be performed.

113.6 Accessibility mode

An accessibility mode is available to help users with disabilities, when active it will tweak display to ensure that, wherever possible, the user interface is accessible.

The option can be found on the login screen, and also in the user pane once they are connected.

The selected value is persisted per user and shared among all its work sessions.

Allows to make all the controls of the form operable through a keyboard interface and thus conform to the WCAG Success Criterion 2.1.1.

Overview of changes applied when accessibility mode is on:

Tabular view	Add an "Access data" button to all row toolbars. Table header cells and their documentation buttons can be focused with keyboard.
HTML editor	Make the toolbar being always expanded.
Forms	Expand/collapse and documentation buttons can now be focused.

Supported screen reader

The Jaws (version 2022) is the only screen reader supported by TIBCO EBX®. Any other technology may have some unexpected behavior.

CHAPTER 114

UI – Workflow launcher

This chapter contains the following topics:

1. Introduction
2. Workflow launcher in data section
3. Creating and setting a launcher
4. Activating workflow launcher
5. Launching a workflow
6. Adding a workflow launcher to the custom toolbar at table top
7. Access a launcher after workflow model modification

114.1 Introduction

A **Workflow Launcher** is a **user service** for **launching workflows in TIBCO EBX®** directly from the data section without passing by the data workflow's inbox. This feature does not create workflow publications but launches existing ones. It offers several advantages, including the ability to launch workflow publication directly from the **data section** (table view, hierarchy view or record form view). In this way, the user experience is improved by avoiding to the user shifting his attention back and forth between the data section and the data workflow section. Hence, the user can launch a workflow while still focusing on his main task.

The second advantage is that it allows to launch the same workflow publication from any data selection. Thanks to the **dynamic mapping** of the workflow data context [p 40] with the current data selection. The dynamic mapping offers the possibility to initialize the data context inputs at launch time. Hence, in order to launch the same workflow from n different data selections, it is not longer necessary to duplicate n times the same workflow model with different data selections or to provide an initial user service to configure the data to select.

The last solution is a programmatic solution which would solve the previously cited problems, however it is not the ideal solution because it does not fulfill the commitment zero line of code.

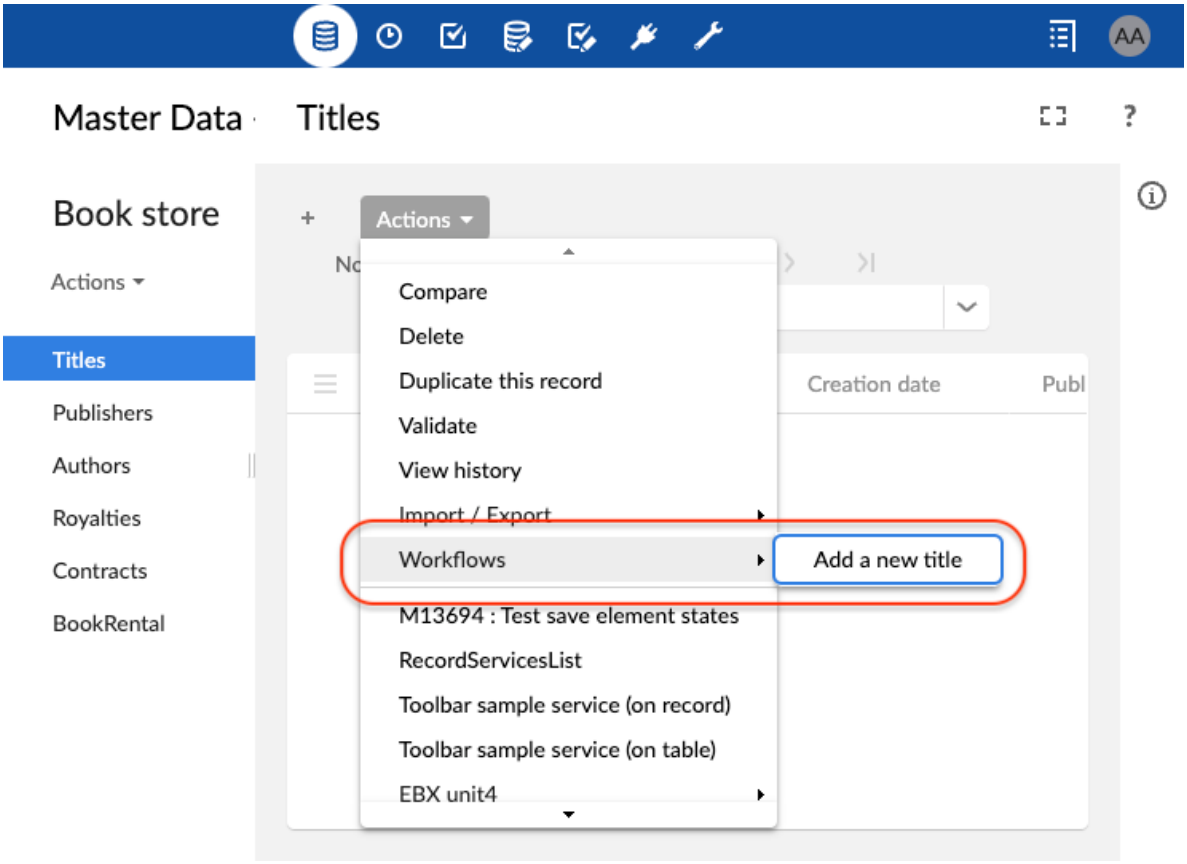
Understanding this feature requires familiarization with the following concepts:

Concept	Description/Link
Workflow model	See documentation [p 39]
Data context	See documentation [p 40]
Workflow publication	See documentation [p 40]
Publication name	A unique identifier of a workflow publication
Data Workflow	See documentation [p 40]
Data section	Is the data user interface [p 123] which displays the datasets and tables in EBX®. it is accessible from the main header.
Workflow launchers dataset	It contains two tables **Launchers** and **Activations**. With this dataset the user can **configure** the launchers of workflow publications and **activate** them for a particular user(s) and table(s). It is available in the **Administration area** in a section called **workflow management**.
Launcher	An entity which is used by the service **workflow launcher** to identify the **workflow publication** to launch and how to initialize its **data context**.
Launchers	A table in the workflow launchers dataset, each record is a **Launcher** type. In order to be able to launch a workflow publication from a data section, a launcher which points to a workflow publication should be added in this table.
Activation	An entity which hides or shows the launcher of a workflow publication for a given user profile(s) and for a particular table(s) from the data section.
Activations	A table in the workflow launchers dataset, each record is an **Activation** type. In order to make a workflow available on the toolbar of a particular table and for a particular user, a record with the corresponding launcher should be created in this table.
Dynamic mapping of data context	Is the process of initialization of data context input variables when launching a workflow publication. Before, the values of the data context were static defined at modeling phase or set dynamically with Java coding, but now it is possible to set dynamically the values of the data contexts, i.e. at launch time, with zero line of code.
Reserved variables for data context	A set of reserved variables. These variables define which data selection should be mapped with a data context variable. The possible values are: **${dataspace}**, **${dataset}**, **${xpath(table)}** and **${xpath(record)}**. For example, if the value **${xpath(table)}** is used for a data context variable, this means that this variable will be mapped with the current adaptation table reference, at launch time.

114.2 **Workflow launcher in data section**

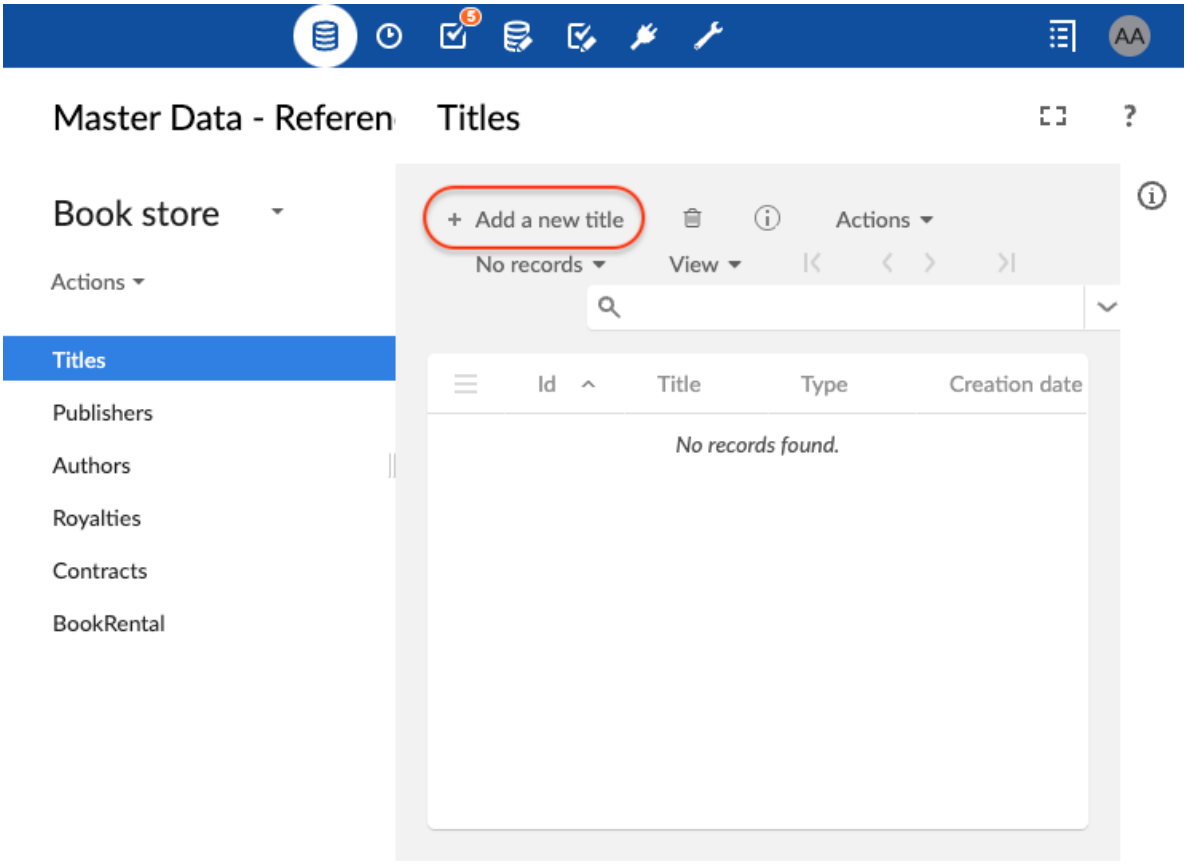
As previously stated, workflow launchers are now available directly on the toolbar of tables, table rows, records and hierarchies. There are a number of ways to display a workflow launcher on the

toolbar of a table in data section. The display depends on the type of the toolbar (default toolbar or custom toolbar in the DMA) and whether or not smart filtering [p 89] is applied. In the case of default toolbar, the action menu displays all the workflow launchers in a separate submenu called **Workflows** (see the screenshot below).



In the case of a custom toolbar,it is possible to define an action button (see the screenshot below) or action menu item for a particular workflow launcher. And finally, if the smart filtering policy is

activated, then all the workflow launchers that are displayed using an action button or action menu item will not appear in the Action menu.



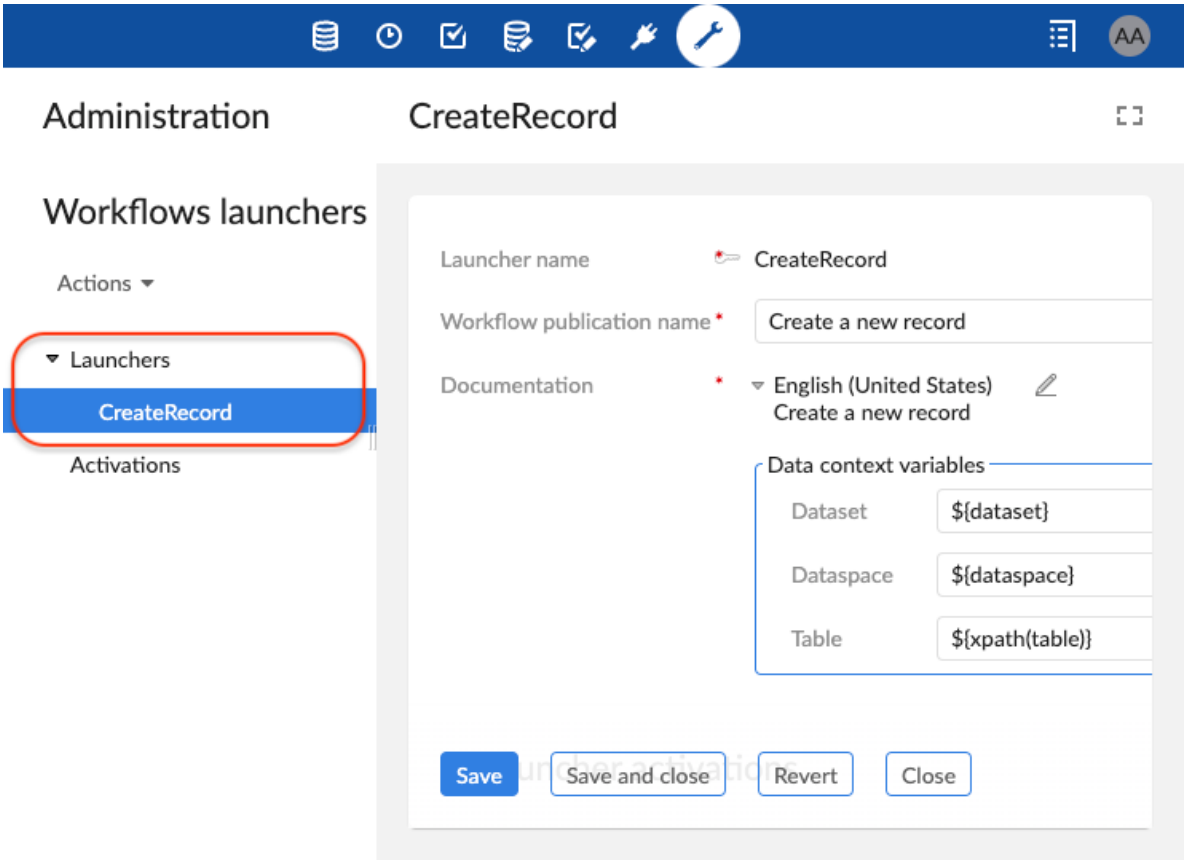
In order to display a workflow launcher on a toolbar, first, a **launcher** must be **created** and **configured** (see creating and setting workflow launcher [p 693] section), then, an **activation** should be **created** for this launcher (see activating workflow launcher [p 695] section). Note that, if the current user has no rights to launch a workflow publication [p 219] then the workflow launcher will not be available on the toolbar. Also, if the workflow publication is deleted or if there are any errors or warnings in the configuration or activation of the launcher, then the corresponding workflow launcher will not be available on the toolbar.

The title and tooltip of the button that will be displayed on the toolbar, are computed in the following order of priority: the custom documentation of the launcher activation is used, otherwise the documentation of the launcher is used. If description is left empty, then the following description is used "This user service will launch a data workflow.".

> **Note**
>
> Particular attention should be paid for the workflow launchers which are **available on the record form and table rows**. Only the **launchers which requires record selection** are displayed on the toolbar of a record form and table rows. A workflow launcher which requires record selection is the one for which one of the data context variable is mapped with the reserved variable **${xpath(record)}** in the configuration of its launcher.

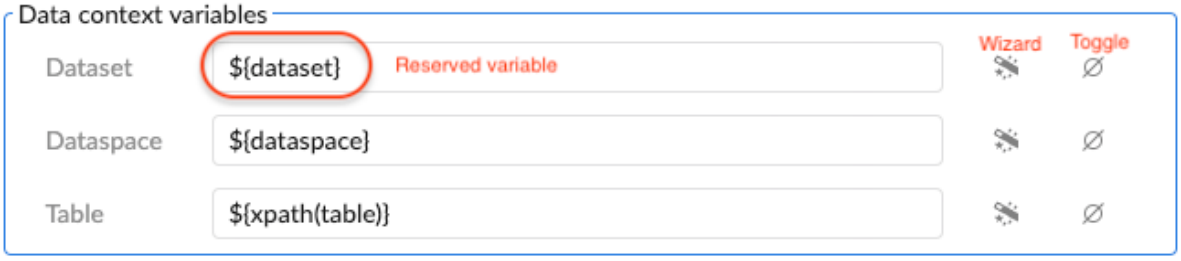
114.3 Creating and setting a launcher

A launcher [p 693] is the entity which is used by the service Workflow Launcher to identify the workflow publication to launch and how to initialize its Data context. In order to create a launcher for a particular workflow publication, first navigate to the administration area, then the workflow management section, select the workflow launchers dataset, select the launchers table, and then add a new record (see below form).

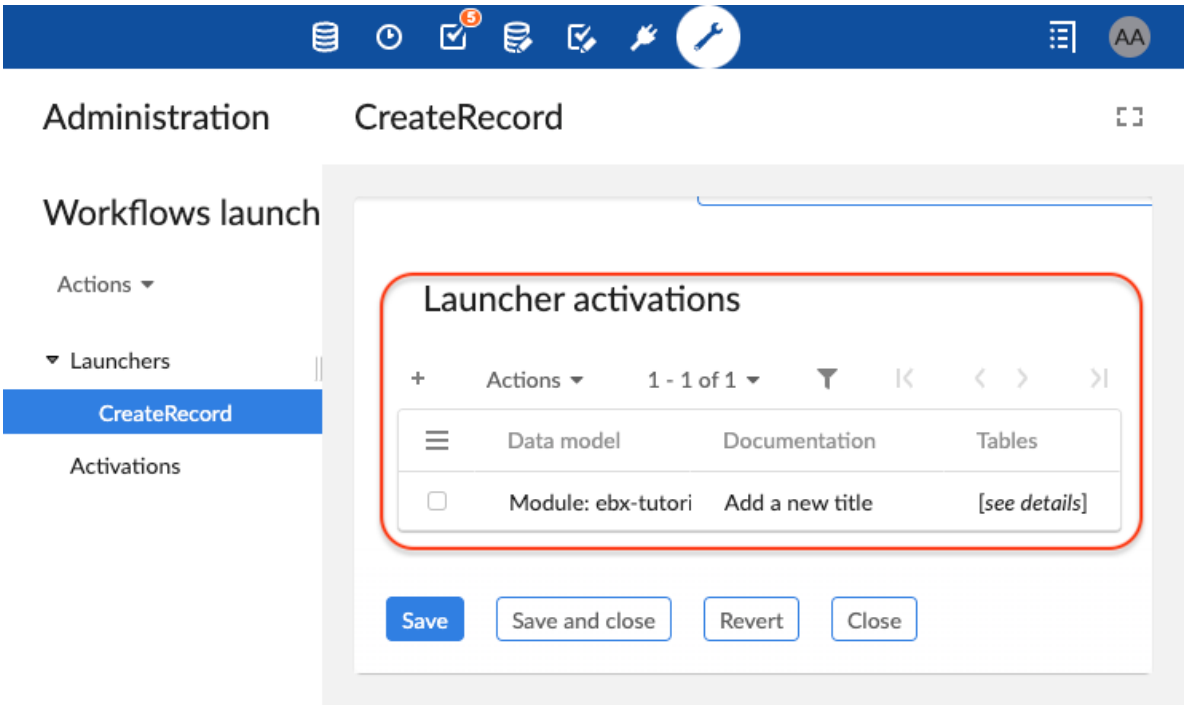


The second step, is to setup the following fields of the record:

Field name	Description
Launcher name	A unique identifier. It is used to select a launcher in the activation phase [p 695].
Workflow publication name	Defines the workflow publication that will be launched when the user service "workflow launcher" is executed. The workflow model should be first defined and published to be available in the list.
Data context variables	This field contains the list of input variables which were defined in the **data context configuration** [p 215] phase of a workflow model. Each line is composed of a label, a value, and a toggle button to switch between default and overwritten value (see the screenshot about the data context [p 694] in the launcher form). The label is the name of the variable set by the user in the data context of the workflow model. By default, the value of each variable is the default one set in the **data context**. If the default value field is left empty in the data context, then the value of the variable is set to **undefined**. If the toggle button is set on overwritten value mode, then a wizard is made available on the right of the input field which allows to select a reserved variable. When overwriting the value of a data context variable, two options are possibles: the override value may be a **constant** or a **reserved variable**. If a constant is used, then the value of the data context variable will not depend on the entity selection at launch time. However, if a reserved variable is used, such as **${dataspace}**, the value of the corresponding variable will be **mapped** to the **current entity selection**, for example the data context variable which is assigned to **${dataspace}** will be initialized with the **current dataspace** at launch time. **Note** If there are no input variables in the data context of the workflow model, then this field should be hidden. If the workflow publication name field changes then this input is updated automatically and displays a new list of data context.
Documentation	A documentation is composed of a label and a description. The default value of the label is the publication name which is inherited from the field "workflow publication name" and the default value of the description is empty. If the "Publication name" field changes then the documentation will update automatically." The label and description can be overridden if needed thanks to the edit button on the right of the documentation widget." The value of the documentation will be inherited by the activation. Note that the documentation defined in the activation has a higher priority over the one defined in the launcher.



At the bottom of a launcher record, a table of all activations of the current launcher is displayed (see the section Launcher activations in the screenshot below).

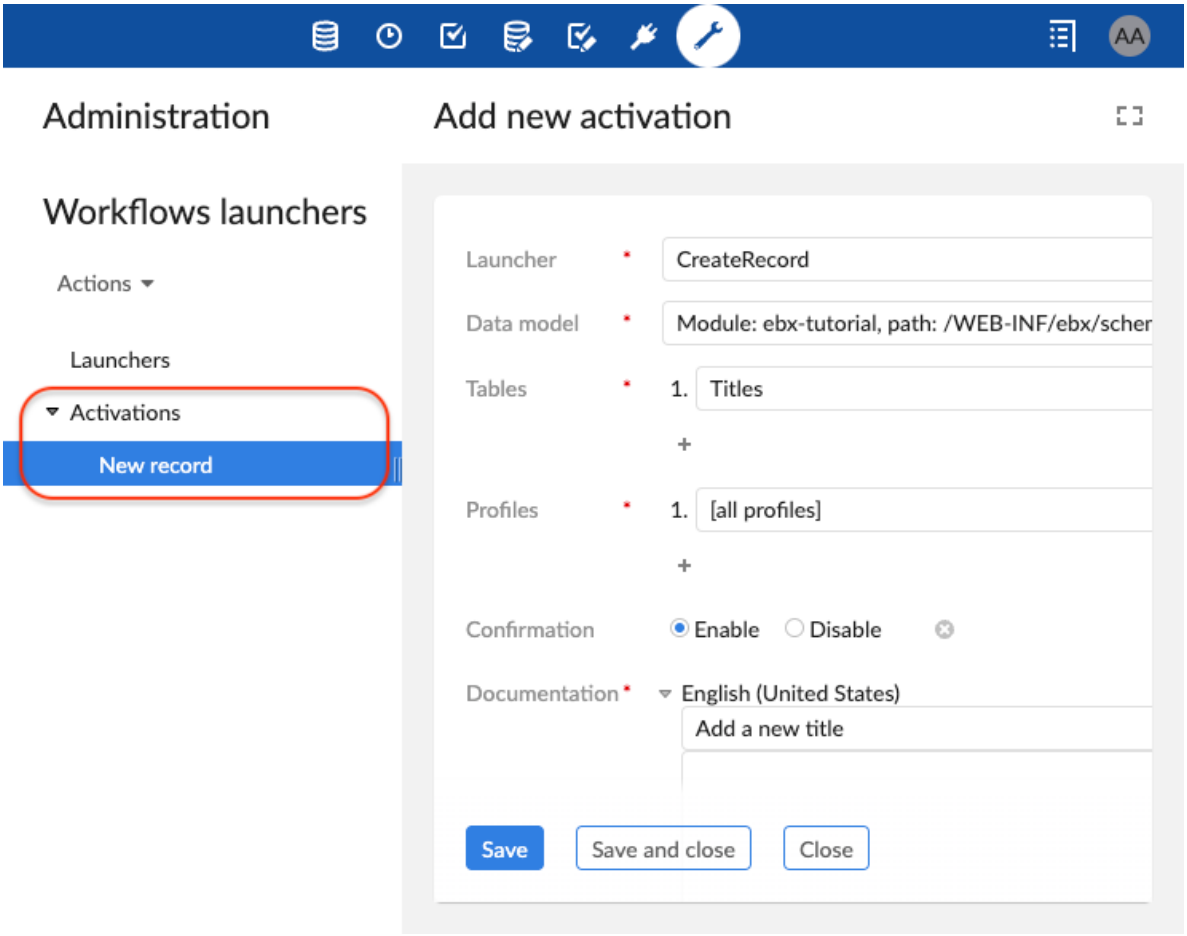


114.4 Activating workflow launcher

An activation is the entity which controls the availability of a workflow launcher on a table or record form. In order to display a workflow launcher in a toolbar of a particular table, first create a launcher [p 693], then navigate to the administration area, select the workflow management section, select the

workflow launchers dataset, select the activations table, and then add a new record. The second step, is to setup the following fields of the record:

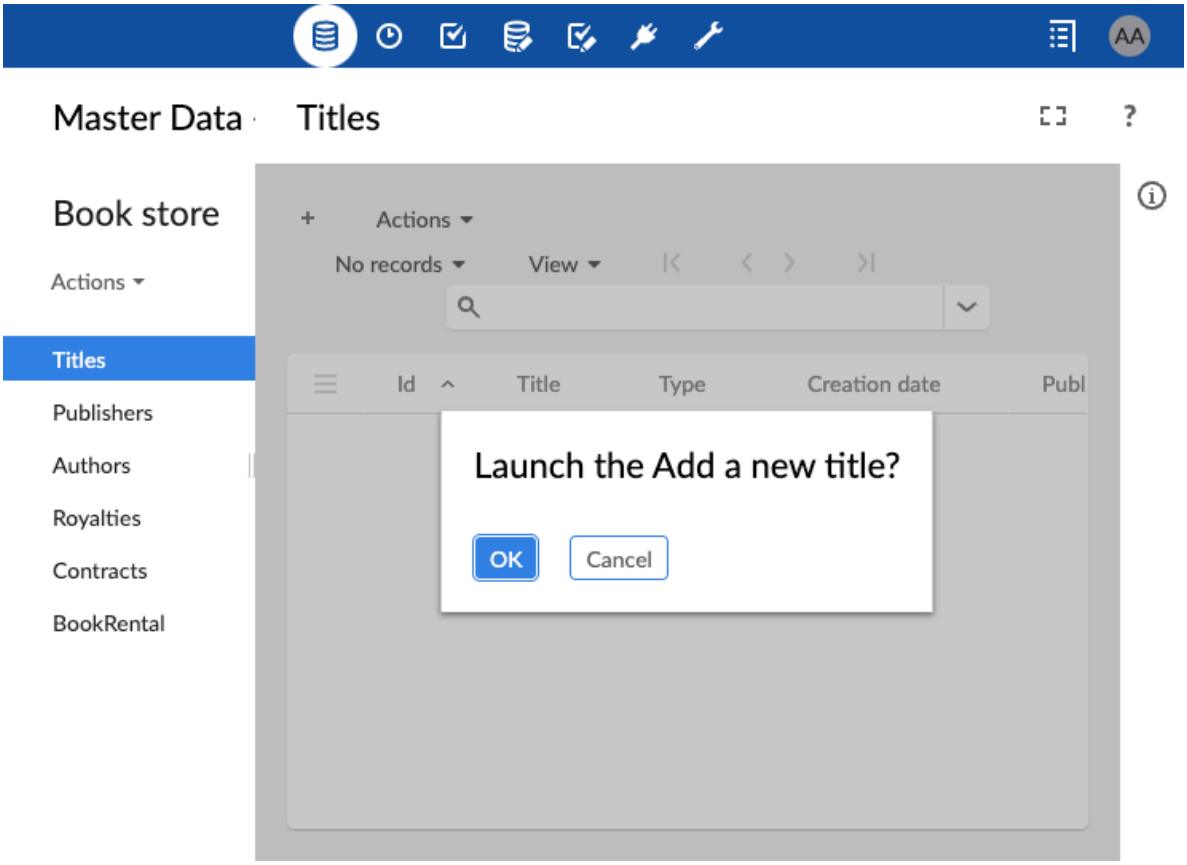
Field name	Description
Launcher	A unique identifier of a launcher which is associated to a publication name of the target workflow publication [p 694]. This field allows to select the launcher which will be displayed in the data section.
Data model	The schema reference of a published Data Model. Only schemas that are published and used in Datasets are available.
Tables	The identifier of a table which will display the workflow launcher. It is possible to select one or many tables. This selector displays all tables which are contained in the field **Data Model** [p 696] .
Profiles	List of profiles which are allowed to see and launch the workflow launcher in the previously selected tables.
Confirmation	This field allows displaying or not a dialog box to confirm launching of a data workflow (see screenshot about launch confirmation message [p 698]. By default this feature is deactivated, in order to display the dialog box, the value "Enabled" should be checked (see screenshot below).
Documentation	A documentation is composed of a label and a description. The default value of the label and description are inherited from the documentation field of the launcher. This field is used to display the title of the button of the user service in the toolbar. The description is displayed when the user hovers over that button. The documentation of the activation has a higher priority over the one of the launcher. Note that, if the description of the documentation is left empty then the following one is displayed "This user service will launch a data workflow". In the case of the custom toolbar, the value of this field is also used for **action button** and **action menu item** if the documentation field of the **action button** or **action menu item** is left empty.



114.5 **Launching a workflow**

Once a launcher is created [p 693] and activated [p 695] the corresponding workflow launcher becomes available on the toolbar's action menu. To launch the workflow click on the button of the workflow launchers(see screenshot about default display of a workflow launcher [p 691]).

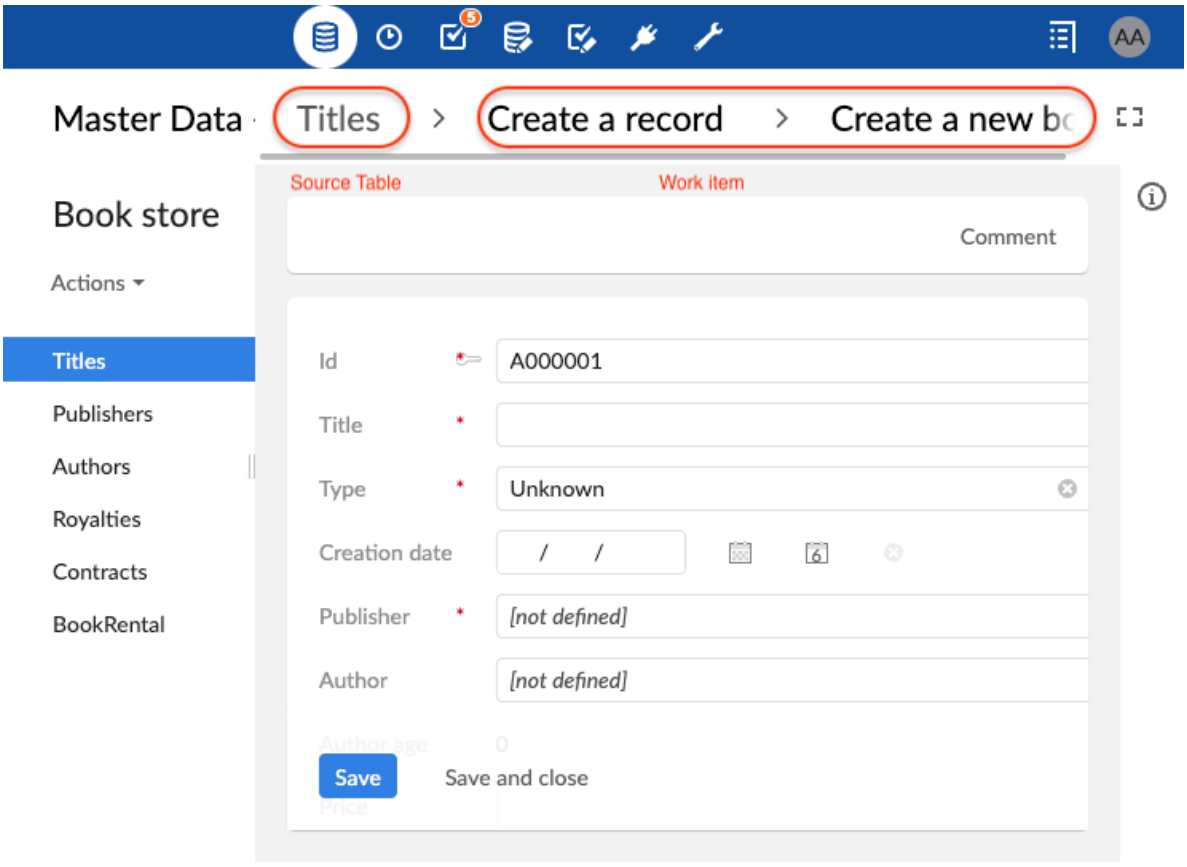
If the option **Confirmation** **[p 696]** is enabled, then a dialog box displays to ask the user the confirmation before launching the selected data workflow (see below screenshot about launch confirmation message [p 698]).



Otherwise, the dialog box does not display and different outcomes are possible:

- If the option **automatically open the first step** **[p 214]** is activated and the user has the rights to execute the first work item [p 201] of the workflow, then the workflow launches and the first work item displays directly in the workspace of the Data section. While displaying the work item in the workspace, the origin data selection (source table) still displayed on the breadcrumb which allows to the user to maintain an overall contextual awareness (see screenshot about displaying a work item on the data section [p 699]).
- If the option **automatically open the first step** **[p 214]** is deactivated and the user has the rights to launch the workflow [p 219], then the workflow is launched and added to the data workflow inbox [p 227]. An information message informs the user that the workflow has been launched successfully and that it is necessary to go to the inbox of the data workflow section [p 227] to display and execute the first work item.
- If the workflow requires record selection (see note in the section workflow launcher in data area [p 692]), the workflow is launched and displayed if and only if a record is selected otherwise an error message is displayed informing the user that a record should be selected. A record can be selected manually from a table view or automatically when displaying a record form.

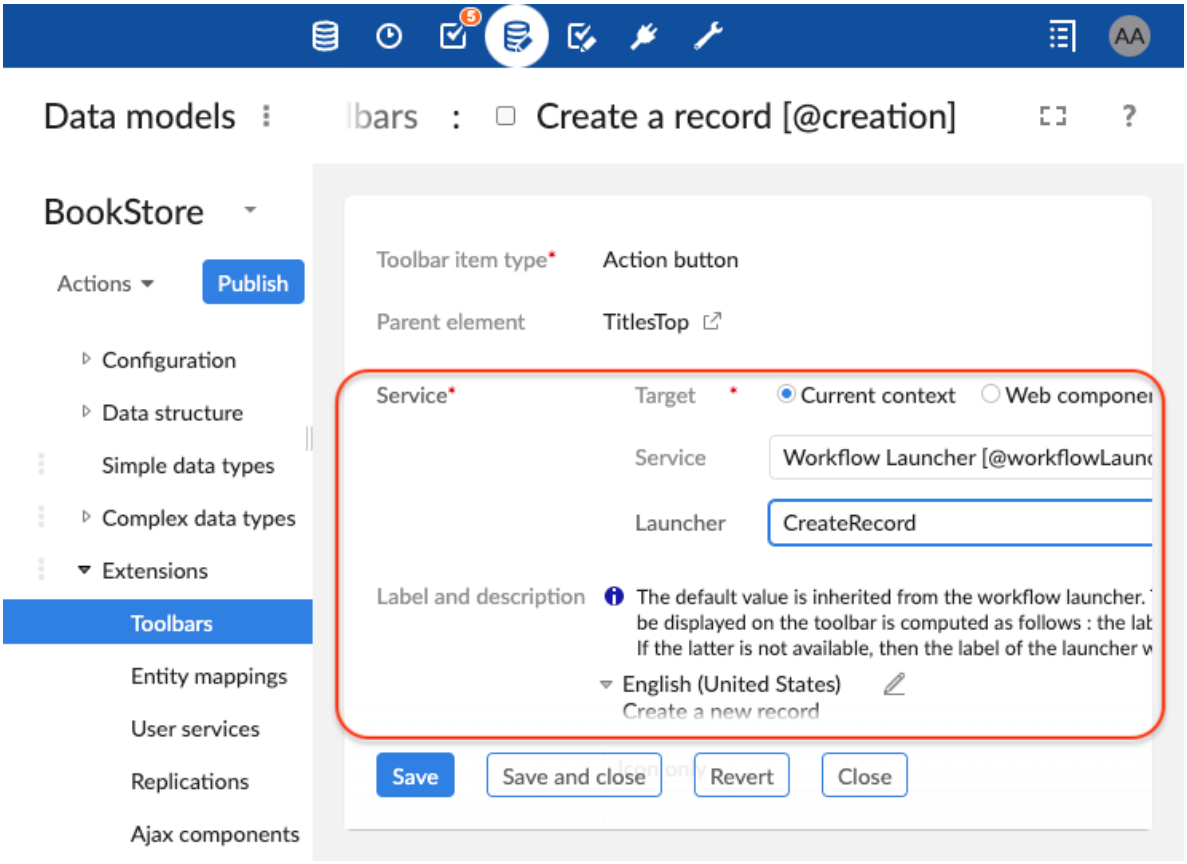
- If more than one record are selected, then a warning message notifies that only one record should be selected.



114.6 Adding a workflow launcher to the custom toolbar at table top

The workflow launchers can be made available not only on default toolbar but also on custom toolbar [p 86]. A workflow launcher can be added on a custom toolbar as an action button [p 87] or as an action menu item [p 90] in a custom menu group [p 89]. Adding an action button or an action menu item to launch a workflow on a custom toolbar follows a similar procedure as for common user services: first, set the **target** field to **current context** otherwise the user service **workflow launcher** will not be available, then select the user service **workflow launcher**. When the service workflow launcher is selected, a

new field **Launcher**, exclusive for workflow launcher user service, appears below the service input field (see below screenshot about adding a workflow launcher on custom toolbar [p 700]).



The field **Launcher** displays the list of launchers that have already been created in the table "Activation/workflow management/workflow launcher/Launchers" (see section about creating and configuring launchers [p 693]) . Therefore, before adding a workflow launcher in a custom toolbar the launcher should be created and configured for this workflow launcher.

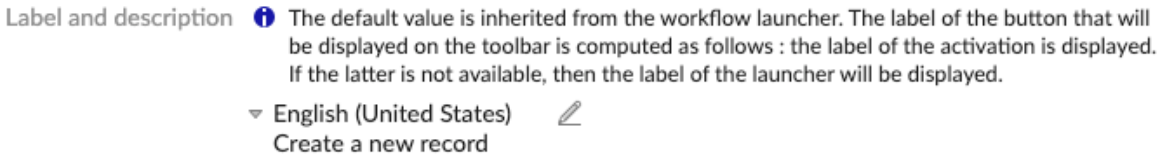
As for default toolbar, in order to show the workflow launcher on a custom toolbar its launcher should be activated in "Activation/workflow management/workflow launcher/activations". A quick link to access the activations table of the launchers is displayed on the right of the field **Launcher** (see the below screenshot about the service "workflow launcher" on custom toolbar [p 700])



The field Label and description displays the label and description inherited from the launcher (see screenshot about the label of the "workflow launcher" service in custom toolbar configuration form [p 701]) , then if the launcher field changes, the label and description should update automatically. This field can be overridden to customize the label and description of the action button on the toolbar. The label and description that will be displayed on the toolbar, are computed in the following order of priority: the custom value of the field Label and description of the action button is used, if this field is left empty or contains the default label of the launcher, then the label and description of the launcher

activation are displayed. If this last one is left empty, then the following description is used "This user service will launch a data workflow.".

> **Note**
>
> The label and description which are displayed on the toolbar in the data section and the label of the action button in the toolbar tree in the DMA could be different. This is the case when the field "Label and description" of the "action button" is left empty (neither default nor custom label is defined). The label of the "action button" in the toolbar tree in the DMA is inherited from the label of the launcher, however, the label and description on the toolbar in the data section are set to the label and description of the activation of the launcher.



Adding a workflow launcher to a toolbar in other locations

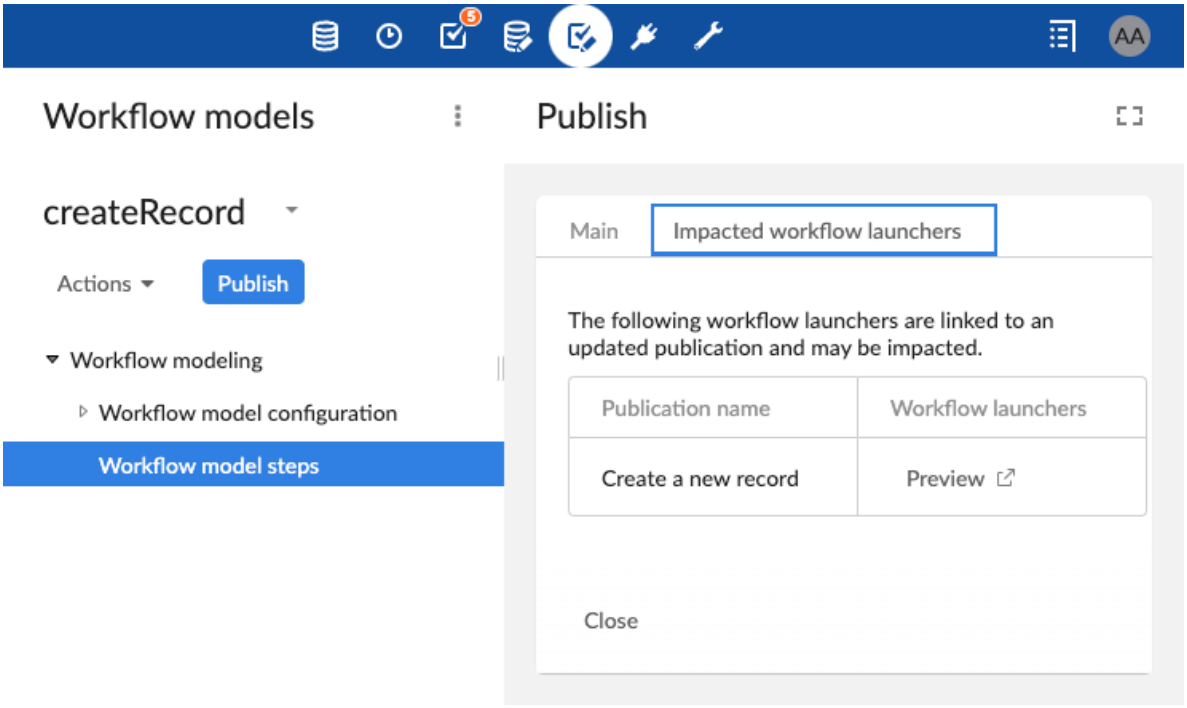
The workflow launcher service is only available on the toolbar that displays in the following locations: table view top, table rows, record form view top, or hierarchy view top. Until now, this service is not available on the toolbar that displays on top of the association table or in the association table rows.

114.7 Access a launcher after workflow model modification

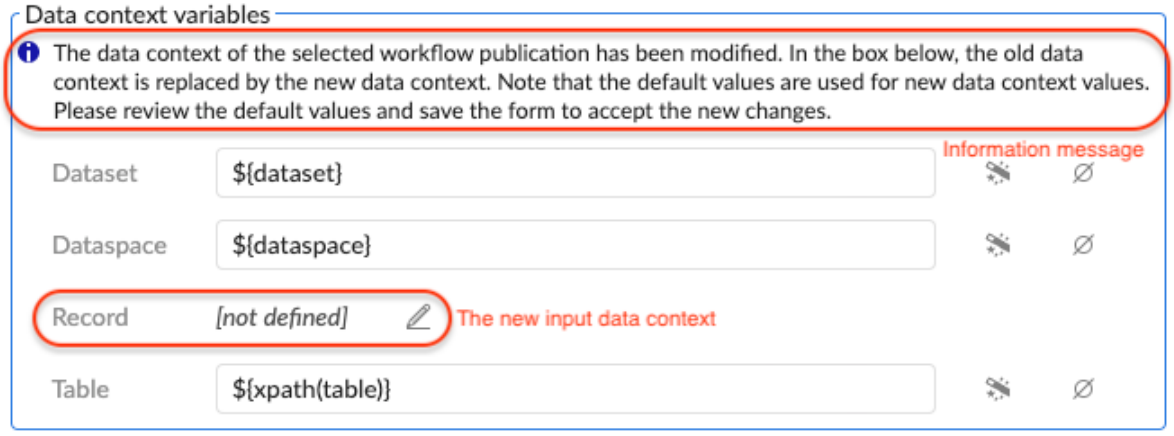
After creating a launcher [p 693] which points to a workflow publication [p 694], the workflow model of this publication may be changed and republished. For instance, the data context of the workflow model may be changed: one or more variable data context can be added; one or more variable data context can be removed; the name of a variable can be changed. In this particular case, the launchers which points to this workflow publication should be reviewed and validated (saved) by the user because the data context used by these launchers is no more valid and it should be updated to match the one of the up-to-date workflow publication.

After republishing a workflow model, the user is notified if any of the workflow launcher points to the current workflow publication and if it should be reviewed. For that purpose, after publishing a workflow model, a preview button is displays allowing a quick access to the launchers in question

(see below screenshot). Note that if the user has no rights to access the workflow launchers list, then the names of those launchers are displayed.



When the user access to one of these launchers, via the preview button in the workflow model section or via the Administration section, the persisted data context is replaced with the up-to-date data context of the workflow publication, however the new data context is not yet persisted for the current launcher. In order to update the data context of the launcher, the user should first review the values and then save to accept the new changes (see screenshot about accessing a launcher after modification of the data context of the corresponding workflow publication [p 702]).



CHAPTER 115

Users and roles directory

This chapter contains the following topics:

1. Overview
2. Directory concepts overview
3. Main directory
4. Synchronized directory

115.1 Overview

TIBCO EBX® uses a directory system to manage user authentication and role definitions. A main directory is required and is the default source for authentication and authorization.

When Single Sign-On (SSO) is enabled, EBX® can use a synchronized directory in addition to the main directory. The synchronized directory enables management of external users authenticated via an identity provider. With this setup:

- Use the main directory for technical users.
- Use the synchronized directory for users managed by the external system.

The following list outlines the general procedure for configuring a directory to handle user authentication and authorization:

- Choose the main directory implementation: Default, built-in LDAP, or Custom directory.
- If needed, enable SSO using one protocol: OIDC or SAML2.
- If needed, enable the synchronized directory using one of the built-in implementations: LDAP, SCIM or On Login.
- Configure the selected directories and protocols.

See also *Configuring TIBCO EBX® directory properties* *[p 575]*

> **Note**
>
> Do not use the Default directory in production. Instead, use your own custom Directory with a more secure password system or a built-in directory option to protect sensitive information.

If both main and synchronized directories are used, usernames must be unique across directories. If a username exists in both, the system merges roles from both directories. However, the user's

information (e.g., name, email, etc.) is taken only from the synchronized directory. This can lead to inconsistencies or unintended overrides, so unique usernames across directories are strongly advised.

115.2 Directory concepts overview

In TIBCO EBX®, a user can be a member of one or more roles, and multiple users can share the same role. Additionally, a role can include other roles. The documentation uses the generic term *profile* to describe either a user or a role. Information related to profiles is primarily defined in the directory.

In addition to roles you define in a directory, TIBCO EBX® provides the following *built-in roles*:

Role	Definition
Profile.ADMINISTRATOR	Grants full access for administrative tasks.
Profile.READ_ONLY	Allows users to view content only. No repository modifications are allowed.
Profile.OWNER	A dynamic role that is only active when the user is (or is included in) the profile defined as the owner of the current element.
Profile.EVERYONE	All users are included in this role.

Attention

Associations between users and the built-in roles *OWNER* and *EVERYONE* are managed automatically by TIBCO EBX®, and must not be modified through the directory.

User permissions are managed separately from the directory. See Permissions [p 487] for more information.

See also

profile [p 33]

role [p 34]

user [p 33]

administrator [p 34]

user and roles directory [p 34]

Policy

The *Directory > Policy* table allows you to configure whether users can manage specific aspects of their own profiles. For example, policies can specify whether:

- users can update their own account information.
- password reminder options are also available.

115.3 Main directory

The main directory is required and handles login authorization using a username and password. EBX® supports the following directory types: default directory [p 705], LDAP directory [p 706], or a custom directory [p 706] implementation.

Default directory

The default directory and related tables is located in the *Administration > Users, roles and preferences > Directory* dataset. This dataset includes tables to create and store: user policies, mailing lists, users and roles. You cannot delete or duplicate the default directory.

Depending on the defined policy, users can modify information related to their own accounts, regardless of the permissions defined on the directory dataset.

> **Note**
>
> If a role inclusion cycle is detected, the role inclusion is ignored during permission resolution. To detect such cycles, refresh and check the directory validation report.

> **Note**
>
> The **Users roles** and **Roles inclusions and salutations** tables are hidden by default [p 968].

Password recovery procedure

In the default directory, passwords are encrypted (using SHA256-like algorithm by default), and stored in this state. As a result, lost passwords cannot be retrieved. A new password must be generated and sent to the user.

There are two options for this procedure:

1. A notification email is sent to the administrator, the administrator manually changes the password and sends the new password to the user. This is the default option.
2. A procedure automatically generates a new password and sends it to the user. To activate this option, set the following property in the TIBCO EBX® main configuration file [p 571]: `ebx.password.remind.auto=true`.

> **Note**
>
> For security reasons, the above password recovery procedure is not available for administrator profiles. Use the administrator recovery procedure instead.

Administrator recovery procedure

If all administrator credentials are lost, use the following special recovery process to use a directory class to redefine a temporary administrative account with `admin`/`admin` as the login and password:

1. Specify the following property in the TIBCO EBX® main configuration file [p 571]:

   ```
   ebx.directory.factory=
   com.orchestranetworks.service.directory.DirectoryDefaultRecoverFactory
   ```

2. Start TIBCO EBX® and wait until the procedure completes.
3. Reset the '`ebx.directory.factory`' property.

4. Restart TIBCO EBX® and connect using the 'admin' account.

> **Note**
>
> While the '`ebx.directory.factory`' property is set for the recovery procedure, all other user authentications are disabled.

LDAP directory

EBX® supports LDAP (Lightweight Directory Access Protocol) as either a main directory or a synchronized directory. See Synchronized LDAP directory [p 706] for instructions on using it as a synchronized directory. To use LDAP as a main directory: Set the directory factory to the built-in LDAP directory factory in Configuring the user and roles directory [p 575].

Custom directory

A custom directory is an alternative to using the EBX® built-in directories. You can integrate a specific company directory contained in a relational database, or a specific directory data model instantiated in TIBCO EBX®. You can also set the default login page to a specific company page. See the following links for the corresponding documentation resources.

See also

Configuring TIBCO EBX® Container Edition environment variables [p 640]

`DirectoryFactory`[API]

`Directory.getUserAuthenticationURI`[API]

115.4 Synchronized directory

The Synchronized directory is represented by the dataset **ebx-synchronized-directory**, available in the **Administration** area after configuration. It is an integrated directory used to authenticate and authorize external users. EBX® manages permissions using internal roles only. External roles must be mapped to internal roles.

A synchronized directory should be used in combination with SSO in EBX®:

- The directory is used to manage users and roles with automatic provisioning to ensure up-to-date access control.
- The SSO handles user authentication.

This approach provides centralized user management and ensures consistent access control across systems.

The synchronized directory and its role mapper are configured through the main EBX® configuration file. See Configuring the synchronized directory [p 576] for instructions.

LDAP directory

TIBCO EBX® built-in support for LDAP (Lightweight Directory Access Protocol) was designed as main and synchronized directory implementations, which allows you to integrate an existing LDAP directory with TIBCO EBX®. This ensures a streamlined and efficient directory management experience through secure and consistent access control that is tailored to your organization's LDAP infrastructure needs.

Role synchronization is done by scheduling the task: `com.orchestranetworks.service.directory.scim.SchedulerLdapSynchronization`

LDAP Configuration

The following four types of configuration parameters are available:

- Connection parameters are needed to establish and handle a connection to the LDAP server. They are also used to tune the connection for optimal performance and security.

  For LDAPS uses, a root certificate must be imported if it is not available in the JDK TrustStore (see Installing a Root Certificate in the TrustStore for more information).
- Mapping parameters are used to map certain built-in roles and users to LDAP directory attributes.
- Display of users and roles using expressions derived from LDAP attributes.
- Search request templates generate LDAP requests, allowing precise and efficient information retrieval from the LDAP directory.

Search request templates

An LDAP search query consists of the following three key elements:

- Base DN (Distinguished Name): sets the initial point in the directory tree.
- Search filter: defines attributes and values to locate.
- Search scope: dictates the search depth within the directory tree (from the base to several levels deep).

It is important to note that queries are paginated to enhance efficiency and manageability. Additionally, each request template can use caching to optimize performance. Use the available parameters to fine-tune page retrieval and cache implementation. Caching is recommended, even if for short durations with a small size. It can significantly improve overall system performance by reducing redundant data fetching.

Search templates support placeholders that adapt to each specific type of query. There are four mandatory templates that:

- Find an individual user.
- Identify a user's groups.
- Find users in a group.
- Retrieve all groups.

Additionally, there is an optional template to find all users. While not required, it is crucial for broader user accessibility. If it is not configured when assigning permissions, selections are limited to groups. Individual users cannot be selected.

See also

Configuring TIBCO EBX® main configuration properties *[p 577]*

Configuring TIBCO EBX® Container Edition environment variables *[p 642]*

Limitations

When using LDAP as a synchronized directory in EBX®, the following restrictions apply:

- Cache mechanisms are not compatible with LDAP servers that rely on session identifiers stored in cookies for pagination. In such cases, the request cache system must be disabled by setting the cache maximum size to 0.
- Only one LDAP server can be configured at a time. Multiple simultaneous connections to LDAP servers are not supported within the same instance.
- There is no support for role inclusion or the addition of specific roles on the EBX® side. Consequently, the structure and nested relationships of groups within LDAP are not considered in the calculation of permissions. This limitation highlights a straightforward, group-based access control approach without the complexities of LDAP's hierarchical group dynamics.
- Only simple and anonymous authentication methods are allowed. More complex or secure authentication protocols used within LDAP are not supported.
- Roles and usernames display is not locale dependent and is limited to one format defined by display expressions in the configuration file. Locale is only considered when the display expressions can not be applied.

SCIM directory

SCIM (System for Cross-domain Identity Management) is a standard protocol designed to automate the exchange of user identity information between IT systems. It simplifies user provisioning and management across multiple platforms by using a common schema and RESTful API.

The schema version used is: `urn:ietf:params:scim:schemas:core:2.0`.

Role synchronization is managed by the scheduled task: `com.orchestranetworks.service.directory.scim.SchedulerScimSynchronization`

Configuration

A SCIM server should be activated on the IdP (Identity Provider) side.

See also

Configuring TIBCO EBX® main configuration properties *[p 578]*

Configuring TIBCO EBX® Container Edition environment variables *[p 647]*

On Login directory

The On Login directory is a persistent directory that synchronizes users and roles on login, based on information from SSO protocols. As a consequent, the list of users is not complete. Users that have never logged in will are not included.

Roles Mapping

A role mapper can be configured in the main EBX® configuration file to map between an external user’s roles and his internal roles.

See also *Configuring the synchronized directory* *[p 576]*

Migration

To ensure the migration of LDAP and SCIM directories from a version prior to 6.2.2, enable roles import on the scheduled task options of each directory. This ensures the consistency of already persisted permissions with external roles.

CHAPTER **116**

Data model administration

This chapter contains the following topics:

1. Administrating publications and versions
2. Migration of previous data models in the repository
3. Schema evolutions

116.1 Administrating publications and versions

Technical data related to data model publications and versions can be accessed in the *Administration* section by an administrator.

Data Modeling contains the following two tables:

- *Publications*. Stores the publications available in the repository.
- *Versions*. Stores the versions of the data models available in the repository.

These tables are read-only but it is however possible to delete manually a publication or a version.

Important: If a publication or a version is deleted, then the content of associated datasets will become unavailable. So this technical data must be deleted with caution.

It is possible to spread this technical data to other TIBCO EBX® repositories exporting an archive from an EBX® repository and importing it to another one. It may be useful for propagating the evolutions of data models to other repositories.

116.2 Migration of previous data models in the repository

In versions before *5.2.0*, published data models not depending on a module were generated in the file system directory `${ebx.repository.directory}/schemas/`, with the name of the data model (*product.xsd* for example if the data model is named *Product*). Since the *5.2.0* version, this kind of data model is now fully managed within EBX® through *Publications*. That is, republishing an existing data model migrates it as a *Publication* and redirects linked datasets to the new embedded data model. The previous XML Schema Document located in `${ebx.repository.directory}/schemas/` is renamed and suffixed with *toDelete*, meaning that the document is no longer used and can be safely deleted.

116.3 Schema evolutions

It is crucial to evaluate the impact of data model changes on the administration side. The following points are to be considered:

Impacts on data persistence

Administration tasks can be related to the database cleanup after a modification of the models. The following link describes how the evolutions of data models are managed at the persistence level: Purging master tables in the database [p 713].

Impacts on side features

Some components rely heavily on the data models and can be impacted by their evolutions. Some examples are: the user interface, the WSDL documents, existing archives, etc.

The 'Administration' section offers the possibility to manage some of these components (such as the views), whereas other components fall out of the administrator's scope, such as archives, WSDL files, etc.

CHAPTER 117

Database mapping administration

This chapter contains the following topics:

1. Overview
2. Renaming columns in the database
3. Purging columns in the database
4. Renaming master tables in the database
5. Renaming auxiliary tables in the database
6. Purging master tables in the database

117.1 Overview

Information and services relative to database mapping can be found in the *Administration* area.

See also

Mapped modes *[p 459]*

`DatabaseMapping`[API]

117.2 Renaming columns in the database

This feature is available on the 'Columns' table records, under the 'Actions' menu. It allows renaming a column in the database.

The administrator can specify the name of each column of the data model in the database for mapped modes.

Once the service is selected on a record, a summary screen displays information regarding the selected column and the administrator is prompted to enter a new name for the column in the database.

> **Note**
>
> It is required that the new identifier begins with a letter.
>
> Besides, the new name must be a valid column identifier, which depends on the naming rules of the underlying RDBMS.

See also `DatabaseMapping`[API]

117.3 Purging columns in the database

This feature is available on the 'Columns' table records, under the 'Actions' menu. It allows purging columns in mapped structures.

A column can be purged if it has been disabled for mapped modes.

A column is disabled for mapped modes when:

- the corresponding field has been removed from the data model, or
- the corresponding field has been changed in the data model, in a way that is not compatible (for example: its data type has been modified), or
- the defined mapped modes have been disabled locally on the corresponding fields, using the elements `osd:history` and `osd:replication`.

See also

Disabling history on a specific field or group *[p 462]*

Disabling replication on a specific field or group *[p 471]*

Note that this behavior will change for aggregated lists:

- when deactivating a complex aggregated list, its inner fields will still be in the `LIVING` state, whereas the list node is disabled. As lists are considered as auxiliary tables in the mapping system, this information can be checked in the 'Tables' table,
- on the other hand, when the deactivation is just for inner nodes of the list, then the list will remain `LIVING`, while its children will be `DISABLED IN MODEL`.

A column can be purged only if its own state is `DISABLED IN MODEL`, or if it is an inner field of a `DISABLED IN MODEL` list.

117.4 Renaming master tables in the database

This feature allows renaming master tables for history tables in the database. It is not available for replicated tables since their names are specified in the data model.

Both features are available on the 'Tables' table records, under the 'Actions' menu.

Master tables are database tables used for persisting the tables of the data model.

The administrator can specify in the database the name of each master table corresponding to a table of the data model.

Once the service is selected on a record, a summary screen displays information regarding the selected master table and the administrator is prompted to enter a new name for the master table in the database.

> **Note**
>
> It is required that the new identifier begins with a letter and with the repository prefix.
>
> For history tables, it is also required for the repository prefix to be followed by the history tables prefix.
>
> Besides, the new name must be a valid table identifier, which depends on the naming rules of the underlying RDBMS.

117.5 Renaming auxiliary tables in the database

This feature allows renaming history auxiliary tables in the database. This feature is not available for replicated tables since their names are specified in the data model.

This feature is available on the 'Tables' table records, under the 'Actions' menu.

Auxiliary tables are database tables used for persisting aggregated lists.

The administrator can specify in the database the name of each auxiliary table corresponding to an aggregated list of the data model.

Once the service is selected on a record, a summary screen displays information regarding the selected auxiliary table and the administrator is prompted to enter a new name for the auxiliary table in the database.

> **Note**
>
> It is required for the new identifier to begin with a letter.
>
> It is required for the new identifier to begin with the repository prefix.
>
> It is also required for the repository prefix to be followed by the history tables prefix.
>
> Besides, the new name must be a valid table identifier, which depends on the naming rules of the underlying RDBMS.

117.6 Purging master tables in the database

This feature allows purging history in the database if it is no longer used.

It is available on the 'Tables' table records, under the 'Actions' menu, and is only available for master tables. This feature only applies to master tables. When a master table is purged, all its auxiliary tables are purged as well.

A mapped table can be purged in the database only if it has been disabled for the corresponding mapped mode.

To disable the mapped mode for a table, follow the procedure hereafter.

- Deactivate historization of the table in the data model, or
- Remove the table from the data model

CHAPTER 118

External services configuration

This chapter contains the following topics:

1. Configuration table
2. Authentication types
3. Keystore file

118.1 Configuration table

Configure external services in the *Administration* > *Integration* > *External services* > *Configuration* table. Enter the following properties:

Property	Definition
Identifier	Unique identifier of the service configuration.
URL	Specifies the base URL of the server to be called. Use HTTPS to encrypt the communication between the client and the server. The URL has the following format: `http[s]://<host>[:<port>]/{path}{?<queryParameters>}` Where: • `host` and `port`: define the server address where the API resides. • `path`: the path of the specific resource within the API. • `queryParameters`: optional parameters used to filter or refine the resource.
Usage type	Defines the usage type of the external service: Assistant AI and/or the REST service.
Authentication type	Defines the user authentication when making API requests. Use the following types to access the API and its resources: basic, custom, EBX and none.

118.2 Authentication types

Authentication ensures that only authorized users can access the service and its resources. The following authentication types are supported:

Type	Description
Basic	A simple and fast method of HTTP authentication that requires a valid username and password. Credentials are encoded in base 64 and included in the HTTP request header. Passwords are securely stored in the EBX® repository with strong encryption with strong encryption. For details see Keystore file [p 716].
EBX	Enables access to the EBX Built-in RESTful [p 1125] service. A valid username and password must be provided. Passwords are securely stored in the EBX® with strong encryption. For details see Keystore file [p 716].
Custom	Enables custom authentication such as API keys, tokens, or OAuth by providing valid key names and key values. The values of the keys can be secret or not. All the keys, secret or not, must have a different name otherwise a validation error is displayed. Secret keys are securely stored in the EBX® repository with strong encryption. For details see Keystore file [p 716].
None	No authentication is needed.

118.3 Keystore file

Introduction

EBX® uses a keystore file to store secret keys used for encrypting or signing data. The location of this file and related details are specified by a *keystore sensitive property file*.

Keystore sensitive property file

Sensitive properties specify the location of the keystore file and other related information, such as the secret key aliases and passwords.

Store sensitive properties in a property file that is separate from the `ebx.properties` file. By default, this file is named `ebx-keystore-sensitive.properties` and is stored in the same directory as the EBX® repository. You can change the location by setting property `ebx.keystore.sensitive.properties.path` in file `ebx.properties`.

This property file has the following format:

```
## This file contains sensitive properties of the EBX application keystore.
## Access to this file should be restricted to the operating system user that is running EBX.
##
## The keystore type.
ebx.keystore.type=PKCS12
##
## The keystore path.
## If relative, it's absolute path is resolved against this file's folder.
```

```
## This property should be removed if the keystore type is not file based.
ebx.keystore.path=ebx-keystore.p12
##
## The keystore password.
ebx.keystore.password.obf=XjJZFfsUXfrzq66GJxfBIa/ulHdPqaeC6/MMMHLyA11XvQZ6/rFqVbmjDZgDL5GPRcDc1+OQQpw6Fg==
##
## The alias of the secret key used to encrypt password type values.
ebx.keystore.key.passwordType.alias=ebx.passwordType
##
## The password of the secret key used to encrypt password type values.
## For PKCS12 keystores, should be the same as the keystore password.
ebx.keystore.key.passwordType.password.obf=XjJZFfsUXfrzq66GJxfBIa/ulHdPqaeC6/MMMHLyA11XvQZ6/
rFqVbmjDZgDL5GPRcDc1+OQQpw6Fg==
##
## The alias of the secret key used to sign web tokens.
ebx.keystore.key.webToken.alias=ebx.webToken
##
## The password of the secret key used to sign web tokens.
## For PKCS12 keystores, should be the same as the keystore password.
ebx.keystore.key.webToken.password.obf=XjJZFfsUXfrzq66GJxfBIa/ulHdPqaeC6/MMMHLyA11XvQZ6/
rFqVbmjDZgDL5GPRcDc1+OQQpw6Fg==
```

Automatic installation

On first startup, EBX® creates the `ebx-keystore.p12` and `ebx-keystore-sensitive.properties` files in the EBX® repository folder. The secret keys are automatically generated and specific to an installation.

Staging

By default, EBX® staging servers have different secret keys. Data encrypted by one staging server cannot be decrypted by another. To avoid this issue and align keys across all EBX® staging servers:

- copy the same keystore file (by default `ebx-keystore.p12`), and
- copy the keystore sensitive property file (by default `ebx-keystore-sensitive.properties`) to all EBX® staging server installations.

CHAPTER 119

Workflow management

This chapter contains the following topics:

1. Workflows
2. Interactions
3. Workflow history

119.1 Workflows

To define general parameters for the execution of data workflows, the management of workflow publications, or to oversee data workflows in progress, navigate to the 'Administration' area. Click on the down arrow in the navigation pane and select *Workflow management* > *Workflows*.

> **Note**
>
> In cases where unexpected inconsistencies arise in the workflow execution technical tables, data workflows may encounter errors. It may then be necessary to run the operation 'Clean up inconsistencies in workflow execution tables' from the 'Actions' menu in the navigation pane under *Administration* > *Workflow Management* > *Workflows*.

Execution of workflows

Various tables can be used to manage the data workflows that are currently in progress. These tables are accessible in *Workflow management* > *Workflows* in the navigation pane.

See also *Administration of data workflows [p 239]*

Workflows table

The 'Workflows' table contains instances of all data workflows in the repository, including those invoked as sub-workflows. A data workflow is a particular execution instance of a workflow model publication. This table provides access to the data context variables for all data workflows. It can be used to access the status of advancement of the data workflow in terms of current variable values, and in case of a data workflow suspension, to modify the variable values.

From the 'Actions' menu of the 'Workflows' table, it is possible to clear the completed data workflows that are older than a given date, by selecting the 'Clean from a date' service. This service automatically ignores the active data workflows.

Tokens table

The 'Tokens' table allows managing the progress of data workflows. Each token marks the current step being executed in a running data workflow, as well as the current state of the data workflow.

See also *token [p 41]*

Work items table

The 'Work items' table contains all the work items associated with user tasks that currently exist. If necessary, you can manually allocate a work item to a user from this table in the case of a blockage in a data workflow. It is preferable, however, to use the buttons in the workspace of the 'Data workflows' area whenever possible to allocate, reallocate, and deallocate work items.

See also *work item [p 41]*

Waiting workflows table

The 'Waiting workflows' table contains all the workflows waiting for an event. If needed, a service is available to clean this table: this service deletes all lines associated with a deleted workflow.

See also *wait task [p 40]*

Comment table

The 'Comments' table contains the user's comments for main workflows and their sub-workflows.

Workflow publications

The 'Workflow publications' table is a technical table that contains all the workflow model publications of the repository. This table associates published workflow models with their snapshots. It is not recommended to directly modify this table, but rather to use the actions available in the workflow modeling area to make changes to publications.

Configuration

Email configuration

In order for email notifications to be sent during the data workflow execution, the following settings must be configured under 'Email configuration':

- The URL definition field is used to build links and value mail variables in the workflow.
- The 'From email' field must be completed with the email address that will be used to send email notifications.

Interface customization

Modeling default values

The default value for some properties can be customized in this section.

The administrator has the possibility to define the default values to be used when a new workflow model or workflow step is created in the 'Workflow Modeling' section.

Work items views

Specific columns are available in the inbox and in the monitoring work items tables, in the 'Data workflows' section.

10 specific columns are available. For each specific column, a customized label can be defined.

Priorities configuration

The property 'Default priority' defines how data workflows and their work items across the repository display if they have no priority level. For example, if this property is set to the value 'Normal', any workflow and work item with no priority will appear to have the 'Normal' priority.

The 'priorities' table defines all priority levels available to data workflows in the repository. As many integer priority levels as needed can be added, along with their labels, which will appear when users hover over the priority icon in the work item tables. The icons that correspond to each priority level can also be selected, either from the set provided by TIBCO EBX®, or by specifying a URL to an icon image file.

Temporal tasks

Under 'Temporal tasks', the polling interval for time-dependent tasks in the workflow can be set, such as deadlines and reminders. If no interval value is set, the 'in progress' steps are checked every hour.

Workflow inbox counter configuration

The workflow inbox counter is refreshed asynchronously, even if the end-user does not launch any action. To adjust it, two parameters need to be set:

Cache expiry (seconds)	Expiration time (in seconds) before a new update of the inbox cache. Please note that this parameter can impact the CPU load and performance since the computation time can be costly for a repository with many work items. If no value is defined, the default value is 600.
User interface refresh periodicity (seconds)	Refresh time (in seconds) between two updates of the inbox counter in the user interface. Please note that this refresh concerns all inbox counters in the user interface: inbox counters of the custom perspective, header inbox counter and Data Workflows inbox counter for the advanced perspective. If no value is defined, default value is 5. If the value is zero (or negative), the refresh is disabled. Also, the modification will only be effective after a logout/login from the user.

Also, please note that some actions can force the inbox counter to refresh:

- access on **Data workflows**
- access on any subdivision of the **Data workflows section**
- accept or reject a work item
- launch a workflow

These parameters are accessible in *Workflow management* > *Workflows* > *Configuration* > *Temporal tasks* in the navigation pane.

119.2 Interactions

To manage workflow interactions, navigate to the Administration area. Click the down arrow in the navigation pane and select the entry *Workflow management* > *Interactions*.

An *interaction* is generated automatically for every work item that is created. It is a local data context of a work item and is accessible from an EBX® session. When a work item is executed, the user performs the assigned actions based upon its interaction, independently of the workflow engine. User tasks define mappings for their input and output parameters to link interactions with the overall data contexts of data workflows.

Interactions can be useful for monitoring the current parameters of work items.

For example, an interaction can be updated manually by a trigger or a user service.

119.3 Workflow history

To view the history data workflow execution, browse the 'Administration' area. Click on the down arrow in the navigation pane and select *Workflow management* > *Workflow history*.

The 'Workflows' table contains all actions that have been performed during the execution of workflows.

This data can be viewed graphically or textually. It is especially useful to view the states of various objects related to workflows at a given moment. This includes actions on work items, variables in the data context, as well as tokens. In case of an error, a technical log is available.

Clean history

From the 'Actions' menu of the 'Workflows' table, the history of completed data workflows older than a given date can be cleared by selecting the 'Clear from a date' service.

Only the history of workflows that have been previously cleaned (e.g. their execution data deleted) is cleared. This service automatically ignores the history associated with existing workflows. It is necessary to clear data workflows before clearing the associated history, by using the dedicated service 'Clear from a date' from the 'Workflows' table. Also, a scheduled 'Clear from a date' can be used with the built-in scheduled task *SchedulerPurgeWorkflowMainHistory*.

Please note that only main processes are cleaned. In order to remove sub-processes and all related data, it will be necessary to run a 'standard EBX® purge'.

See also *How to clean workflow history [p 668]*

> **Note**
>
> An API is available to fetch the history of a workflow. Direct access to the underlying workflow history SQL tables is not supported. See **WorkflowEngine.getProcessInstanceHistory** `WorkflowEngine.getProcessInstanceHistory`[API].

CHAPTER 120

Task scheduler

This chapter contains the following topics:

1. Overview
2. Configuration from EBX®
3. Cron expression
4. Task definition
5. Task configuration

120.1 Overview

TIBCO EBX® offers the ability to schedule programmatic tasks.

> **Note**
>
> In order to avoid conflicts and deadlocks, tasks are scheduled in a single queue.

120.2 Configuration from EBX®

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the 'Administration' area.

- **Schedules**: defines scheduling using "cron expressions".
- **Tasks**: configures tasks, including parametrizing task instances and user profiles for their execution.
- **Scheduled tasks**: current schedule, including task scheduling activation/deactivation.
- **Execution reports**: reports of each scheduled task run that appear immediately after the task is triggered. The reports include actions to interrupt, pause, or resume running tasks, when made available by the task definition.

120.3 Cron expression

(An extract of the Quartz Scheduler documentation)

The task scheduler uses "cron expressions", which can create firing schedules such as: "At 8:00am every Monday through Friday" or "At 1:30am every last Friday of the month".

Format

A cron expression is a string composed of 6 or 7 fields separated by a white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

Field Name	Mandatory	Allowed Values	Allowed Special Characters
Seconds	Yes	0-59	, - * /
Minutes	Yes	0-59	, - * /
Hours	Yes	0-23	, - * /
Day of month	Yes	0-31	, - * ? / L W
Month	Yes	1-12 or JAN-DEC	, - * /
Day of week	Yes	1-7 or SUN-SAT	, - * ? / L #
Year	No	empty, 1970-2099	, - * /

A cron expression can be as simple as this: "**0 * * * * ?**",

or more complex, like this: "**0/5 14,18,3-39,52 * ? JAN,MAR,SEP MON-FRI 2002-2010**".

> **Note**
>
> The legal characters and the names of months and days of the week are not case sensitive. MON is the same as mon.

Special characters

A cron expression is a string composed of 6 or 7 fields separated by a white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

- ***** ("all values") - used to select all values within a field. For example, "*" in the Minutes field means "every minute".
- **?** ("no specific value") - useful when you need to specify something in one of the two fields in which the character is allowed, but not the other. For example, if I want my trigger to fire on a particular day of the month (say, the 10th), but don't care what day of the week that happens to be, I would put "10" in the day-of-month field, and "?" in the day-of-week field. See the examples below for clarification.
- **-** - used to specify ranges. For example, "10-12" in the hour field means "the hours 10, 11 and 12".
- **,** - used to specify additional values. For example, "MON,WED,FRI" in the day-of-week field means "the days Monday, Wednesday, and Friday".
- **/** - used to specify increments. For example, "0/15" in the seconds field means "the seconds 0, 15, 30, and 45". And "5/15" in the seconds field means "the seconds 5, 20, 35, and 50". You can also

specify '/' after the '' **character - in this case** '' is equivalent to having '0' before the '/'. '1/3' in the day-of-month field means "fire every 3 days starting on the first day of the month".

- **L** ("last") - has different meaning in each of the two fields in which it is allowed. For example, the value "L" in the day-of-month field means "the last day of the month" - day 31 for January, day 28 for February on non-leap years. If used in the day-of-week field by itself, it simply means "7" or "SAT". But if used in the day-of-week field after another value, it means "the last xxx day of the month" - for example "6L" means "the last friday of the month". When using the 'L' option, it is important not to specify lists, or ranges of values, as you'll get confusing results.
- **W** ("weekday") - used to specify the weekday (Monday-Friday) nearest the given day. As an example, if you were to specify "15W" as the value for the day-of-month field, the meaning is: "the nearest weekday to the 15th of the month". So if the 15th is a Saturday, the trigger will fire on Friday the 14th. If the 15th is a Sunday, the trigger will fire on Monday the 16th. If the 15th is a Tuesday, then it will fire on Tuesday the 15th. However if you specify "1W" as the value for day-of-month, and the 1st is a Saturday, the trigger will fire on Monday the 3rd, as it will not 'jump' over the boundary of a month's days. The 'W' character can only be specified when the day-of-month is a single day, not a range or list of days.

  > **Note**
  >
  > The 'L' and 'W' characters can also be combined in the day-of-month field to yield 'LW', which translates to "last weekday of the month".

- # - used to specify "the nth" day-of-week day of the month. For example, the value of "6#3" in the day-of-week field means "the third Friday of the month" (day 6 = Friday and "#3" = the 3rd one in the month). Other examples: "2#1" = the first Monday of the month and "4#5" = the fifth Wednesday of the month. Note that if you specify "#5" and there is not 5 of the given day-of-week in the month, then no firing will occur that month.

Examples

Expression	Meaning
0 0 12 * * ?	Fire at 12pm (noon) every day.
0 15 10 ? * *	Fire at 10:15am every day.
0 15 10 * * ?	Fire at 10:15am every day.
0 15 10 * * ? *	Fire at 10:15am every day.
0 15 10 * * ? 2005	Fire at 10:15am every day during the year 2005.
0 * 14 * * ?	Fire every minute starting at 2pm and ending at 2:59pm, every day.
0 0/5 14 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day.
0 0/5 14,18 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at 6pm and ending at 6:55pm, every day.
0 0-5 14 * * ?	Fire every minute starting at 2pm and ending at 2:05pm, every day.
0 10,44 14 ? 3 WED	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
0 15 10 ? * MON-FRI	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday.
0 15 10 15 * ?	Fire at 10:15am on the 15th day of every month.
0 15 10 L * ?	Fire at 10:15am on the last day of every month.
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month.
0 15 10 ? * 6L 2002-2005	Fire at 10:15am on every last friday of every month during the years 2002, 2003, 2004 and 2005.
0 15 10 ? * 6#3	Fire at 10:15am on the third Friday of every month.
0 0 12 1/5 * ?	Fire at 12pm (noon) every 5 days every month, starting on the first day of the month.
0 11 11 11 11 ?	Fire every November 11th at 11:11am.

> **Note**
>
> Pay attention to the effects of '?' and '*' in the day-of-week and day-of-month fields.
>
> Support for specifying both a day-of-week and a day-of-month value is not complete (you must currently use the '?' character in one of these fields).
>
> Be careful when setting fire times between the hours of the morning when "daylight savings" changes occur in your locale (for US locales, this would typically be the hour before and after 2:00 AM - because the time shift can cause a skip or a repeat depending on whether the time moves back or jumps forward.

120.4 Task definition

The EBX® scheduler includes predefined tasks to do the following:

- Delete expired temporary staging folders.
- Clean up (purge) the repository.
- Delete outdated datapace.
- Delete dataspace history.
- Purge execution reports for scheduled tasks.
- Purge workflow proccess history.
- Launch a workflow.

Custom scheduled tasks can be added by the means of **scheduler** `Package` `com.orchestranetworks.scheduler`[API] Java API.

Declare and configure schedules and tasks in the user interface under *Administration* > *Technical configuration* > *Task scheduler*.

120.5 Task configuration

A user must be associated with a task definition; this user will be used to generate the **session** `Session`[API] that will run the task.

> **Note**
>
> The user will not be authenticated, and no password is required. As a consequence, a user with no password set in the directory can only be used to run scheduled tasks.

A custom task can be parameterized by means of a JavaBean specification (getter and setter).

Supported parameter types are:

- java.lang.boolean
- java.lang.int
- java.lang.Boolean
- java.lang.Integer
- java.math.BigDecimal
- java.lang.String

- java.util.Date
- java.net.URI
- java.net.URL

Parameter values are set in XML format.

CHAPTER 121

Background tasks

This chapter contains the following topics:

1. Overview
2. Configuring background tasks
3. Background task overview

121.1 Overview

EBX® has the capability to store very large volumes of data in the repository. Even with a high-performance system, some actions, such as importing, exporting, or transferring large amounts of data, can take a long time to complete. When users have to wait for these tasks to finish before resuming work, it can negatively impact productivity. Configuring these tasks to run in the background can free users to perform other activities in EBX®.

Background tasks are centralized in the user interface on a sidebar. From this interface, users can:

- View progression of a running task.
- Interrupt a running task.
- View the report of a terminated task. Tasks stay visible until users remove them from the view.

121.2 Configuring background tasks

Currently, the TIBCO EBX® Data Exchange (New) Add-on allows you to configure its import, export, and transfer services to run in the background. Please refer to this add-ons' documentation for configuration instructions.

121.3 Background task overview

The following sections provide an overview of background tasks:

- Task properties [p 729]
- Sidebar view [p 730]

Task properties

Tasks include the following properties:

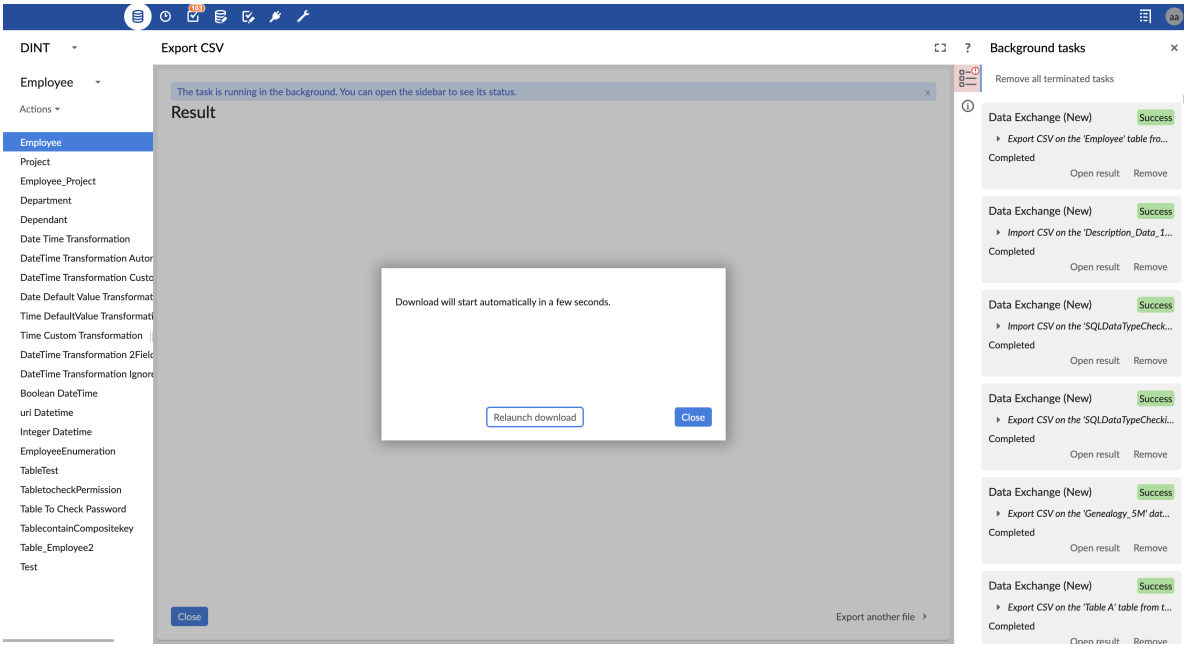
- A unique ID number.

- The submission date.
- The end date (if the task is terminated).
- A localized description.
- An owner; the user that submitted the task.
- A state, which can be one of the following:
  - Pending: the task was submitted but the execution has not yet started.
  - Running: the task is currently executing. Progress information may be available.
  - Interrupting: the task has received an interrupt request but is still executing.
  - Interrupted: the task is stopped because it was interrupted.
  - Success: The task has stopped with no errors.
  - Warning: The task has stopped with a warning. A localized warning message is available.
  - Failed: The task has stopped due to a failure. A localized error message is available.

Sidebar view

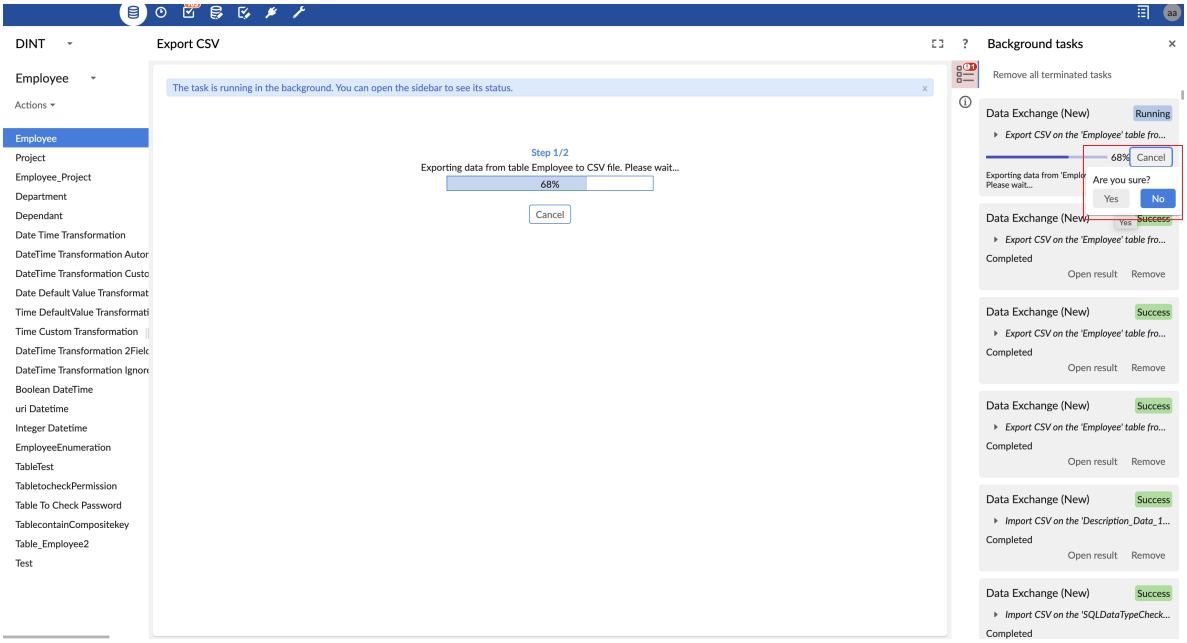
The sidebar view displays all tasks that the user is permitted to view. A task provider can implement two forms of progress information to display to an end user:

- A progress percentage (a number between 0 and 100) displayed as a progress bar.
- A short localized message ("step 1", "step 2", etc.).



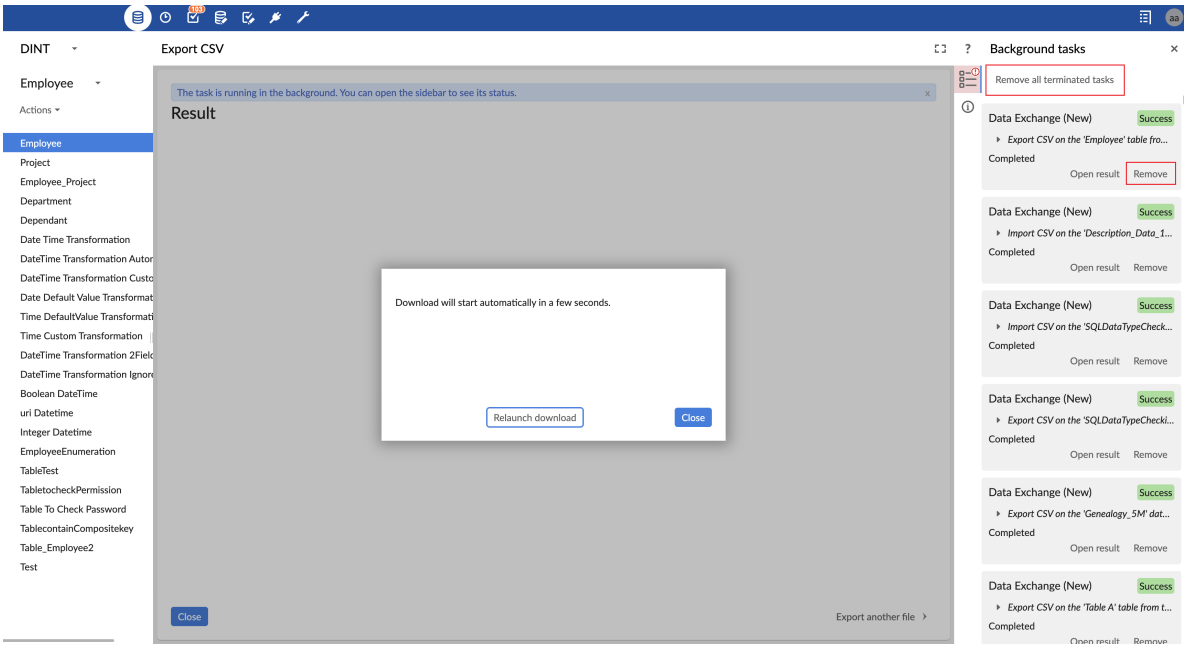
Canceling a running task

A user can cancel a task only if this is implemented by the task provider (some tasks may not be interruptible).



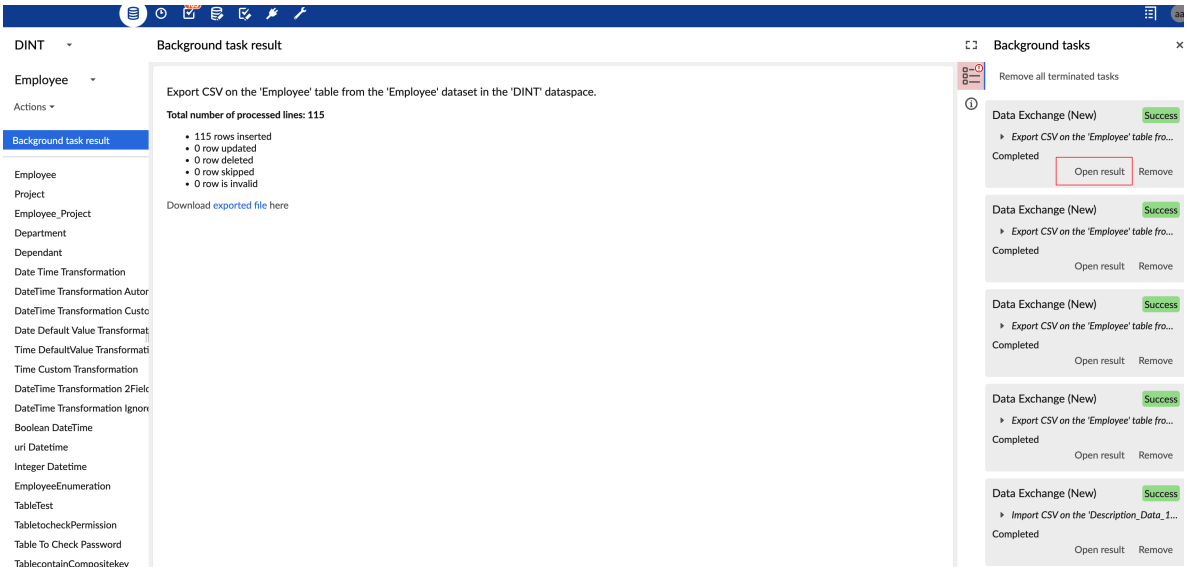
Deleting a terminated task

A terminated task can be deleted from the sidebar any time using the **Remove** or **Remove all terminated tasks** buttons.



Task report

The task report is optional and specific to the type of task. It can be available for the following statuses: "success", "warning", "error" or "interrupted". The report is implemented by the background task provider. If a report is available, a link allows users to navigate to it.



CHAPTER 122

Audit trail

This chapter contains the following topics:

1. Overview
2. Log content
3. Legacy XML Audit trail (deprecated)

122.1 Overview

The audit trail feature allows to log all views access and user services launch in TIBCO EBX®.

The feature is disabled by default; to enable audit trail logging, configure the `audit` logging category with a level that enables `INFO` messages.

See also *Activating the audit trail* *[p 581]*

122.2 Log content

The information logged by audit trail comes in 2 parts: the common header and the event-specific content described in the following sections.

Common header

Here are the header attributes that are always logged:

user-login	Login of the user
user-ip	IP address of the user, when available
event	Type of event performed by the user, among the following value: • TABULAR_ACCESS: when a tabular view is accessed in the UI; • HIERARCHY_EXPAND_NODE: when a node is expanded in a hierarchical view, or when the view is displayed; • HIERARCHY_EXPAND_ALL: when several nodes are expanded in a hierarchical view; • DATASPACE_PERMISSION_TABULAR_ACCESS: when the permission table for a dataspace is accessed; • DATASET_PERMISSION_TABULAR_ACCESS: when the permission table for a dataset is accessed; • USER_SERVICE: when the export, compare, or validate built-in service, an addon or a custom user service is launched.

Table view events

This section describes the events specific to table views, which are:

- TABULAR_ACCESS
- HIERARCHY_EXPAND_NODE
- HIERARCHY_EXPAND_ALL
- DATASPACE_PERMISSION_TABULAR_ACCESS
- DATASET_PERMISSION_TABULAR_ACCESS

These events are triggered when the table view is opened or refreshed, when a search is performed and when the user navigates using pagination.

Here are the content attributes specific to these events:

dataspace	Key of the dataspace containing the table
dataset	Key of the dataset containing the table
table	Path of the table being accessed (does not apply to permission events)
view-id	Identifier of the custom view currently applied if any (does not apply to permission events)
view-name	Published name of the custom view currently applied if any (does not apply to permission events)
pagination-info	Information about the pagination of the tabular view or the expanded node
hierarchy-expanded-node	Identifier of the node being expanded (only applies to hierarchy events). When several levels are expanded (search or expand all), it is the node on the higher level. The identifier is the node label prefixed by ":". The root node is identified as ":". When a hierarchy is initially displayed, only the root node is expanded so the event is HIERARCHY_EXPAND_NODE.
filters	All filters applied to the view, aggregated in a JSON format. There are 3 kinds of filter: • `quick-search`: user input in the quick search field, if any; • `structured-search`: user advanced filters configured using the advanced search panel in a JSON format, if any; • `view-filter`: filter defined by the view in a XPath format, if any. For instance: `filters={"structured-search":["starts-with","/field/city","city"],"view-filter":"matches(./country,'France')"}`

User service

Here are the content attributes specific to this event:

service-key	Service key of the service being executed
service-mode	A service can be executed on any intermediate level of the selection entity. For example, if the current selection entity is a record, then the service may be executed on the following levels: dataspace, dataset, table or a record. This attribute allows to define on which level of the selection entity the service is executed.
selection	Entity selection on which the service is executed (branch or version, dataset, etc.) in a JSON format. When more than 100 records are selected, only the 100th first records are logged.
service-parameters	Additional input parameters specified for the service execution : • for built-in services, see the list of parameters [p 427]; • for custom services, the parameters are specified in the declaration [p 1051]; • for addon services, see the addon documentation.

122.3 Legacy XML Audit trail (deprecated)

XML audit trail logs data updates to XML files. This legacy feature, now deprecated, will be removed in a future version. Although this type of audit trail should no longer be used, the *legacy documentation* is still available.

CHAPTER 123

Collaborative features (Team Up)

This chapter contains the following topics:

1. Introduction
2. Getting started with Team Up
3. Managing tags and tag catalogs

123.1 Introduction

Enable Team Up to facilitate collaboration and engagement between business users. They can use Team Up to leave comments on records, rate records, and assign tags to records. Access to, and visibility of Team Up features is highly customizable. Global settings are available at the data model level, but can be overridden at the dataset level.

The following provides an overview of an administrator's role with Team Up features:

- Comments: Enable commenting on records. Assign moderators to tables that can manage comments from other user profiles.
- Tags: Enable tagging of records. Tags are classified in tag catalogs. Assign an administrator to manage tag catalogs.
- Ratings: Enable ratings for records.

123.2 Getting started with Team Up

If you have administrative access to an EBX® data model, register individual tables to enable Team Up. By default, when a table is registered, all Team Up features are set to active and visible; after the data model is published, users with Read-write or Read-only access can use Team Up. Use Team Up configuration settings to restrict user access and individual features. Additionally, you can configure visibility of each feature. For example, a feature can be active but hidden; users will not see them but previous comments, ratings, or tags are still associated with records.

Registering tables with Team Up

Use the **Team Up** data model extension to register each table:

1. Navigate to *Extensions* > *Team Up* > *Table configuration* and create a new record.

   > **Note**
   >
   > If the **Team up** extension is not visible, see Data model extensions [p 83] for instructions on enabling it.

2. Use the **Table path** menu to select the table where you want to activate Team Up features.
3. Define user permissions, feature availability, moderators, and tag catalogs using the **Ratings**, **Comments**, and **Tags** groups.

   See Managing user access and Team Up features [p 738] for additional details.
4. After saving changes, publish the data model.

Managing user access and Team Up features

Manage user access to Team Up features at the data model and dataset levels. The table below shows the settings available in each location:

	In data models:	**In datasets:**
Activate features	Yes	No
Manage feature visibility	Yes	No
Restrict feature access permissions	Yes	Yes ***
Add moderators for comments	Yes	Yes
Assign tag catalog to table	Yes	No
Configuration location:		
	When viewing the data model navigate to: *Extensions* > *Team Up* and select the table you want to configure.	From the dataset's **Actions** menu select **Team Up features configuration**. Expand the features for each table you want to configure. *** Note: Permission restriction at the dataset level overrides data model permissions.

123.3 Managing tags and tag catalogs

Use tags and tag catalogs to help find and organize related data. Each table that has Team Up enabled is automatically associated with the default tag catalog. You can change the associated catalog. Only

tags in the associated catalog are assignable to the table's records. The following table describes how to perform actions related to tags and tag catalogs:

Creating tag catalogs and administrators	• With administrative access, create tag catalogs in: *Administration > Metadata management > Team Up Administration > Tags > Tag catalogs*. • Assign an administrator to a catalog: *Administration > Metadata management > Team Up Administration > Tags > Tags catalog administrators*.
Creating tags	• A new tag is automatically created when a business users assigns a tag to a record that does not exist in the tag catalog. • With administrative access, create tags in: *Administration > Metadata management > Team Up Administration > Tags > Tags*.
Assigning tags to catalogs	Change the catalog a tag belongs to: *Administration > Metadata management > Team Up Administration > Tags > Tags*. Note that this tag will no longer be available for records it was assigned to.

CHAPTER 124

AI Assistant

This chapter contains the following topics:

1. Introduction
2. LLM configuration
3. Assistance functions

124.1 Introduction

The AI Assistant leverages Generative AI—specifically Large Language Models (LLMs)—to help EBX® users complete tasks by generating suggestions based on record data. Typical use cases include generating text, summarizing text, translating values and standardizing formats.

> **Note**
>
> EBX® requires connection to an external LLM provider service, as it does not include nor provide any. See below for more information.

124.2 LLM configuration

An LLM configuration defines the LLM provider service and the model to be used. An active account with a cloud-based LLM provider service (e.g. OpenAI, Azure OpenAI), or a local/on-premise LLM server (e.g. Ollama) is required. While configuring the LLM, you will need to have any required credential and network access to reach the provider's API.

To create an LLM configuration:

1. Go to *Administration > AI Assistant > AI Assistant administration*.
2. Open the **LLMs** table and create a record to add a new LLM configuration.
3. Select your LLM provider and type. After choosing the provider, an information message displays with details on the key to use when creating an **External service configuration** in the next step.

   If the required LLM model isn't listed under **LLM type**, follow these steps to specify it:

   1. From the **LLM type** dropdown, select **[custom]**.
   2. Click **Save** to display the **LLM custom type** field.
   3. Enter the name of the required LLM model in the **LLM custom type** field.
4. Select an **External service configuration**, or create a new one:

1. Open the menu and select **+ Create**.
2. After adding the id, provide the following:
   - **URL**: You only need to provide the correct base URL for your specific service. EBX® automatically appends the necessary API paths (e.g. `/chat/completions`) when constructing the full request URL. The following lists provider-specific examples:

Open AI	The URL is: `https://api.openai.com/v1`
Azure Open AI	The URL corresponds to your specific resource endpoint and uses the following format, which can be found in the Azure portal: `https://YOUR_RESOURCE_NAME.openai.azure.com/` where `YOUR_RESOURCE_NAME` is the unique name assigned to the Azure Open AI service when created in the Microsoft Azure portal.
Ollama	The URL is the address of your Ollama service in the following format: `http://OLLAMA_SERVER_ADDRESS:PORT` • `OLLAMA_SERVER_ADDRESS` is the address of the server running your Ollama service. • `PORT` is the port number that the Ollama service is configured to use.

   - **Usage**: leave **AI Assistant** checked
   - **Authentication**: Select **Custom** if a key is required to connect to the LLM provider or **None** if no key is required.
   - **Keys** / **Secret keys**: When applicable, add the key that was displayed in the earlier step in the **Keys** list or in the **Secret keys** list, depending on your security requirements. Then paste your API key in the **Value** field.

     Keys defined in the **Secret keys** list are encrypted and securely stored in the EBX® repository. For more information, see Keystore file [p 716].
3. Save and close the record.

5. Ensure the external configuration you just created is selected and save and close the record.

The LLM configuration is now available to use when you create an Assistance function.

124.3 Assistance functions

An assistance function runs in the context of a specific table and defines how the AI Assistant uses an LLM to generate useful suggestions for end users. Specifically, it uses record data, user input, and a predefined prompt to suggest a value for one or more fields in the record.

Creating assistance functions

To create an assistance function:

1. Open a data model in the Data Model Assistant (DMA).
2. Ensure that the **AI Assistant** data model extension is enabled. See Data model extensions [p 83] for instructions on how to enable it.
3. Navigate to *Extensions* > *AI Assistant* > *Assistance functions* and create a new record.
4. Define the input and output fields, the prompt and a proper description. See the next section for more details on each field.
5. Write the prompt that the LLM will use.
6. Save the function.
7. Use the **Preview function** service to test the function before publishing.
8. Publish the data model to make the assistance function available to end users.

> **Note**
>
> Always test functions with realistic data to verify that the LLM returns expected results.

Assistance function fields description

Table Path	The table where the assistance function will run.
Label and description	A localized label and description for the assistance function.
Enabled	Marks the assistance function as enabled or disabled.
Input fields	Table fields that provide values to the prompt. For each input field, provide the following information: • **Field:** The field path in the selected table. • **Alias:** A unique identifier to use in the prompt.
Virtual input fields	Optional fields that do not exist in the table, but provide additional inputs to the assistance function. For each virtual input field, provide the following information: • **Alias:** A unique identifier to use in the prompt. • **Data type:** The data type for the virtual input field. • **Label and description:** A localized label and description for the virtual input field. • **Maximum length:** The maximum number of characters allowed for the field (applies only to text fields).
Output fields	The fields where the generated result is saved. For each output field, provide the following information: • **Field:** The path in the selected table to the output field. **Warning:** When you use a foreign key as an output, be aware of the following consequences: • All labels from the referenced table are sent to the LLM, which might expose confidential data. • Sending a large list of labels can impact performance.
Virtual output fields	Optional fields that do not exist in the table, but provide additional outputs to the assistance function. For each virtual output field, provide the following information: • **Identifier:** A unique identifier for the virtual output field. • **Data type:** The data type for the virtual output field.

- **Label and description:** A localized label and description for the virtual output field.

Prompt	The instruction for the LLM to guide its generated output. Reference the input fields and the virtual input fields in the prompt using the placeholder pattern **${Alias}**, where **Alias** is an input field alias or a virtual input field alias.

Examples of assistance functions

Function	Implementation
Spell checking assistance function	A function to help users spell check the value of a string field. **Prompt:** Spell check ${string_field}. Where **string_field** is the alias of the field to spell check and is configured as the input and output of the function.
Text summarization assistance function	A function to help users summarize the value of a string field. **Prompt:** Summarize ${long_string_field}. Where **long_string_field** is the alias of the field to summarize and is configured as the input of the function.
Translation assistance function	A function to help users translate the value of a string field. **Prompt:** Translate ${input_string_field} to French. Where **input_string_field** is the alias of the field to translate and is configured as the input of the function.
Postal address standardization assistance function	A function to help users standardize a postal address. **Prompt:** Standardize the postal address: ${string_field}. Where **string_field** is the alias of the field to standardize and is configured as the input and output of the function.
Description generation assistance function	A function to help users generate a description based on the fields of a record. **Prompt:** Generate a description based on the following information: label: ${label_field}, category: ${category_field}. features: ${features_field}. Where **label_field**, **category_field** and **features_field** are the aliases of the fields configured as inputs of the function.

CHAPTER 125

Other

This chapter contains the following topics:

1. Lineage
2. Event broker

125.1 Lineage

To administer lineage, three tables are accessible:

- **Authorized profiles**: Profiles must be added to this table to be used for data lineage WSDL generation.
- **History**: Lists the general data lineage WSDLs and their configuration.
- **JMS location**: Lists the JMS URL locations.

125.2 Event broker

Overview

TIBCO EBX® offers the ability to receive notifications and information related to specific events using the event broker feature. This feature consists in sending notifications related to EBX® core events to the subscriber according to their chosen topics.

Terminology

Event broker	Notification component for loosely-coupled event handling. Consists of dispatching fired events from EBX® core to concerned subscribers. The event broker is mainly used for monitoring and statistical purposes.
Topic	Corresponds to the EBX® event type that contains messages. The number of subscribers registered to a topic is unlimited.
Subscriber	Client implementation in the modules that receive the events related to the subscribed topic(s).

Topics

Dataspace and snapshot	Corresponds to operations in the dataspace and in the snapshot, such as: create, close, reopen, delete, archive export and archive import (only for dataspace merge).
Repository	Corresponds to operations in the repository, such as: start-up and purge.
User session	Corresponds to the operations related to user authentication, such as: login and logout.

Administration

The management console is located under 'Event broker' in the 'Administration' area. It contains three tables: 'Topics', 'Subscribers' and 'Subscriptions'.

All content is read-only, except for the following operations:

- Topics and subscribers can be manually activated or deactivated using dedicated services.
- Subscribers that are no longer registered to the broker can be deleted.

The event broker is based on a thread pool mechanism. The maximum number of threads can be defined in the properties file as follows:

```
# Defines the number of thread pool executors to
# guarantee the publication of asynchronous events.
# The default value is 2
ebx.eventBroker.threadPool.size=2
```

Distributed Data Delivery (D3)

CHAPTER **126**

Introduction to D3

This chapter contains the following topics:

1. Overview
2. D3 terminology
3. Known limitations

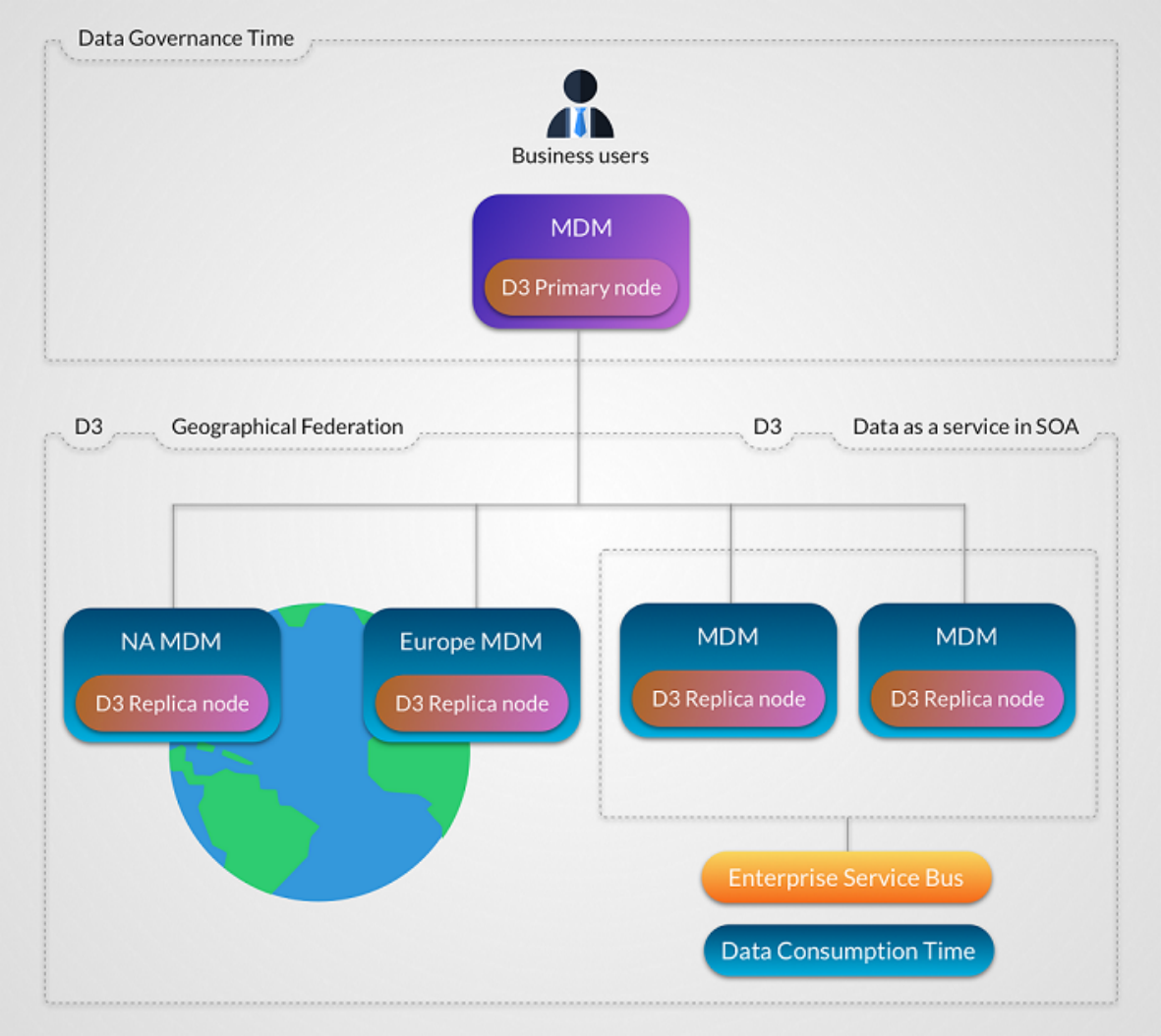
126.1 Overview

TIBCO EBX® offers the ability to send data from an EBX® instance to other instances. Using a broadcast action, it also provides an additional layer of security and control to the other features of EBX®. It is particularly suitable for situations where data governance requires the highest levels of data consistency, approvals and the ability to rollback.

D3 architecture

A typical D3 installation consists of one primary node and multiple replica nodes. In the primary node, a Data Steward declares which dataspaces must be broadcast, as well as which user profile is allowed to broadcast them to the replica nodes. The Data Steward also defines delivery profiles, which are groups of one or more dataspaces.

Each replica node must define from which delivery profile it receives broadcasts.



Involving third-party systems

The features of D3 also allow third-party systems to access the data managed in EBX® through data services. Essentially, when a system consumes the data of a delivery dataspace, the data is transparently redirected to the last broadcast snapshot. This ensures a more controlled and reliable view of the managed data.

Third-party systems can either access data directly through the primary node or through a replica node. Thus, a physical architecture consisting of a primary node and no replica nodes is possible.

Protocols

If JMS is activated, the conversation between a primary node and a replica node is based on SOAP over JMS, while archive transfer is based on JMS binary messages.

If JMS is not activated, conversation between a primary node and a replica node is based on SOAP over HTTP(S), while binary archive transfer is based on TCP sockets. If HTTPS is used, make sure that the target node connector is correctly configured by enabling SSL with a trusted certificate.

See also *JMS for distributed data delivery (D3)* *[p 759]*

126.2 D3 terminology

broadcast	Send a publication of an official snapshot of data from a primary node to replica nodes. The broadcast transparently and transactionally ensures that the data is transferred to the replica nodes.
delivery dataspace	A delivery dataspace is a dataspace that can be broadcast to authenticated and authorized users using a dedicated action. By default, when a data service accesses a delivery dataspace on any node, it is redirected to the last snapshot that was broadcast. See Data services [p 757].
delivery profile	A delivery profile is a logical name that groups one or more delivery dataspaces. Replica nodes subscribe to one or more delivery profiles.
cluster delivery mode	Synchronization with subscribed replica nodes is performed in a two-phase commit transactional process. This delivery mode is designed to respond to a high volume of queries using load balancing and/or fault tolerance. It ensures the consistency of data in the cluster between replica nodes and their primary node delivery dataspaces. Primary and replica nodes use the same last broadcast snapshots.
federation delivery mode	Synchronization is performed in a single phase, and with each registered replica node independently. This delivery mode is designed to be used with geographically distributed and/or heterogeneous architectures where response time and network availability cannot be guaranteed. At any one time, replica nodes can be at different last broadcast snapshots. The synchronization processes are thus independent of one another and replay of individual replica nodes are performed for certain broadcast failures.
Primary node	An instance of EBX® that can define one or more delivery dataspaces, and to which replica nodes can subscribe. A primary node can also act as a regular EBX® server.
Replica node	An instance of EBX® attached to a primary node, in order to receive delivery dataspace broadcasts. Besides update restrictions on delivery dataspaces, the replica node acts as a regular EBX® server.

Hub node	An instance of EBX® acting as both a primary node and a replica node. Primary delivery dataspaces and replica node delivery dataspaces **must** be disjoint.

126.3 Known limitations

General limitations

- Each replica node must have only one primary node.
- Embedded data models cannot be used in D3 dataspaces. Therefore, it is not possible to create a dataset based on a publication in a D3 dataspace.
- The compatibility is not assured if at least one replica node product version is different from the primary node.

Broadcast and delivery dataspace limitations

- Access rights on dataspaces are not broadcast, whereas access rights on datasets are.
- Dataspace information is not broadcast.
- If a dataspace and its parent are broadcast, their parent-child relationship will be lost in the replica nodes.
- Once a snapshot has been broadcast to a replica, subsequent broadcasts of *any* snapshot with the same name will result in restoring the originally broadcast version of that same name on the replica node. That is, if the original snapshot on the primary node is purged and a new one is created with the same name and subsequently broadcast, then the content of the replica will be restored to that of the previously broadcast snapshot, and not to the latest one of the same name.
- To guarantee dataspace consistency between D3 nodes, the data model on which the broadcast contents are based, must be the same between the primary node and its replica nodes.
- On a replica delivery dataspace, if several replica nodes are registered, and if replication is enabled in data models, it will be effective for all nodes. No setting is available to activate/deactivate replication according to D3 nodes.
- Replication on replica nodes does not take part in the distributed transaction: it is automatically triggered after commit.

Administration limitations

Technical dataspaces cannot be broadcast, thus the EBX® default user directory cannot be synchronized using D3.

CHAPTER 127

D3 broadcasts and delivery dataspaces

This chapter contains the following topics:

1. Broadcast
2. Replica node registration
3. Accessing delivery dataspaces

127.1 Broadcast

Scope and contents of a broadcast

A D3 broadcast occurs at the dataspace or snapshot level. For dataspace broadcasts, D3 first creates a snapshot to capture the current state, then broadcasts this newly created snapshot.

A broadcast performs one of the following procedures depending on the situation:

- An update of the differences computed between the new broadcast snapshot and the current 'commit' one on the replica node.
- A full synchronization containing all datasets, tables, records, and permissions. This is done on the first broadcast to a given replica node, if the previous replica node commit is not known to the primary node, or on demand using the user service in '[D3] Primary node configuration'.

See also *Services on primary nodes* *[p 772]*

Performing a broadcast

The broadcast can be performed:

- By the end-user, using the **Broadcast** action available in the dataspace or snapshot (this action is available only if the dataspace is registered as a delivery dataspace)
- Using custom Java code that uses `D3NodeAsMaster`[API].

Conditions

In order to be able to broadcast, the following conditions must be fulfilled:

- The authenticated user profile has permission to broadcast.

- The dataspace or snapshot to be broadcast has no validation errors.

  Note: Although it is not recommended, it is possible to force a broadcast of a delivery dataspace that contains validation errors. In order to do this, set the maximum severity threshold allowed in a delivery dataspace validation report under '[D3] Primary node configuration' in the 'Administration' area.
- The D3 primary node configuration has no validation errors on the following scope: the technical record of the concerned delivery dataspace and all its dependencies (dependent delivery mappings, delivery profiles and registered replica nodes).
- There is an associated delivery profile.
- If broadcasting a dataspace, the dataspace is not locked.
- If broadcasting a snapshot, the snapshot belongs to a dataspace declared as delivery dataspace and is not already the current broadcast snapshot (though a rollback to a previously broadcast snapshot is possible).
- The dataspace or snapshot contains differences compared to the last broadcast snapshot.

Persistence

When a primary node shuts down, all waiting or in progress broadcast requests abort, then they will be persisted on a temporary file. On startup, all aborted broadcasts are restarted.

See also *Temporary files* *[p 774]*

Destination

On the target replica or hub node side:

- The `ebx-d3-reference` dataspace identifier is the common parent of all the delivery dataspaces.
- The delivery dataspace has the same identifier in primary, replica or hub nodes.
- If the delivery dataspace is missing, it will be created on the first or on the full synchronization broadcast.
- If the delivery dataspace already exists on the first broadcast or full synchronization, it will be overridden.
- If an existing dataspace with the same identifier as the delivery one is detected outside of the `ebx-d3-reference`, an error will be raised.

See also *Known limitations* *[p 754]*

> **Note**
>
> Broadcasts are performed asynchronously. Therefore, no information is displayed in the user interface about the success or failure of a broadcast. Nevertheless, it is possible to monitor the broadcast operations inside '[D3] Primary node configuration'. See Supervision [p 773].

127.2 Replica node registration

Scope and contents

An initialization occurs at the replica node level according to the delivery profiles registered in the TIBCO EBX® main configuration file of the replica node. When the primary node receives that initialization request, it creates or updates the replica node entry, then sends the last broadcast snapshot of all registered delivery dataspaces.

> **Note**
>
> If the registered replica node repository ID or communication layer already exists, the replica node entry in the 'Registered replica nodes' technical table is updated, otherwise a new entry is created.

Performing an initialization

The initialization can be done:

- Automatically at replica node server startup.
- Manually when calling the replica node service 'Register replica node'.

Conditions

To be able to register, the following conditions must be fulfilled:

- The D3 mode must be 'hub' or 'slave'.
- The primary and replica node authentication parameters must correspond to the primary node administrator and replica node administrator defined in their respective directories.
- The delivery profiles defined on the replica node must exist in the primary node configuration.
- All data models contained in the registered dataspaces must exist in the replica node. If embedded, the data model names must be the same. If packaged, they must be located at the same module name and the schema path in the module must be the same in both the primary and replica nodes.
- The D3 primary node configuration has no validation error on the following scope: the technical record of the registered replica node and all its dependencies (dependent delivery profiles, delivery mappings and delivery dataspaces).

> **Note**
>
> To set the parameters, see the replica or hub EBX® properties in Configuring primary, hub and replica nodes [p 769].

127.3 Accessing delivery dataspaces

Data services

By default, when a data service accesses a delivery dataspace, it is redirected to the current snapshot, which is the last broadcast one. However, this default behavior can be modified either at the request level or in the global configuration.

See also *Common parameter 'disableRedirectionToLastBroadcast' [p 1082]*

Access restrictions

On the primary node, a delivery dataspace can neither be merged nor closed. Other operations are available depending on permissions. For example, modifying a delivery dataspace directly, creating a snapshot independent from a broadcast, or creating and merging a child dataspace.

On the replica node, aside from the broadcast process, no modifications of any kind can be made to a delivery dataspace, whether by the end-user, data services, or a Java program. Furthermore, any dataspace-related operations, such as merge, close, etc., are forbidden on the replica node.

D3 broadcast Java API

The last broadcast snapshot may change between two calls if a broadcast has taken place in the meantime. If a fully stable view is required for several successive calls, these calls need to specifically refer to the same snapshot.

To get the last broadcast snapshot, see `D3Node.getBroadcastVersion`[API].

CHAPTER 128

D3 JMS Configuration

This chapter contains the following topics:

1. JMS for distributed data delivery (D3)

128.1 JMS for distributed data delivery (D3)

To configure D3 to use JMS instead of the default HTTP and TCP protocols, you must configure the JMS connection factory [p 541] and the following queues declared in the `WEB-INF/web.xml` deployment descriptor of the '`ebx`' web application.

> **Note**
>
> If the TIBCO EBX® main configuration does not activate JMS and D3 ('slave', 'hub' or 'master' node) through the properties `ebx.d3.mode`, `ebx.jms.activate` and `ebx.jms.d3.activate`, then the environment entries below will be ignored by EBX® runtime. See JMS [p 588] and Distributed data delivery (D3) [p 588] in the EBX® main configuration properties for more information on these properties.

Common declarations on primary and replica nodes (for shared queues)

Reserved resource name	Default JNDI name	Description
`jms/EBX_D3MasterQueue`	JBoss: `java:/jms/EBX_D3MasterQueue`	D3 primary JMS queue (only for D3 mode 'slave' or 'hub'). It specifies the queue name used to send SOAP requests to the D3 primary node. The message producer sets the primary node repository ID as a value of the header field `JMSType`. Java type: `jakarta.jms.Queue`
`jms/EBX_D3ReplyQueue`	JBoss: `java:/jms/EBX_D3ReplyQueue`	D3 Reply JMS queue (for all D3 modes except the 'single' mode). It specifies the name of the reply queue for receiving SOAP responses. The consumption is filtered using the header field `JMSCorrelationID`. Java type: `jakarta.jms.Queue`
`jms/EBX_D3ArchiveQueue`	JBoss: `java:/jms/EBX_D3ArchiveQueue`	D3 JMS Archive queue (for all D3 modes except the 'single' mode). It specifies the name of the transfer archive queue used by the D3 node. The consumption is filtered using the header field `JMSCorrelationID`. If the archive weight is higher than the threshold specified in the property `ebx.jms.d3.archiveMaxSizeInKB`, the archive will be divided into several sequences. Therefore, the consumption is filtered using the header fields `JMSXGroupID` and `JMSXGroupSeq` instead. Java type: `jakarta.jms.Queue`
`jms/EBX_D3CommunicationQueue`	JBoss: `java:/jms/EBX_D3CommunicationQueue`	D3 JMS Communication queue (for all D3 modes except 'single' mode). It specifies the name of the communication queue where the requests are received. The consumption is filtered using the header field `JMSType` which corresponds to the current repository ID. Java type: `jakarta.jms.Queue`

> **Note**
>
> These JNDI names are set by default, but can be modified inside the web application archive `ebx.war`, included in `EBX.ear` (if using JBoss, Websphere or other application servers).

Optional declarations on primary nodes (for replica-specific queues)

> **Note**
>
> Used for ascending compatibility prior to 5.5.0 or for mono-directional queues topology.

The deployment descriptor of the primary node must be manually modified by declaring specific communication and archive queues for each replica node. It consists in adding resource names in 'web.xml' inside 'ebx.war'. The replica-specific node queues can be used by one or more replica nodes.

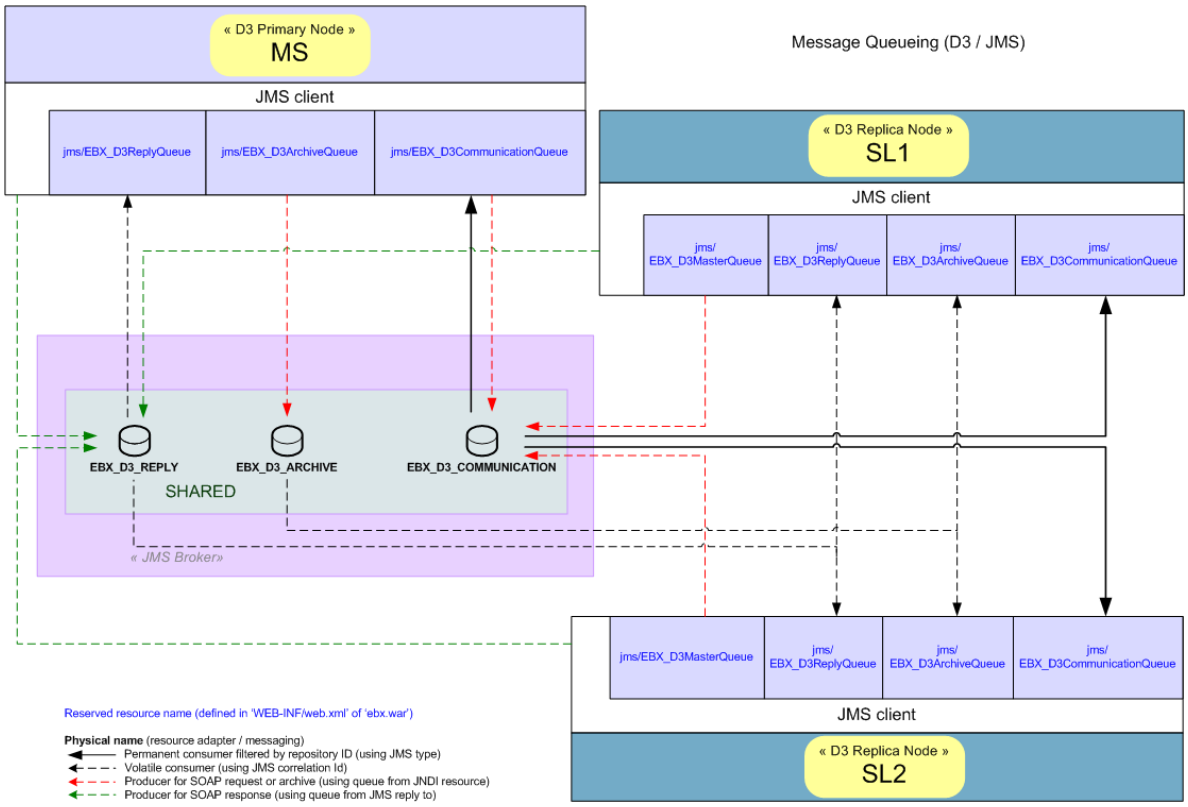
Resources can be freely named, but the physical names of their associated queue must correspond to the definition of replica nodes for resources `jms/EBX_D3ArchiveQueue` and `jms/EBX_D3CommunicationQueue`.

> **Note**
>
> Physical queue names matching: on registration, the replica node sends the communication and archive physical queue names. These queues are matched by physical queue name among all resources declared on the primary node. If unmatched, the registration fails.

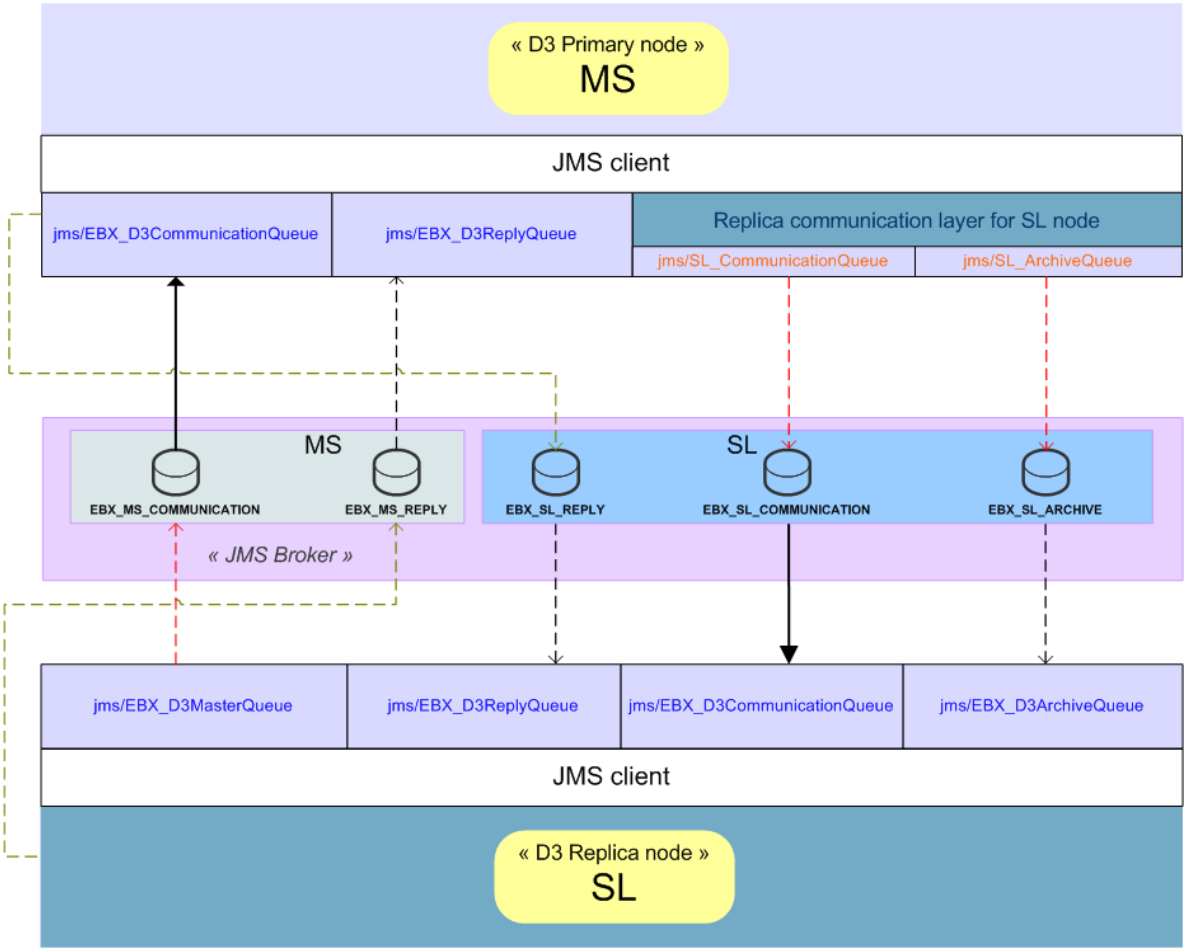
Examples of JMS configuration

	Shared queues	**Specific queues**
Primary-Replica nodes architecture	Between a primary node and two replica nodes with shared queues [p 762]	Between a primary node and a replica node with replica-specific queues [p 763]
Hub-Hub architecture	Between two hub nodes with shared queues [p 764]	Between two hub nodes with replica-specific queues [p 765]

Between a primary node and two replica nodes with shared queues



Between a primary node and a replica node with replica-specific queues

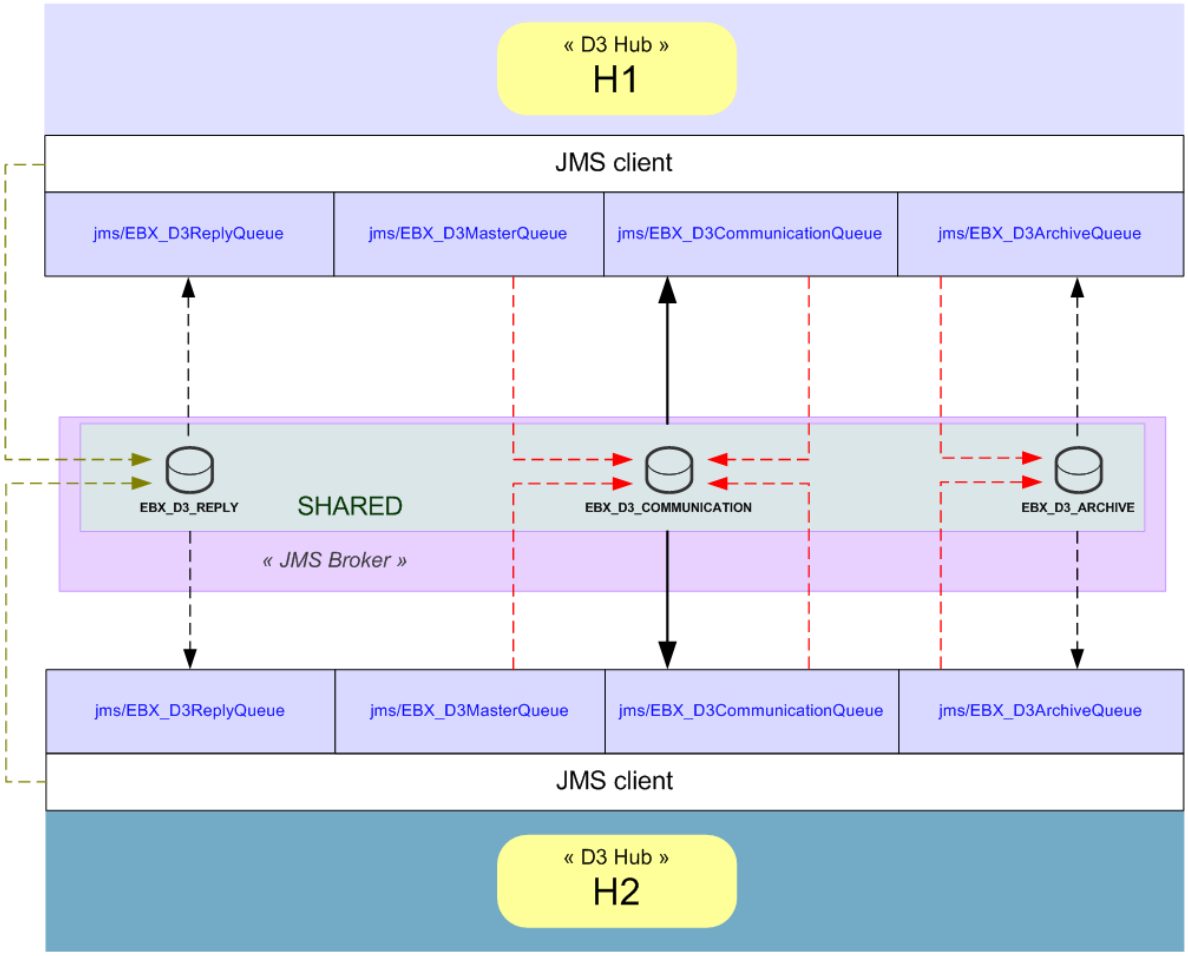


Reserved or custom resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

Physical name (resource adapter / messaging)

- Permanent consumer filtered by repository ID (using JMS type)
- Volatile consumer (using JMS correlation Id)
- Producer for SOAP request or archive (using queue from JNDI resource)
- Producer for SOAP response (using queue from JMS reply to)

Between two hub nodes with shared queues

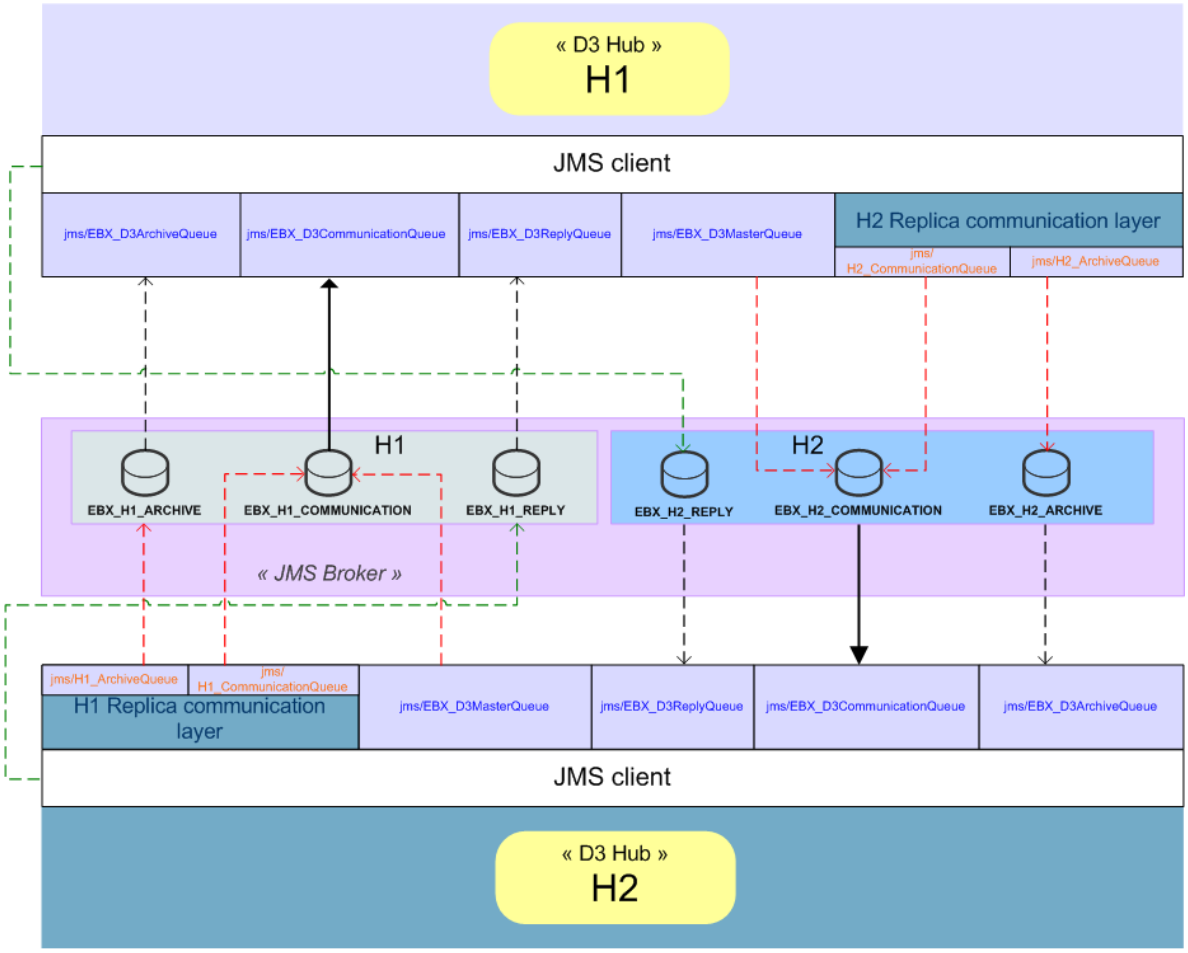


Reserved resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

Physical name (resource adapter / messaging)

- Permanent consumer filtered by repository ID (using JMS type)
- Volatile consumer (using JMS correlation Id)
- Producer for SOAP request or archive (using queue from JNDI resource)
- Producer for SOAP response (using queue from JMS reply to)

Between two hub nodes with replica-specific queues



Reserved or custom resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

Physical name (resource adapter / messaging)

- Permanent consumer filtered by repository ID (using JMS type)
- Volatile consumer (using JMS correlation Id)
- Producer for SOAP request or archive (using queue from JNDI resource)
- Producer for SOAP response (using queue from JMS reply to)

CHAPTER 129

D3 administration

This chapter contains the following topics:

1. Quick start
2. Configuring D3 nodes
3. Supervision

129.1 Quick start

This section introduces the configuration of a basic D3 architecture with two TIBCO EBX® instances. Before starting, please check that each instance can work properly with its own repository.

> **Note**
>
> Deploy EBX® on two different web application containers. If both instances are running on the same host, ensure that all communication TCP ports are distinct.

Declare an existing dataspace on the primary node

The objective is to configure and broadcast an existing dataspace from a *primary* node.

This configuration is performed on the entire D3 infrastructure (primary [p 753] and replica [p 753] nodes included).

Update the `ebx.properties`*primary* node configuration file with:

1. Define D3 mode as `primary` in key `ebx.d3.mode`.

> **Note**
>
> The *primary* node can be started after the configuration.

After authenticating as a built-in administrator, navigate within the administration tab:

1. Prerequisite: Check that the node is configured as a *primary* node (in the 'Actions' menu use 'System information' and check 'D3 mode').
2. Open the '[D3] Primary configuration' administration feature.
3. Add the dataspace to be broadcast to the 'Delivery dataspaces' table, and declare the allowed profile.
4. Add the delivery profile [p 753] to the 'Delivery profiles' table (it must correspond to a logical name) and declare the delivery mode. Possible values are: cluster mode [p 753] or federation mode [p 753].

5. Map the delivery dataspace with the delivery profile into the 'Delivery mapping' table.

> **Note**
>
> The *primary* node is now ready for the replica node(s) registration on the delivery profile.
>
> Check that the D3 broadcast menu appears in the 'Actions' menu of the dataspace or one of its snapshots.

Configure replica node for registration

The objective is to configure and register the *replica* node based on a delivery profile and communications settings.

Update the `ebx.properties` replica node configuration file with:

1. Define D3 mode as `replica` in key `ebx.d3.mode`.
2. Define the delivery profile [p 753] set on the *primary* node in key `ebx.d3.delivery.profiles` (delivery profiles must be separated by a comma and a space).
3. Define the *primary* node user authentication (must have the built-in administrator profile) for node communications in `ebx.d3.master.username` and `ebx.d3.master.password`.
4. Define HTTP/TCP protocols [p 771] for *primary* node communication, by setting a value for the property key `ebx.d3.master.url`

   (for example `http://localhost:8080/ebx-dataservices/connector`).
5. Define the *replica* node user authentication (must have the built-in administrator profile) for node communications in `ebx.d3.slave.username` and `ebx.d3.slave.password`.
6. Define HTTP/TCP protocols [p 771] for *replica* node communication, by setting a value for the property key `ebx.d3.slave.url`

   (for example `http://localhost:8090/ebx-dataservices/connector`).

> **Note**
>
> The *replica* node can be started after the configuration.

After authenticating as a built-in administrator, navigate inside the administration tab:

1. Prerequisite: Check that the node is configured as the *replica* node (in the 'Actions' menu use 'System information' and check 'D3 mode').
2. Open the '[D3] Replica configuration' administration feature.
3. Check the information on the 'Primary information' screen: No field should have the 'N/A' value.

> **Note**
>
> Please check that the model is available before broadcast (from data model assistant, it must be published).
>
> The *replica* node is then ready for broadcast.

129.2 Configuring D3 nodes

Runtime configuration of primary and hub nodes through the user interface

The declaration of delivery dataspaces and delivery profiles is done by selecting the '[D3] Primary configuration' feature from the 'Administration' area, where you will find the following tables:

Delivery dataspaces	Declarations of the dataspaces that can be broadcast.
Delivery profiles	Profiles to which replica nodes can subscribe. The delivery mode must be defined for each delivery profile.
Delivery mapping	The association between delivery dataspaces and delivery profiles.

> **Note**
>
> The tables above are read-only while some broadcasts are pending or in progress.

Configuring primary, hub and replica nodes

This section details how to configure a node in its EBX® main configuration file.

See also *Overview* *[p 572]*

Primary node

In order to act as a *primary* node, an instance of EBX® must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=master` node:

```
######################################################################
## D3 configuration
######################################################################
######################################################################
# Configuration for master, hub and slave
######################################################################
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=master
```

See also *primary node* *[p 753]*

Hub node

In order to act as a *hub* node (combination of primary and replica node configurations), an instance of EBX® must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=hub` node:

```
######################################################################
## D3 configuration
######################################################################
######################################################################
# Configuration for master, hub and slave
```

```
######################################################################
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=hub

######################################################################
# Configuration dedicated to hub or slave
######################################################################
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

See also *hub node [p 754]*

Replica node

In order to act as a *replica* node, an instance of EBX® must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=slave` node:

```
######################################################################
## D3 configuration
######################################################################
######################################################################
# Configuration for master, hub and slave
######################################################################
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=slave

######################################################################
# Configuration dedicated to hub or slave
######################################################################
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

See also *replica node [p 753]*

Configuring the network protocol of a node

This section details how to configure the network protocol of a node in its EBX® main configuration file.

See also *Overview [p 572]*

HTTP(S) and socket TCP protocols

Sample configuration for `ebx.d3.mode=hub` or `ebx.d3.mode=slave` node with HTTP(S) network protocol:

```
####################################################################
# HTTP(S) and TCP socket configuration for D3 hub and slave
####################################################################
# URL to access the data services connector of the master
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: [protocol]://[master_host]:[master_port]/ebx-dataservices/connector
# Where the possible values of 'protocol' are 'http' or 'https'.
ebx.d3.master.url=

# URL to access the data services connector of the slave
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: [protocol]://[slave_host]:[slave_port]/ebx-dataservices/connector
# Where the possible values of 'protocol' are 'http' or 'https'.
ebx.d3.slave.url=

# Minimum port to use to transfer archives on TCP mode.
# Must be a positive integer above zero and below 65535.
# If not set, a random port will be used.
#ebx.d3.slave.socket.range.min=

# Max port to use on TCP mode to transfer archives.
# Must be a positive integer above ebx.d3.slave.socket.range.min and below 65535.
# Mandatory if ebx.d3.slave.socket.range.min is set.
#ebx.d3.slave.socket.range.max=
```

JMS protocol

If JMS is activated, the following properties can be defined in order to enable JMS functionalities for a D3 node.

Sample configuration for all D3 nodes with JMS network protocol:

```
####################################################################
## JMS configuration for D3
####################################################################
# Taken into account only if Data Services JMS is configured properly
####################################################################
# Configuration for master, hub and slave
####################################################################
# Default is false, activate JMS for D3
## If activated, the deployer must ensure that the entries
## 'jms/EBX_D3ReplyQueue', 'jms/EBX_D3ArchiveQueue' and 'jms/EBX_D3CommunicationQueue'
## are bound in the operational environment of the application server.
## On slave or hub mode, the entry 'jms/EBX_D3MasterQueue' must also be bound.
ebx.jms.d3.activate=false

# Change the default timeout when using reply queue.
# Must be a positive integer that does not exceed 3600000.
# Default is 10000 milliseconds.
#ebx.jms.d3.reply.timeout=10000

# Time-to-live message value expressed in milliseconds.
# This value will be set on each message header 'JMSExpiration' that defines the
# countdown before the message deletion managed by the JMS broker.
# Must be a positive integer equal to 0 or above the value of 'ebx.jms.d3.reply.timeout'.
# The value 0 means that the message does not expire.
# Default is 3600000 (one hour).
#ebx.jms.d3.expiration=3600000

# Archive maximum size in KB for the JMS body message. If exceeds, the message
# is transferred into several sequences messages in a same group, where each one does
# not exceed the maximum size defined.
# Must be a positive integer equals to 0 or above 100.
# Default is 0 that corresponds to unbounded.
#ebx.jms.d3.archiveMaxSizeInKB=

####################################################################
# Configuration dedicated to hub or slave
####################################################################
# Master repository ID, used to set a message filter for the concerned master when sending JMS message
# Mandatory property if ebx.jms.d3.activate=true and if ebx.d3.mode=hub or ebx.d3.mode=slave
```

```
#ebx.jms.d3.master.repositoryId=
```

See also *JMS for distributed data delivery (D3)* *[p 759]*

Services on primary nodes

Services to manage a primary node are available in the 'Administration' area of the replica node under '[D3] Primary node configuration' and also in the 'Delivery dataspaces' and 'Registered replica nodes' tables. The services are:

Relaunch replays	Immediately relaunch all replays for waiting federation deliveries.
Delete replica node delivery dataspace	Delete the delivery dataspace on chosen replica nodes and/or unregister it from the configuration of the D3 primary node. To access the service, select a delivery dataspace from the 'Delivery dataspaces' table on the primary node, then launch the wizard.
Fully resynchronize	Broadcast the full content of the last broadcast snapshot to the registered replica nodes.
Subscribe a replica node	Subscribe a set of selected replica nodes.
Deactivate replica nodes	Remove the selected replica nodes from the broadcast scope and switch their states to 'Unavailable'. **Note** The "in progress" broadcast contexts are rolled back.
Unregister replica nodes	Disconnects the selected replica nodes from the primary node. **Note** The "in progress" broadcast contexts are rolled back.

Note

The primary node services above are hidden while some broadcasts are pending or in progress.

Services on replica nodes

Services are available in the 'Administration' area under *[D3] Configuration of replica* node to manage its subscription to the primary node and perform other actions:

Register replica node	Re-subscribes the replica node to the primary node if it has been unregistered.
Unregister replica node	Disconnects the replica node from the primary node. **Note** The "in progress" broadcast contexts are rolled back.
Close and delete snapshots	Clean up a replica node delivery dataspace. To access the service, select a delivery dataspace from the 'Delivery dataspaces' table on the replica node, then follow the wizard to close and delete snapshots based on their creation dates. **Note:** The last broadcast snapshot is automatically excluded from the selection.

129.3 Supervision

The last broadcast snapshot is highlighted in the snapshot table of the dataspace, it is represented by an icon displayed in the first column.

Primary node management console

Several tables make up the management console of the primary node, located in the 'Administration' area of the primary node, under '[D3] Primary node configuration'. They are as follows:

Registered replica nodes	Replica nodes registered with the primary node. From this table, several services are available on each record.
Broadcast history	History of broadcast operations that have taken place.
Replica node registration log	History of initialization operations that have taken place.
Detailed history	History of archive deliveries that have taken place. The list of associated delivery archives can be accessed from the tables 'Broadcast history' and 'Initialization history' using selection nodes.

Primary node supervision services

Available in the 'Administration' area of the primary node under '[D3] Primary node configuration'. The services are as follows:

Check replica node information	Lists the replica nodes and related information, such as the replica node's state, associated delivery profiles, and delivered snapshots.
Clear history content	Deletes all records in all history tables, such as 'Broadcast history', 'Replica node registration log' and 'Detailed history'.

Replica node monitoring through the Java API

A replica node monitoring class can be created to implement actions that are triggered when the replica node's status switches to either 'Available' or 'Unavailable'. To do so, it must implement the `NodeMonitoring` interface. This class must be outside of any EBX® module and accessible from the class-loader of 'ebx.jar' and its full class name must be specified under '[D3] Replica node configuration'.

See also *`NodeMonitoring`*[API]

Primary node notification

A D3 administrator can set up mail notifications to receive broadcast events:

- On broadcast failure,
- On federation broadcast, if replays exceed a given threshold.

The mail contains a table of events with optional links to further details.

To enable notifications, open the '[D3] Primary node configuration' dataspace from the 'Administration' area and configure the 'Notifications' group under 'Global configuration'.

The 'From email' and 'URL definition' options should also be configured by using the 'Email configuration' link.

Log supervision

The technical supervision can be done through the log category 'ebx.d3', declared in the EBX® main configuration file. For example:

```
ebx.log4j.category.log.d3= INFO, Console, ebxFile:d3
```

See also *Configuring the EBX® logs* *[p 582]*

Temporary files

Some temporary files, such as exchanged archives, SOAP messages, broadcast queue, (...), are created and written to the EBX® temporary directory. This location is defined in the EBX® main configuration file:

```
################################################
```

```
## Directories for temporary resources.
################################################
# When set, allows specifying a directory for temporary files different from java.io.tmpdir.
# Default value is java.io.tmpdir
ebx.temp.directory = \\${java.io.tmpdir}

# Allows specifying the directory containing temporary files for cache.
# If unset, the used directory is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.cache.directory = ${ebx.temp.directory}/ebx.platform

# When set, allows specifying the directory containing temporary files for import.
# If unset, the used directory is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.import.directory = ${ebx.temp.directory}/ebx.platform
```

CHAPTER 130

Staging

This chapter contains the following topics:

1. Overview
2. Glossary
3. Performing staging
4. Staging domains and best practices
5. Staging permissions and administrative delegation
6. Error handling
7. Supported types of staging elements
8. REST API
9. Temporary folders handling
10. Automatic import on startup

130.1 Overview

The staging feature allows you to move TIBCO EBX® repository elements from one environment to another. For example, it can stage configuration settings from a development environment to a testing environment. Many elements are supported as part of the staging feature. However, its main purpose is to handle configurable elements contained in the repository and *not* data that are updated by end users. See Supported types of staging elements [p 784] for a list of all supported types of staging elements.

> **Note**
>
> If a group has no elements to stage, it does not display. For example, **Historization profile** does not display if no custom historization profile has been created.

By default, the staging features are active. Use the `ebx.staging.activated` [p 589] parameter contained in the `ebx.properties` file to activate or to deactivate staging in different environments.

> **Note**
>
> Administrative access, or delegated staging access is required to use the staging feature. See Staging permissions and administrative delegation [p 781] for more information.

130.2 Glossary

Staging server

An environment composed of deployed software (including EBX®) that is dedicated to development, testing, or production. Staging is used to build, test, and review an updated, or new solution before it is moved (staged) from one server to another.

Staging element

A configuration element in an EBX® repository that is stageable. Different types of staging elements include: data model publications, workflow model publications, dataspaces, etc.

Staging domain

A set of staging elements that can be transferred between EBX® servers.

Staging archive

File used to stage elements of a domain from a server to another.

Staging operations

The actions of importing and exporting a staging archive to and from a server.

130.3 Performing staging

The process of staging begins with the creation of domains to organize and stage elements. Once a domain is ready, it is exported as an archive file. Finally, the archive is imported into the target environment.

This operation can be performed in the EBX® graphical user interface, or by using the REST APIs [p 1185]. See Staging domains and best practices [p 781] for staging domain best practices.

> **Attention**
> Only product versions with the same minor number are compatible for staging.

The following steps show how to use the user interface to perform a staging operation:

1. Create a domain:

   See also *Staging domains and best practices [p 781]*

   1. In the source repository, navigate to *Administration* > *Repository management* and select **Staging**.
   2. Select **Domains** and click **Add a domain**.
   3. Provide a name and save, but do not close.

4. Define the elements to stage by selecting the desired elements and if required, the specific content in the category. The list of selections is automatically saved on changes.

   > **Note**
   >
   > Any elements that are in a state of error are hidden and cannot be selected.

   > **Note**
   >
   > Additionally, each node in the repository is limited to displaying 1000 elements. It is possible to change this default value using the `ebx.staging.maxElementPerList` property contained in the `ebx.properties` file.

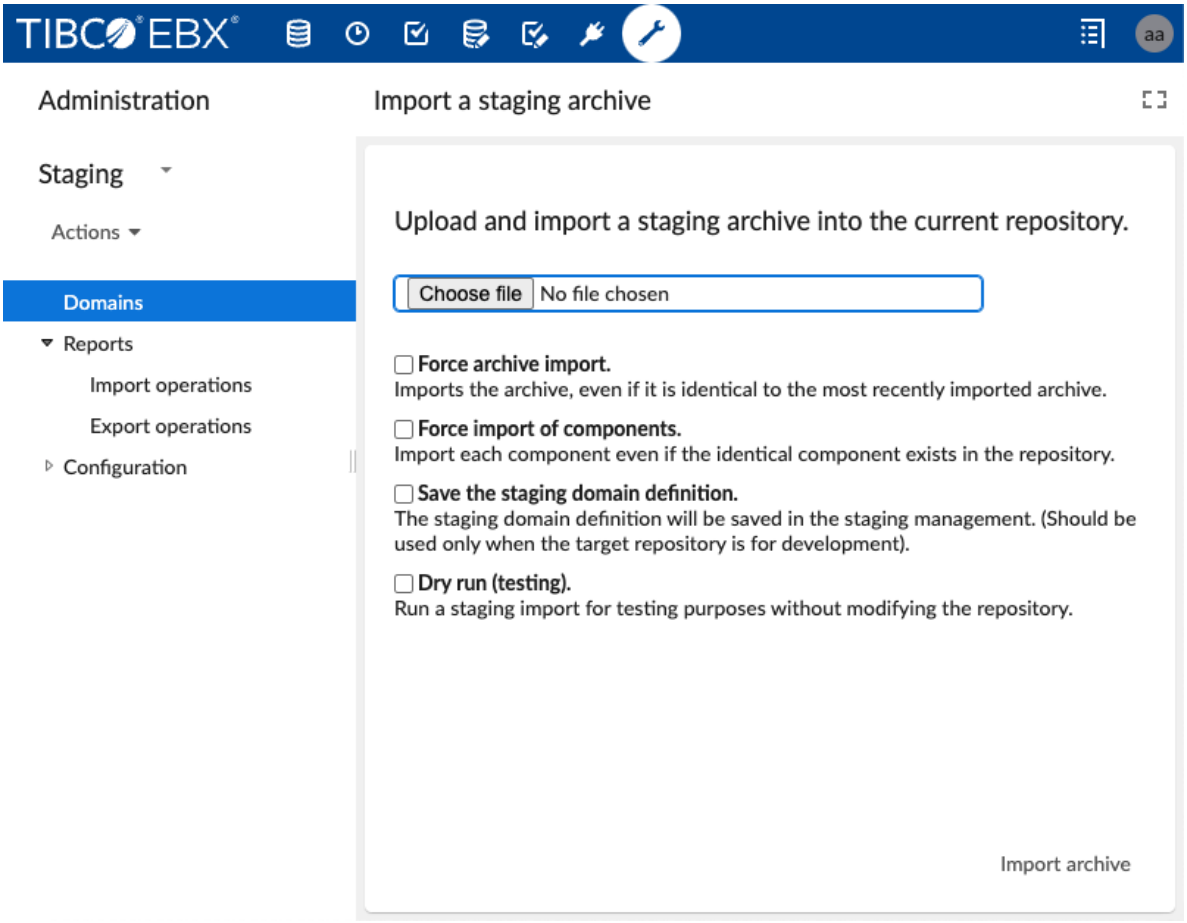
   See also *Activating and configuring staging* *[p 589]*

2. Export and download the staging archive by selecting **Download** at the top of the screen or at each line of the domains table.
3. Import the staging archive:
   1. In the target repository, navigate to *Administration* > *Repository management*and select **Staging**.
   2. Click **Import a staging archive** at the top of the screen.
   3. Browse to select the staging archive.
   4. Choose one of the import options. See Specifying import behavior [p 780] for more information.
   5. Click **Import archive**.

      See Error handling [p 782] for more information.

Specifying import behavior

The following image and table highlight available options when importing a staging archive:



Force archive import	If the same staging archive was already successfully imported, the staging import is ignored by default. Use this option to force import execution of the archive. Use this mode if an initial import did not go as expected, then retry after making changes in the source repository.
Force import of components	If the same staging component was already successfully imported, it is ignored by default. Use this option to force import execution of the component. Use this mode after modifying an archive and force replay the entire persistence process before going into production. **Note** If the same archive has already been successfully imported, the option `Force archive import` must also be checked to force

	a component import. Otherwise, the entire import process will be skipped.
Save the staging domain definition	By default, staging domain records are not created in the **Domains** table when executing an import operation. Use this option to import manually the domain definition.
Dry run (testing)	This option is used for testing purposes to run an import without modifying the repository. Use this option to ensure the existence of declared dependencies.

130.4 Staging domains and best practices

Staging domains are an effective means to organize the repository configuration settings. Use a domain to break large repository configurations into smaller parts. This allows each part to have different life-cycles. For example, HR and Procurement applications require updates at different times. In this case, create a separate domain for each application. Then define and stage these domains in multiple environments (dev, test, prod, etc.). This method also avoids having to test the entire configuration repository each time.

Domains require a unique name that conforms to dot notation. Supported characters include alpha and numeric from ASCII, underscores, dashes, and dots (the name cannot begin with an underscore). This behavior can be used like a namespace to create a hierarchy. For example: `com.acme.hr`, `com.acme.procurement`, `com.acme.procurement.catalog`, `com.acme.procurement.catalog.2022`.

Some best practices include:

- Begin to clearly define domains as soon as possible in the staging life-cycle.
- Use a separate domain for permissions because development environment permissions are usually different from production permissions.
- Staging should only handle configuration data in the repository. The types "Data tables" and "Filtered data" are supported, but it should apply to data that are not updated by users (regions, countries, etc.) as the staging data will update them and might override changes by users.
- Dataspaces that include long-duration transactional data hold operational data, and not configuration data. They can be numerous and clutter up view of staging components. Prevent them from displaying using the settings available in: *Administration > Repository management > Staging > Configuration > Hidden dataspaces*.
- Create a backup of the database before importing one or more staging archives.

130.5 Staging permissions and administrative delegation

By default, administrators in EBX® have permission to access staging and can grant access rights to non-administrative users. Access rights can specify availability of individual actions, or all actions. However, since staging has the ability to create and update configuration elements, it is important to use caution when granting access to staging.

Grant a user staging administration permissions using administrative delegation. This process is completed using User permissions apply when selecting the content of a staging domain; they cannot select elements to stage that they don't have access to.

You can configure administrative delegation in the **Administration** panel under *User interface* > *Global permissions* *[p 671]*.

See also *Configuring the user and roles directory* *[p 575]*

The following table highlights the permission differences between an administrator and a user with staging administration delegation:

Administrator	• Can export a staging archive. • Can import a staging archive.
A user with delegated staging administration	• Can export a staging archive. • **Cannot** import a staging archive. **Note** This type of user has access to, and can export domains created by other users or administrators even if the user does not have access to the staging elements. Choose this option when you want to ensure the consistency of the domain definition. Only users with write permissions can create domains.

130.6 Error handling

Error handling during import operation

An import operation may return errors and warnings. The import can be successful, partially successful, on failure, have elements that were ignored or totally ignored. The import operation does not stop on first error, but tries all components. When an error occurs, the component will not import correctly. It might require a change in either the source or target repository to complete successfully. The error message should provide enough information to help discern the issue.

Note

Importing is not fully transactional and errors might lead to an unstable repository.

A dependency check is done, and if a required element is missing, an error is returned. This can happen when importing a domain that contains a dataset and the dataspace associated with the dataset is not in the domain, or in the destination repository.

When a warning occurs, the component has probably been imported correctly but might work in a degraded way, or is not accessible. During the dependency check, if a referenced optional element is missing, a warning is returned. This is the case of the owner profile of a dataspace.

> **Note**
>
> Staging tries to avoid writing as much as possible by default. This behavior can be overridden to force create or update staging elements. See Specifying import behavior [p 780] for more information.

Import report

An import report is generated on each import operation performed through the UI or REST. It includes the import status of the domain and its components. The report also contains messages with severity levels (`info`, `warning`, `error`) that can help troubleshoot issues.

Import domain status values:

- **Success:** All components are imported successfully. The import might ignore some components if they are already present and as expected in the destination repository.
- **Partially failed:** Some component imports raised errors. Other components were imported successfully. The Status of each component [p 783] import should be checked.
- **Failed:** An exception occurred while reading the archive, or the component imports are all in error. Unlike the "Partially failed" status, no changes are performed to the repository.
- **Ignored:** The archive is ignored as it was already imported or all the component imports were ignored. No changes are performed to the repository.
- **Waiting:** Used in asynchronous mode. The import operation is queued and waiting to be processed.
- **Started:** The import process is started and not finished.

> **Note**
>
> On system start, all imports that were started or waiting will pass automatically to failed because the job was interrupted.

Import component status values:

- **Success:** The staging element was successfully created or updated.
- **Failed:** An error occurred while importing the staging element.
- **Ignored:** The import is ignored since the element is as expected in the repository.
- **Started:** The import is started and not finished.

Error handling during export operation

Unlike import operations, the export operation only succeeds on full archive export; it fails if any error occurs. There is no partial states for staging exports.

If components in the domain are missing, an error is returned and the archive is not generated.

Export report

An export report is a report generated on each export operation performed through the UI or REST. It includes the export status of the archive.

Export domain status values:

- **Exported:** The archive was successfully exported.
- **Requested:** Used in asynchronous mode. The export is requested and currently processing.
- **Aborted:** The archive export was aborted due to export errors or a processing interruption.
- **Expired:** Using the REST service's asynchronous mode, an archive was generated but expired. It is no longer available.

130.7 Supported types of staging elements

The following types of repository elements are supported for the staging feature:

- Dataspace
  - Dataspace permissions
  - Dataset (see Datasets [p 785] for more information)
    - Dataset permissions
    - Data table
      - Filtered data
- Data model (see Data models [p 785] for more information)
  - Data model permissions
- Data model publication (see Data models [p 785] for more information)
- Workflow publication
- Perspective
  - Perspective permissions
- Profile (Roles) (see Profiles [p 785] for more information)
- View publication
  - View publication permissions
- Administration
  - Recommended perspective
  - Recommended perspective permissions
  - Historization profile
  - History and logs permissions
  - Task scheduler permissions
  - Workflow execution permissions
  - Workflow history permissions
  - Workflow interaction permissions

    - Workflow launcher permissions
    - Workflow technical configuration
    - Dataspace administration permissions
    - Data modeling permissions
    - Auto increment permissions
- Global permissions
- Scheduled tasks
- Workflow models
- Script publications
- Script sources

Data tables

Data tables and filtered data are imported with a default configuration:

- The default import mode [p 153] is update or insert
- The blocking constraints [p 945] are disabled
- The by delta [p 154] mode is enabled
- The table triggers [p 952] are activated on import

Profiles

Only roles can be staged. Profiles are ignored when importing a staging archive, except for roles and if the directory is the default directory. When exporting a staging domain, a staging element linked to one or more profiles (user or role) will be staged with the link(s) whenever the users or roles are in the directory or not.

> **Note**
>
> In the case of a cycle in role relationships, all roles constituting the cycle are ignored

Data models

Data model publications and data models that are located in modules are not available for staging. However, a data model can be staged even if its corresponding publication is located in a module (the publication will not be staged).

Datasets

It's not possible to stage child datasets. They don't display when adding staging element to a domain.

EBX® Add-ons

EBX® Add-ons have specific components that can be staged. The sections below give information on the add-on components available to stage and where to find the components in the user interface. For reference purposes, the Staging location column in each table below refers to the following location: *Administration* > *Repository management* > *Staging* > *Domains* > *<<SelectedDomain>>* > *Components*.

The add-ons that include stageable components are:

- TIBCO EBX® Data Model and Data Visualization Add-on [p 786]
- TIBCO EBX® Data Exchange Add-on (New) [p 787]
- TIBCO EBX® Insight Add-on (New) [p 787]
- TIBCO EBX® Digital Asset Manager (New) Add-on [p 787]
- TIBCO EBX® Match and Merge Add-on [p 788]

TIBCO EBX® Data Model and Data Visualization Add-on

The table below shows the add-on's components available to stage and their location:

Add-on component	Staging location (beginning with category)
Saved graph via DMA	*Data models > Saved graph via DMA: [Name]*
Value and relationship graph configurations	*Dataspace > Datasets > Value and relationship graph configuration: [Name]*
Data model graph configurations via DMA	*Data models > Data model graph configurations via DMA: [Name]*
Data model graph configurations	Dataspace > Datasets > Data model graph configuration: [Name]
Saved graphs	Dataspace > Datasets > Saved graph: [Name]
Custom graph	*Custom graph: [Name]*

TIBCO EBX® Data Exchange Add-on (New)

The table below shows the add-on's components available to stage and their location:

Add-on component	Staging location (beginning with category)
SQL configuration	*Dataspace > Datasets > Database and data model connections* **Note** The database configuration is automatically migrated for the selected **Database and data model connection** configuration.
User template	*Dataspace > Datasets > Template information* **Note** If a template is used for an SQL service, the corresponding SQL service is automatically included.

TIBCO EBX® Insight Add-on (New)

The table below shows the add-on's components available to stage and their location:

Add-on component	Staging location (beginning with category)
Dashboards	*Dashboards* **Note** Each dashboard has dependencies to the specific assets it contains, such as datasets, tables, and fields. Empty dashboards and invalid indicators are automatically excluded.

TIBCO EBX® Digital Asset Manager (New) Add-on

The table below shows the add-on's components available to stage and their location:

Add-on component	Staging location (beginning with category)
Digital asset component	*Dataspace > Datasets > Digital assets components*
Digital assets	*Dataspace > Datasets > Digital assets in Drives* **Note** Staging can only migrate the Digital asset. Physical files must be migrated separately

TIBCO EBX® Match and Merge Add-on

The table below shows the add-on's components available to stage and their location:

Add-on component	Staging location (beginning with category)
Matching configuration	*Data model publications* > *Matching configuration* > *Table* > *Matching configuration* **Note** All elements that belong to a given matching configuration (main information, matching policies, merging policies and replications) are also migrated.
Matching policy	*Data model publications* > *Matching configuration* > *Table* > *Matching policy* **Note** All components related a given matching policy are migrated, including a merge policy if it is linked with the selected matching policy. The exception is that the matching configuration's main information is not included.
Merge policy	*Data model publications* > *Matching configuration* > *Table* > *Merge policy* **Note** All components related to a given merge policy are also migrated.

130.8 REST API

See Staging operations [p 1185] for more information.

130.9 Temporary folders handling

Temporary folders are created to support staging operations and asynchronous process. Each folder created expires after a configured delay. The scheduled task *DeleteExpiredStagingTemporaryFolders* ensures periodic deletion of expired folders.

> **Note**
>
> The dedicated size on disk of temporary folders created for staging can be configured through the property `ebx.staging.maxTemporaryFolderSizeInBytes` [p 589].

130.10 Automatic import on startup

You can configure staging to automatically import specific archives on startup. This prevents you from having to manually initiate the import process and can ensure that all necessary data is available in your environment in a timely manner.

To set up automatic staging archive import:

1. In the `ebx.properties` file configure the `ebx.staging.importOnStartup` [p 589] property.

2. Declare the folders containing the archives you want to import:
    1. The folders should be physically located in a `/startup` directory under the configured repository directory [p 574] (example: `${ebx.home}/ebxRepository/`**`startup`**`/folder1`).
    2. Each folder should contain a JSON file called `instructions.json`, which specifies the order and options of import for the archives in that folder.

The `instructions.json` file has the following format:

```
{
  "imports": [
    {
      "archiveFilePath": "myArchive1.zip"
    },
    {
      "archiveFilePath": "myArchive2.zip",
      "advanced_options": {
        "forceUpdateOnIdenticalComponent": false,
        "importDefinition": true
      }
    }
  ]
}
```

If an import fails, an `errors.json` file is created in the folder and contains a detailed report of the errors encountered during the import process. If the import is successful, an `output.json` file is generated in the folder and provides a report of the successfully processed imports.

> **Note**
>
> The advanced options are optional and lead to the same behaviour as the import options in the UI: Force import of components and Save the staging domain definition [p 780].

> **Note**
>
> Any folder containing the file `output.json` will be ignored during startup since it will be considered as already successfully imported.

> **Note**
>
> The declared folders and the archives specified in the instruction files must be strictly located under the startup folder of the configured repository. Navigation to parent folders, as well as the use of symbolic links, is not allowed.

CHAPTER 131

New user interface preview

This chapter contains the following topics:

1. Overview
2. Enabling the new UI
3. Accessing the new UI
4. General navigation principles
5. Glossary

131.1 Overview

The EBX® UI is undergoing major revision. The objective is to provide a modern interface that provides an intuitive and efficient navigation experience. Beginning with this release, a preview of the new UI is included. The new navigation model can coexist with the current UI.

> **Note**
>
> Use of the new UI is optional. It can be evaluated alongside the existing UI, which remains the recommended interface for production use.

Current limitations

The available content and functional scope of the UI preview are intentionally limited. Additional capabilities will be introduced in future updates.

> **Attention**
>
> As this is a preview, all functionality, UI components, and labels are subject to change in future releases.

Submitting feedback

Feedback on any aspect of EBX® is always welcome. This is especially true for the UI preview. Use the special *UI > UI Preview* category in the EBX® Ideas Portal to submit any feedback.

131.2 Enabling the new UI

The preview of the new UI must be enabled by an administrator. When you enable the new UI:

- It is available to all users.
- Does not affect existing user interfaces, such as perspectives, with the exception of the home page.
- The preview operates alongside the current UI and does not replace it.

To enable the new UI preview:

- Deploy the `ebx-hub.war` file alongside `ebx.jar`.
- In your `ebx.properties` file, set the property `ebx.internal.hub.unrestricted=true`. Note that when new UI module is deployed and this property is not set, only Home is available. The other new UI components and features are not.

131.3 Accessing the new UI

After deploying and enabling the new UI, you can access it:

- From the **Perspectives** menu, using the **Homepage** option. If the new UI's artifacts are not deployed, the **Homepage** option takes you to the default homepage.
- By appending the following to your EBX® URL: `/ebx-hub`.

See the instructions for end users [p 27] for more detailed information on interacting with the new interface.

131.4 General navigation principles

The new UI is composed of three primary components: the Dock, Workspace, and Cards. These components define how users access functionality and navigate data. From an administration perspective, user-facing functionality is exposed through services, which appear as cards in the workspace.

The Dock is the persistent vertical area on the left side of the screen. Its main purpose is to enable navigation between a user's **Home**, available perspectives, in-progress data operations, and open tables and records.

The Workspace is the primary area displayed to the right of the Dock. When accessing the new UI, the Workspace defaults to the user's **Home**. When you navigate using the Dock, the Workspace focus automatically updates to the selected view or component.

Services display for users as Cards and are organized into multiple columns. Each column represents a navigation level, such as a dataspace, dataset, table, or record and provides context for what you are viewing or working on. As you navigate deeper, new columns open to the right, while previous columns remain visible to preserve context.

The content of some cards that display on a user's **Home** is configurable. After navigating away from **Home**, the content shown in cards is determined by the perspective configuration.

131.5 Glossary

This section provides definitions for terms used in this document.

Dock	The persistent vertical bar on the left side of the screen that provides navigation to EBX® Home, configured perspectives, current data operations, open tables and records.
Workspace	The primary area displayed to the right of the dock. This is where users view, interact with, and manage data to complete their tasks.
Service	A service represents functionality that can be configured and made available to users. Services define which data and actions are accessible to users. When a service is enabled in a Perspective, it is presented to end users through one or more cards in the workspace.
Card	A card is a structured container in the workspace that presents information, actions, or tools. Cards provide a consistent way to view and interact with data across the product. Depending on permissions, cards may be editable, configurable, or read-only.
Home	The personalized landing area in the new UI. Home displays user-configurable cards such as shortcuts, drafts, and recent activity, and serves as the starting point for navigation.
Preview UI	A pre-general availability version of the EBX® user interface provided for evaluation and early adoption. Preview features may change between releases.
Perspective	A configured view of EBX® that defines which services, data, and actions are available to users based on role and context.

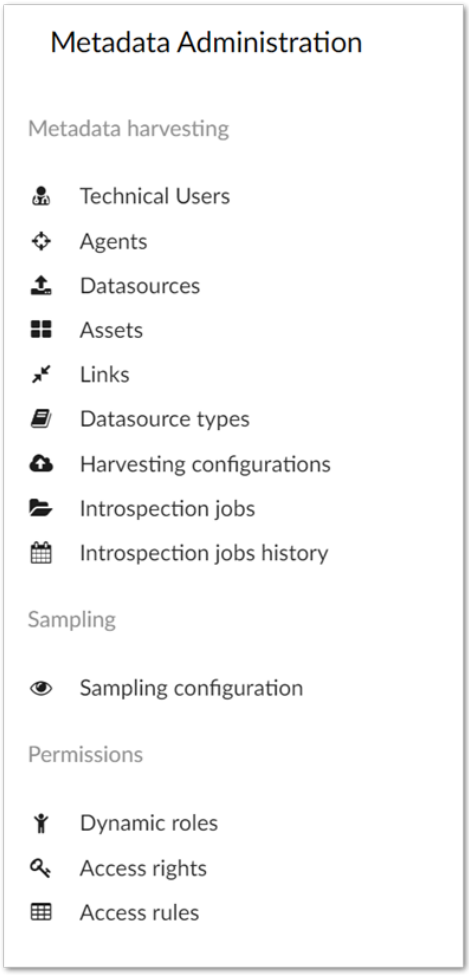
EBX® Metadata Management application

CHAPTER 132

Metadata Administration Perspective

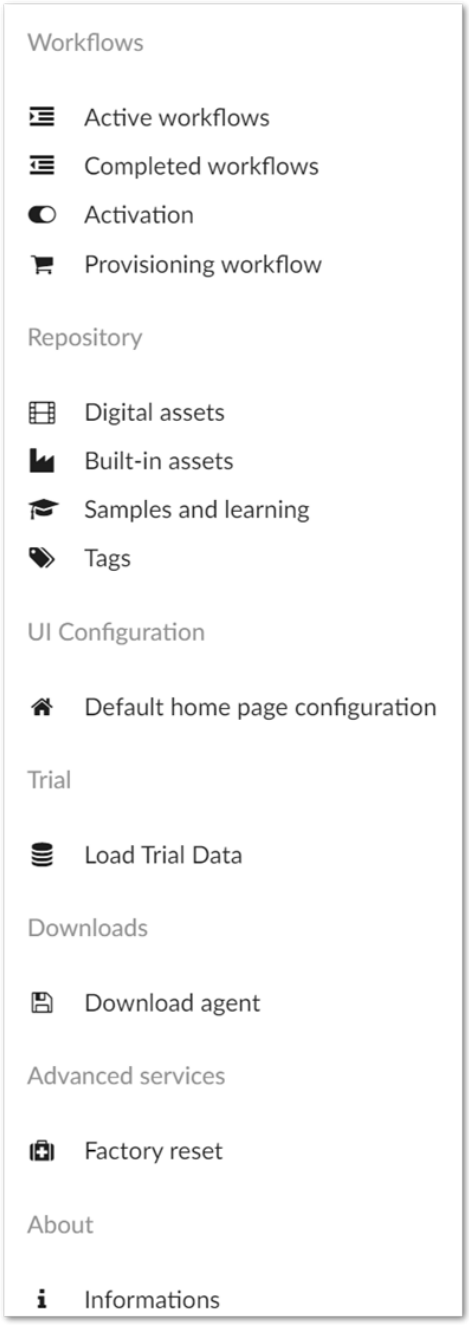
The main navigation pane, located at the left of the screen, contains perspectives. Perspectives allow you to access to all features that are enabled by permissions associated with your profile. Features are groups by section; each section is identified by its own color. You can resize the perspective by dragging is right border and reduce display to only icons by clicking on the arrow at the upper right corner.

The **Metadata Administration** perspective gives you access to the following sections.



Metadata harvesting This section allows you to configure the TIBCO® Metadata Agent. You can view a list of available agents, supported data source types and a list of different objects managed by these

agents. You can follow status and troubleshooting of metadata harvesting jobs and configurations. From here you can also create technical users for the Agent and other applications that call the Metadata API. Sampling This section allows you configuring sampling default size. Permissions This section covers administration of the application permissions for different groups of users and different types of actions and services in the EBX® Metadata Management application. You can assign roles to the users, define access rules for your metadata according to user profiles and apply dynamic roles to workflow tasks. You can also define collaboration permissions.



Workflows This section allows you to follow active and completed workflows executed in the system. It also allows you to activate and deactivate workflow execution for create and update actions, and define provisioning parameters. Repository From this section you can access documents attached to your metadata as well as sample files retrieved using Asset sampling. You can see three drives here: User data drive, Built-in drive and Machine Learning/Samples drive. UI configuration This section

allows you to define a default home page configuration and grid forms. Trial This section is only available for users subscribed to trial data. It allows you to load a trial metadata sample in your system. Downloads You can download a TIBCO® Metadata Agent ready for installation package from here. Advanced services This section allows you to reset the system's configuration to the default state. A reset applies to all permissions and views and resets them to their original state. About From here you can access information about the EBX® Metadata Management application.

CHAPTER 133

User roles

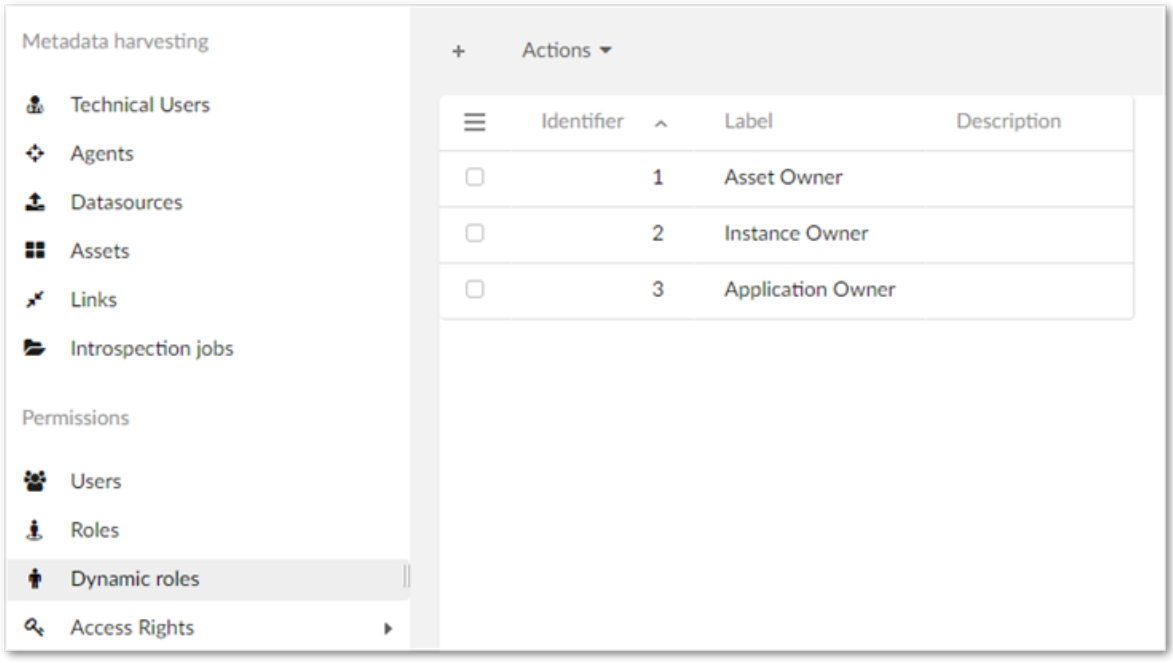
This chapter contains the following topics:

1. Dynamic roles

133.1 Dynamic roles

A role is dynamic when it is relative to data. When a static role can be given to a group of persons for all occurrences of a particular entity, a dynamic role can be given to a different group of users for each record. For instance, the owner of a given occurrence of an entity can be set as a field of this occurrence which refers to a group of persons. This group itself refers to a static role. Going through these relations we can then link an occurrence to a static role. The Dynamic role thus describes how to resolve static roles for a given occurrence based on its values.

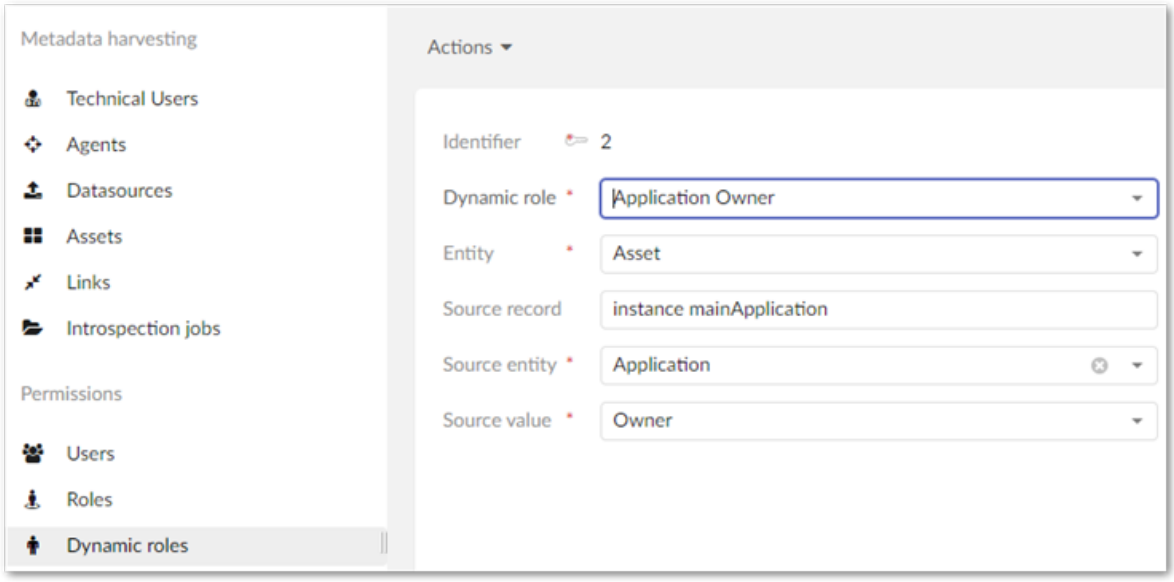
The configuration of dynamic roles is made from the **Metadata Administration** perspective.



For each dynamic role, a resolution path can be defined from entities. In the example below, the application owner can be resolved for a given Asset by getting its containing Instance and the main Application of this Instance. Now that the source record has been retrieved following the specified

path, we can get the source value (Owner here) which must be a reference to a Group. This group being linked to a static role, the dynamic role can be resolved by this static role.

The value that must fill the field **Source record** can be an empty string if the source value is directly in the current occurrence of the chosen entity. Otherwise, it should be a succession of paths to foreign keys or associations within the records we consult.



Main tab

Field Description Identifier This field is mandatory. The identifier is automatically generated. This cannot be modified. Dynamic role This field is mandatory. Define which dynamic role this plan will resolve. Entity The entity from which we want to resolve the dynamic role. An occurrence of this entity will be the starting point of the resolution. Source record List of successive paths to references separated by a space. If empty, the source record is the current occurrence of the entity defined in the previous field Entity. Source entity This field is mandatory.

CHAPTER **134**

Permissions

You can access permission configuration from the **Administration** perspective. The **Permissions** group allow you to define access rights on the different Data categories and Access rules.

Access rights

Default access rights are set on every data entity of every category according to what each role is expected to do. This is a default configuration that you can change by accessing the categories under the **Access rights** menu. By defining access rights, we specify for every element that can be an entity, one of its fields or a set of those, if it is displayed and if so, if it is editable for a given user's role. Access rights are also applicable to operations such as the creation of new records and services such as the import and export of files.

Access rules

While access rights involve user roles and data structure, access rules restrict access according to data values. You can apply them to the occurrences of an entity or to a field. If the field is left undefined, the access rule will determine the access to every occurrence. The example below determines the access to occurrences of the Application: All applications containing the word Secret in their label are hidden from business users. The condition is expressed as an XPath predicate starting at the root of the entity where every field must be seen as a step in the path. See Supported XPath syntax for more information.

CHAPTER 135

User interface configuration

This chapter contains the following topics:

1. Change default home page template

135.1 Change default home page template

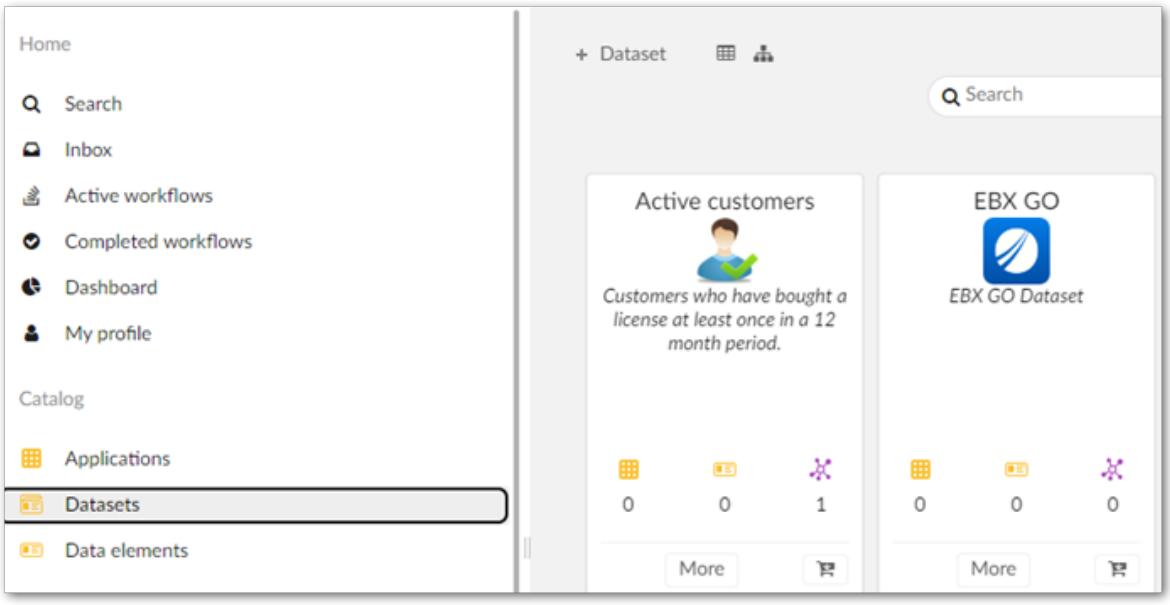
On the administration perspective, select **Default home page configuration**.

You will see the default home page template that you can customize. Click the gear icon to modification of the page. You can add, remove, edit, and rearrange the homepage widgets. To learn more about home page customization, please refer to Home page [p 335].

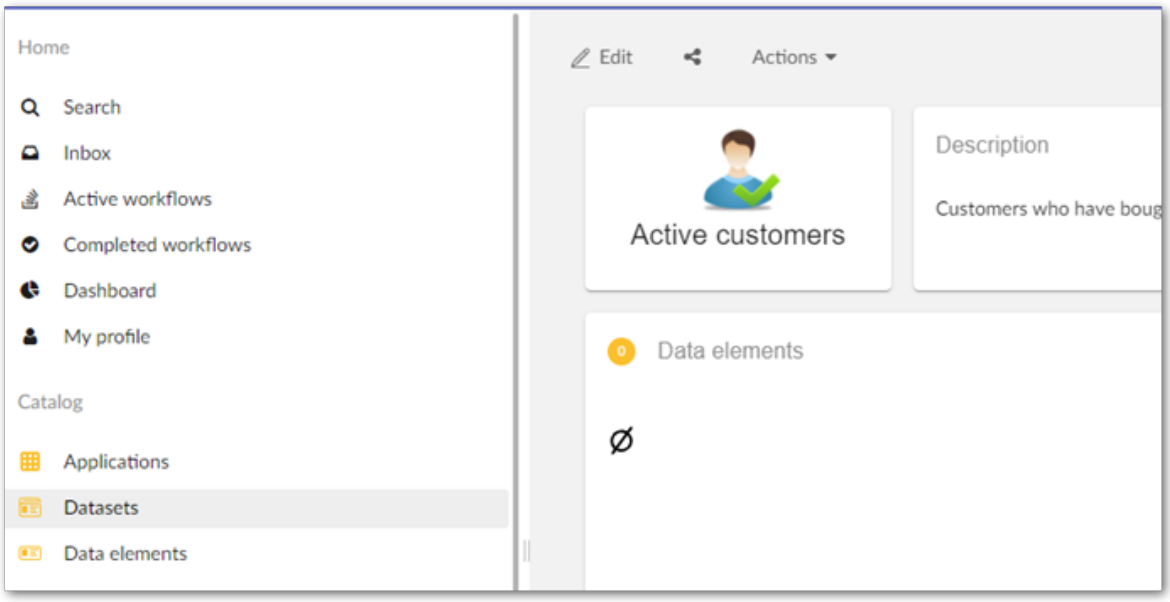
CHAPTER 136

Workflows

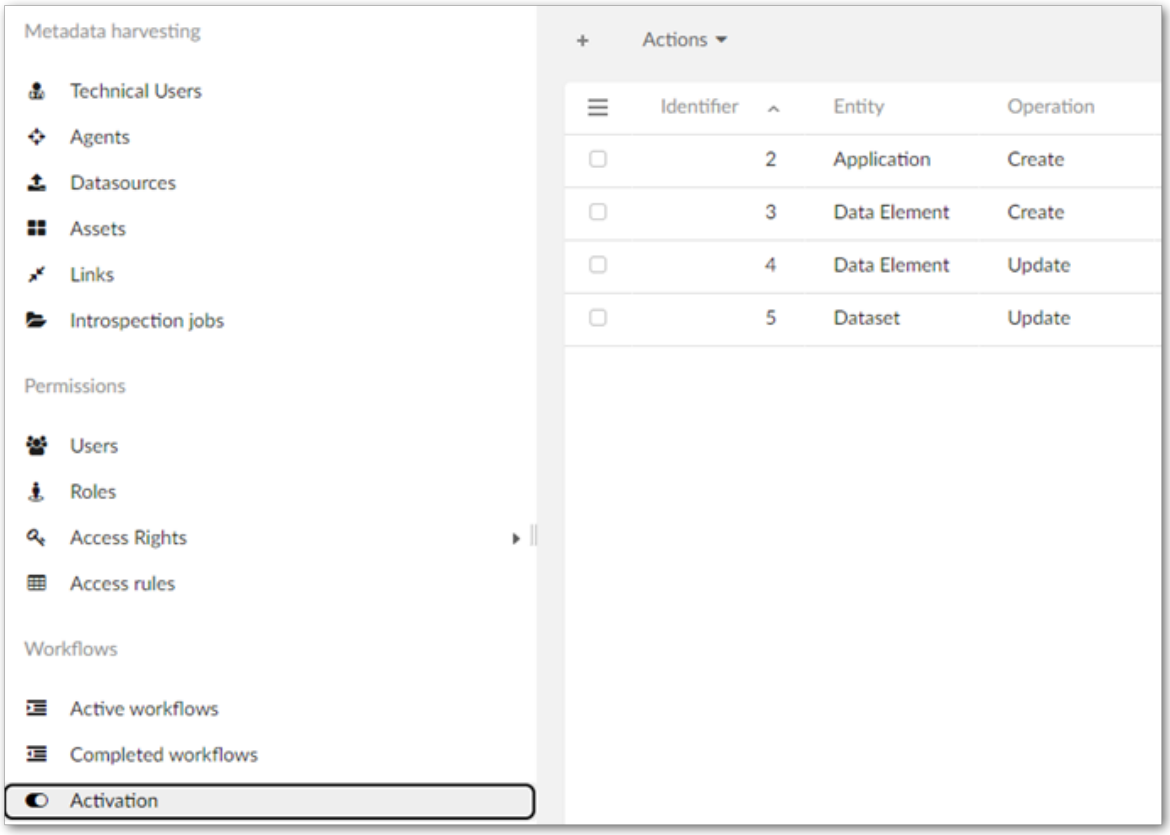
All users permitted to create a new record for an entity have access to an icon with a plus sign followed by the name of the entity. This icon is located in the upper left side of global views such as the card, table and hierarchical views.



Following the same logic, users can update a record for an entity when they see the **Edit** icon. When they select this icon, they are redirected to an edit screen.

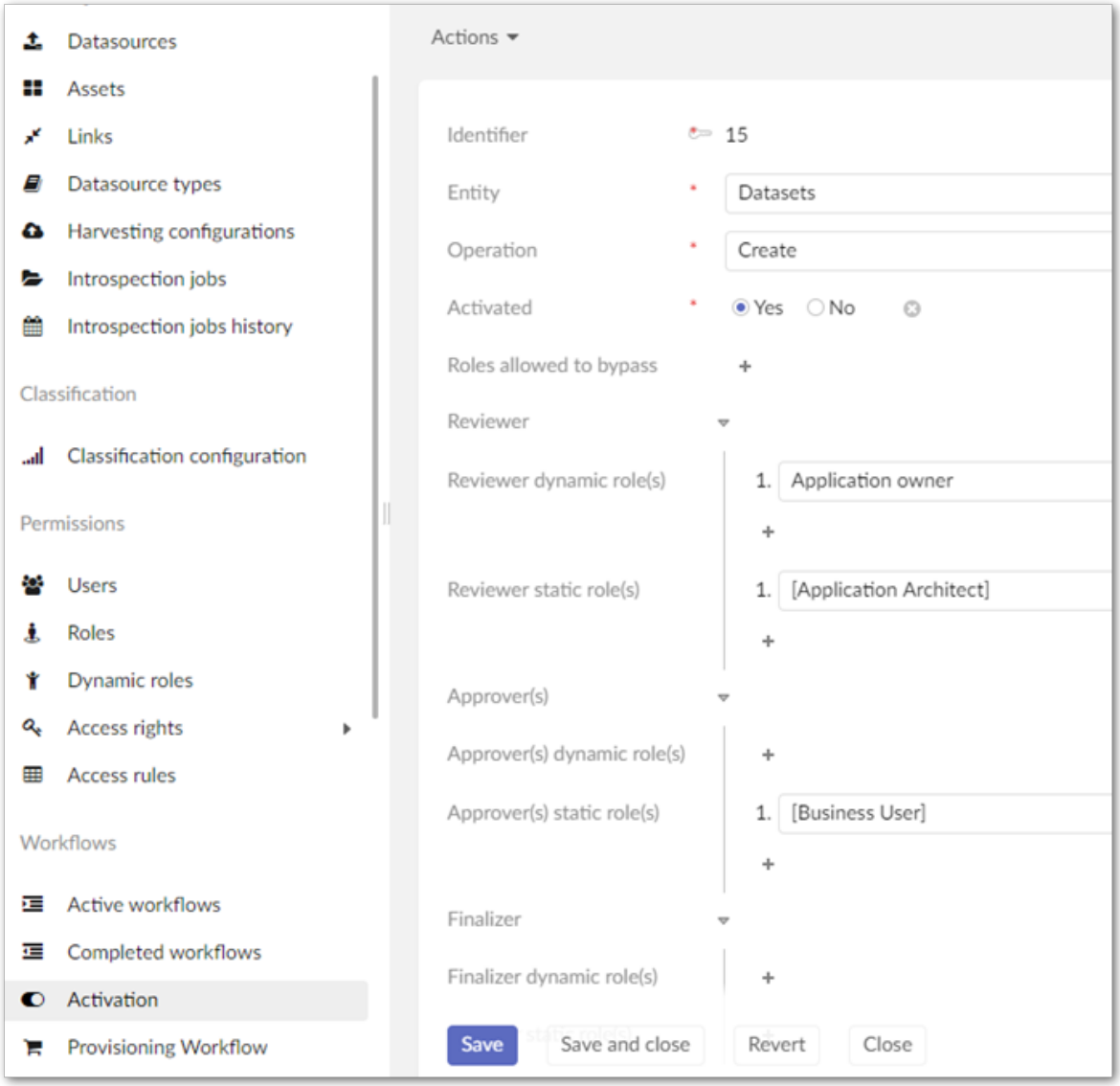


By default, these actions are free from control and users permitted to see affected records can see these changes. In order to enforce controls, you can activate workflows to involve actors that review, approve and finalize any creation or update operated by some profiles. Use the **Metadata Administration** perspective to activate workflows.



You can activate workflows for create and update operations. Duplicating a record is considered the same as creation, so if you activate the workflow for create, it will also apply for duplicating.

Activating the workflow on Create causes the creation icon to redirect the user to the first step of the workflow consisting of creating a record in a workspace. This workspace isolates all modifications operated in the workflow until its completion. The completion of the workflow can lead the creations or updates being validated and visible to all, or cause their cancellation. You can define different actors for each operation and entity.



You can define each actor as a collection of user roles. These roles are either static or dynamic. A static role is defined in the directory and directly impacts users. A dynamic role is resolved from the data and is conditional to the subject of the workflow being an occurrence of an entity. A dynamic role can be, for example, the owner of this occurrence which is set on the occurrence itself as a reference to a **Group** itself referencing a static role. A dynamic role is then a dynamic allocation of a static role contextual to data. See **Dynamic roles** for more information.

If no role is defined for an actor, or if only dynamic roles are defined but the latter are not mapped to static roles, the act of reviewing, approving or finalizing is then skipped.

The reviewer reviews the creation or update of an occurrence to make sure all required information is present and qualitative enough to be submitted to approvers. If not, he can reject the changes leaving a mandatory comment. The initiator of a workflow will receive a task to either cancel the operation or make the necessary changes according to the reviewer's comment and ask for a new review. This review, even if offered to many roles, will be operated by only one user; the one who takes the task.

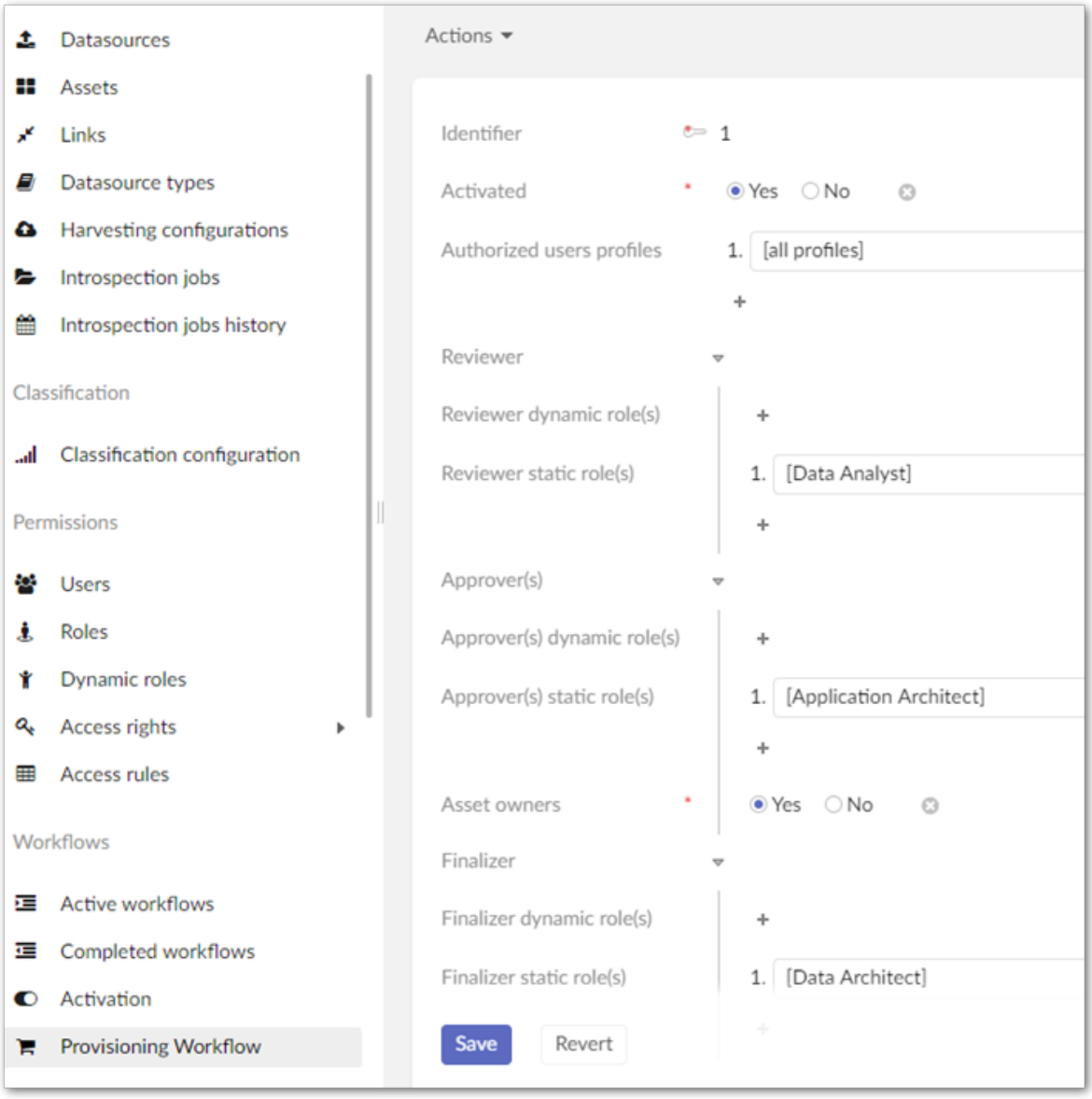
Approvers must approve or reject the operation. Offered to many roles, one user from each role must perform this action. If one of them rejects the operation, the initiator of the workflow will receive a task to either cancel the operation or make the necessary changes according to the approver's comment and ask for a new review. Once approved by all approvers, the operation can be finalized.

A user can be involved to finalize the operation if extra actions must be performed but are not authorized for previous actors.

Provisioning workflow configuration

The provisioning workflow follows the same pattern except that a third kind of approvers can be configured, the assets owners.

The finality of this workflow being the composition of a view from collected assets, each of these assets having or not a owner defined, these last can be involved to approve the creation of the view and finally the usage of their assets.



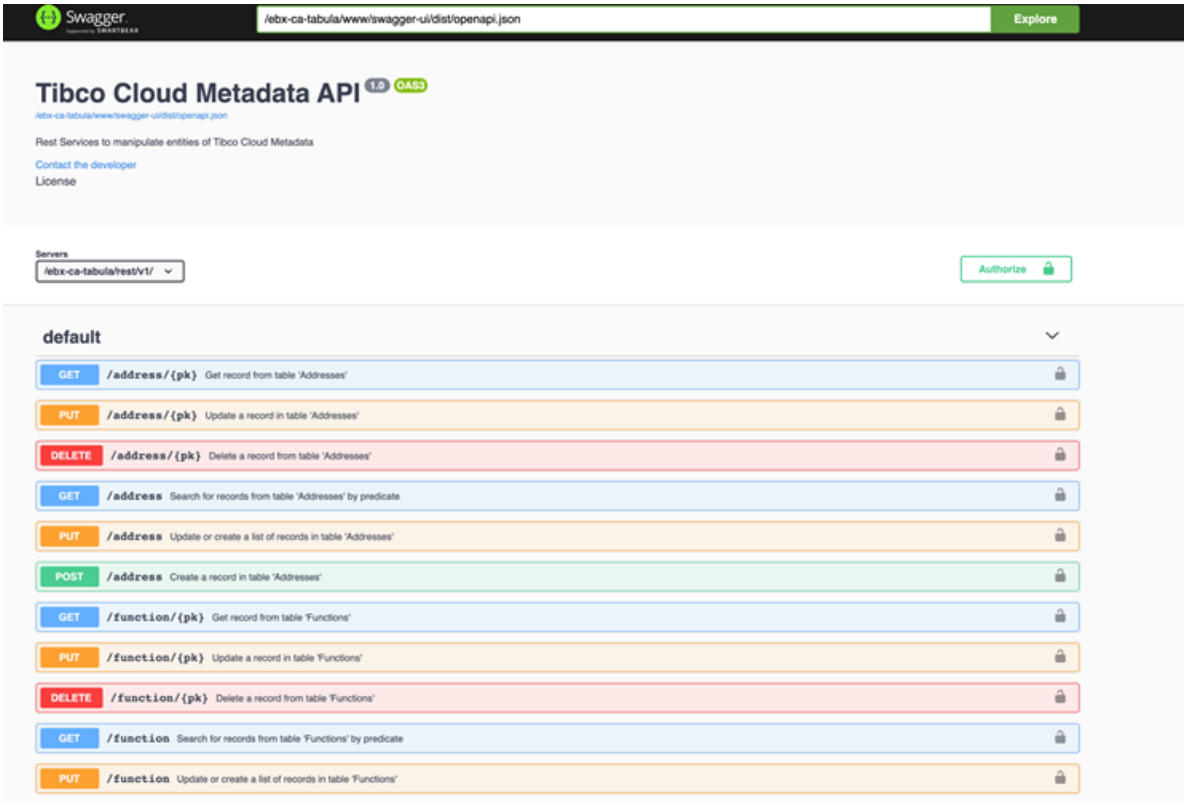
CHAPTER 137

Web Services and API

Beware: these operations are advanced ones and should be used with precautions.

A swagger Website allows using TIBCO Cloud Embedded Web Services:

From your application context root, go to `[CONTEXT-ROOT]/ebx-ca-tabula/api-documentation` in your browser.

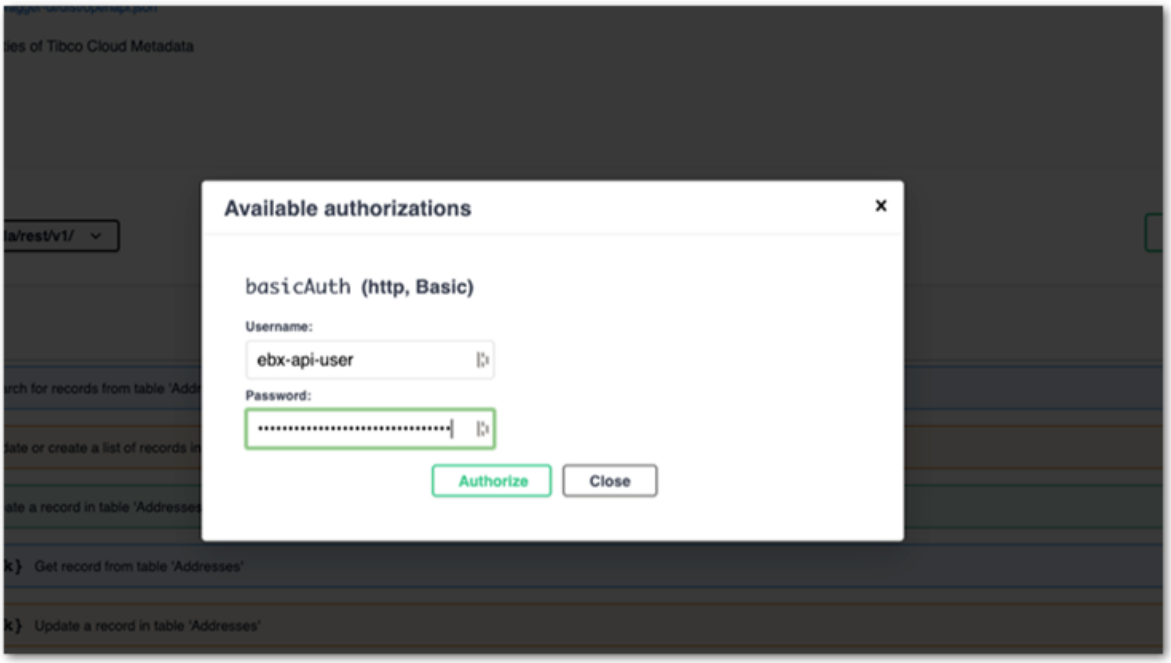


These services are not available without authentication.

To enable authentication, click on **Authorize** button and enter credentials.

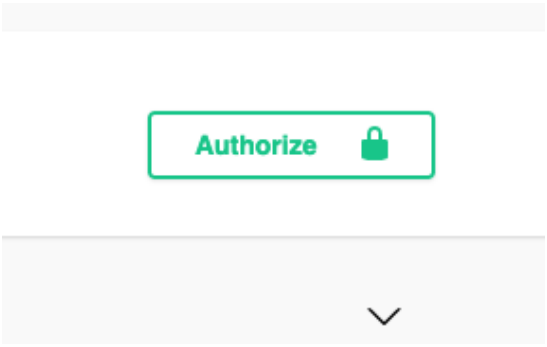
Currently, only basic authentication is available, so you'll have to use the credentials of a Technical User.

The rights applied to this technical user, including its profile(s) will be applied to API operations as well.

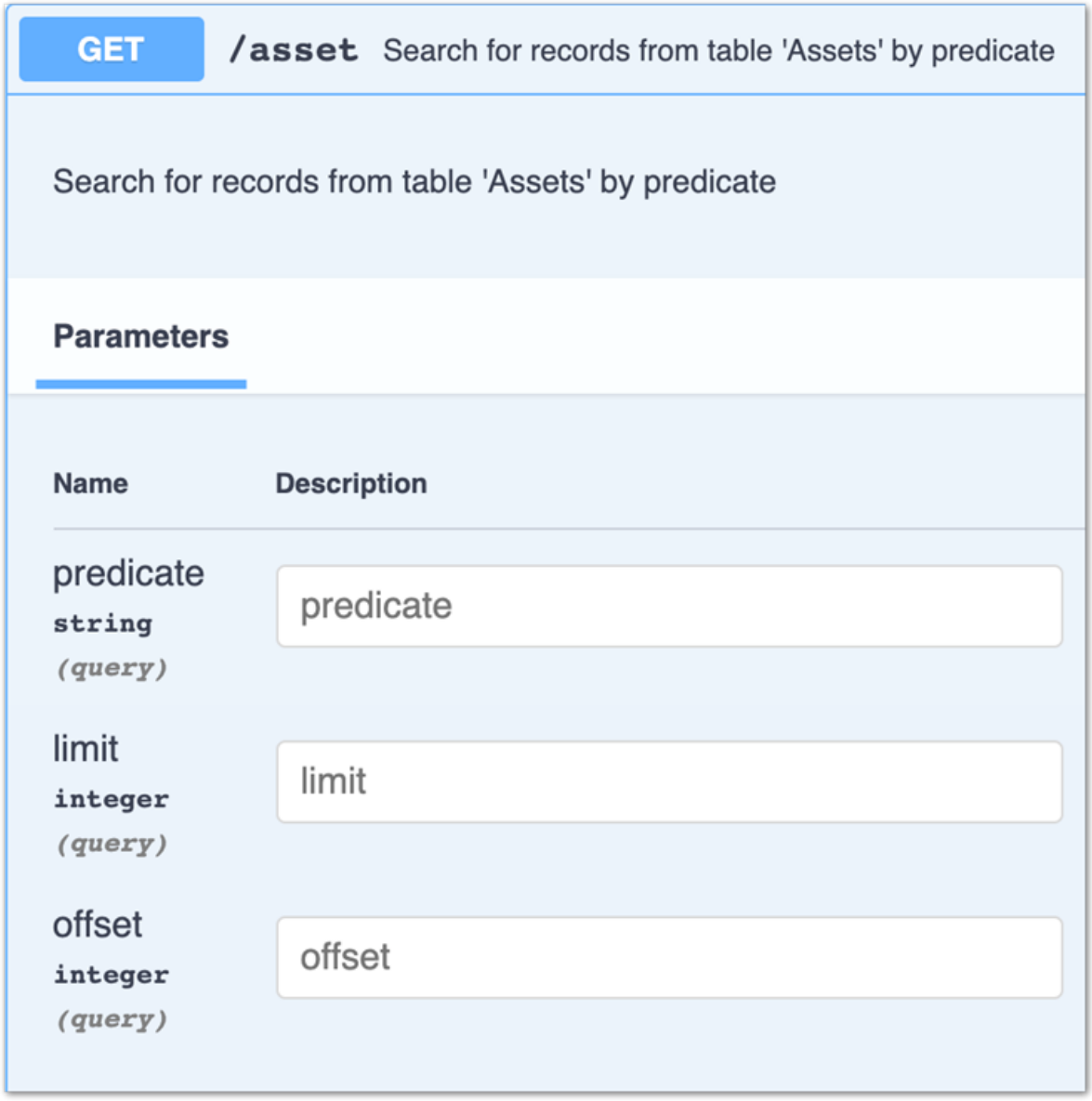


See Technical Users [p 843] to define a technical user.

Once connected, the keylock will be displayed as follows



Then you can choose the operation that you want to execute. Examples of syntax are given in the swagger interface.



When you click on select, you'll be able to get the result (HTTP Status + JSON response structure)

Responses

Curl

```
curl -X GET "http://localhost:8080/ebx-ca-tabula/rest/v1/asset" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/ebx-ca-tabula/rest/v1/asset
```

Server response

Code	Details
200	**Response body**

```
[
  {
    "assetType": "3",
    "dataElementAutoAssigned": false,
    "documentation": [],
    "id": 352213,
    "instance": "383",
    "isDisabled": false,
    "label": "public",
    "lastIntrospectionDate": "2020-04-10T14:22:43.049",
    "logo": {
      "attachment": "E88396B5-AD13-4A33-8CAF-CF93F228D921"
    },
    "name": "public"
  },
  {
    "assetType": "1",
    "dataElementAutoAssigned": false,
    "documentation": [],
    "id": 352214,
    "instance": "383",
    "isDisabled": false,
    "label": "data_src",
    "lastIntrospectionDate": "2020-04-10T14:22:43.102",
    "logo": {
      "attachment": "17740222-EF00-4F24-91AD-B54E1685D751"
    },
    "name": "data_src",
```

Response headers

```
cache-control: no-cache
content-language: fr-FR
content-type: application/json
date: Wed, 15 Apr 2020 13:39:17 GMT
expires: -1
pragma: no-cache
transfer-encoding: chunked
```

These APIs are usable with the same authentication and rights management from any Enterprise Service Bus or with an Integration Platform.

CHAPTER 138

Installing the TIBCO® Metadata Agent

This chapter contains the following topics:

1. Installation and configuration
2. EBX® Metadata Management application Connection Encryption
3. Agent execution (as a Standalone Java Application)
4. Troubleshooting

138.1 Installation and configuration

TIBCO® Metadata agent is a Java application provided as a jar file and configured using a properties file. Its main purpose is to establish interactions between the source system instance (through specific adapter) installed on a server and the EBX® Metadata Management application. One agent can be connected to one instance of a source system. The EBX® Metadata Management application supports connections to multiple agents that manage different data sources.

An agent can have one or multiple adapters to support metadata extraction from different data sources. An adapter is a separate Java application that ensures one or several of the metadata capabilities for a given data source type(s):

- Data source management
- Metadata harvesting
- Data sampling
- Metadata provisioning
- Lineage extraction.

Prior to agent configuration, you must create a technical user from the **Administration** perspective. For more details, see Technical Users [p 843].

Note: the technical user created is automatically prefixed with ebx-.

Prerequisites

The TIBCO® Metadata Agent requires:

- JRE 17 LTS minimum

If the TIBCO® TDV Metadata Adapter is used (see TIBCO TDV Metadata Adapter installation [p 825]):

- TIBCO® Data Virtualization version 8.3 minimum

Security and Cloud Access

In order to access to EBX® (CIC Cloud), and to allow the EBX® Metadata Management application to access to the TIBCO® Metadata Agent, you must take into account the following infrastructure considerations:

- TIBCO® Metadata Agent hosting server must be opened to the Internet.
- It is recommended to define a DNS Name for the TIBCO® Metadata Agent hosting server, in order to ease certificate management.
- TIBCO® Metadata Agent hosting server must be reachable via HTTPs (HTTP over SSL), using a CA-signed certificate (no self-signed certificate).
- TIBCO® Metadata Agent external port must be port 443 (default HTTPs port), using a reverse-proxy (IIS, Apache, Ngninx, …). (ma.http.ws-url has to be set to the reverse-proxied URL to the agent, reachable from Internet).
- TIBCO® Metadata Agent internal port can be configured on any available port on the machine (see ma.http.port in Agent configuration properties), and use in the reverse-proxy configuration to expose the TIBCO® Metadata Agent to Internet.

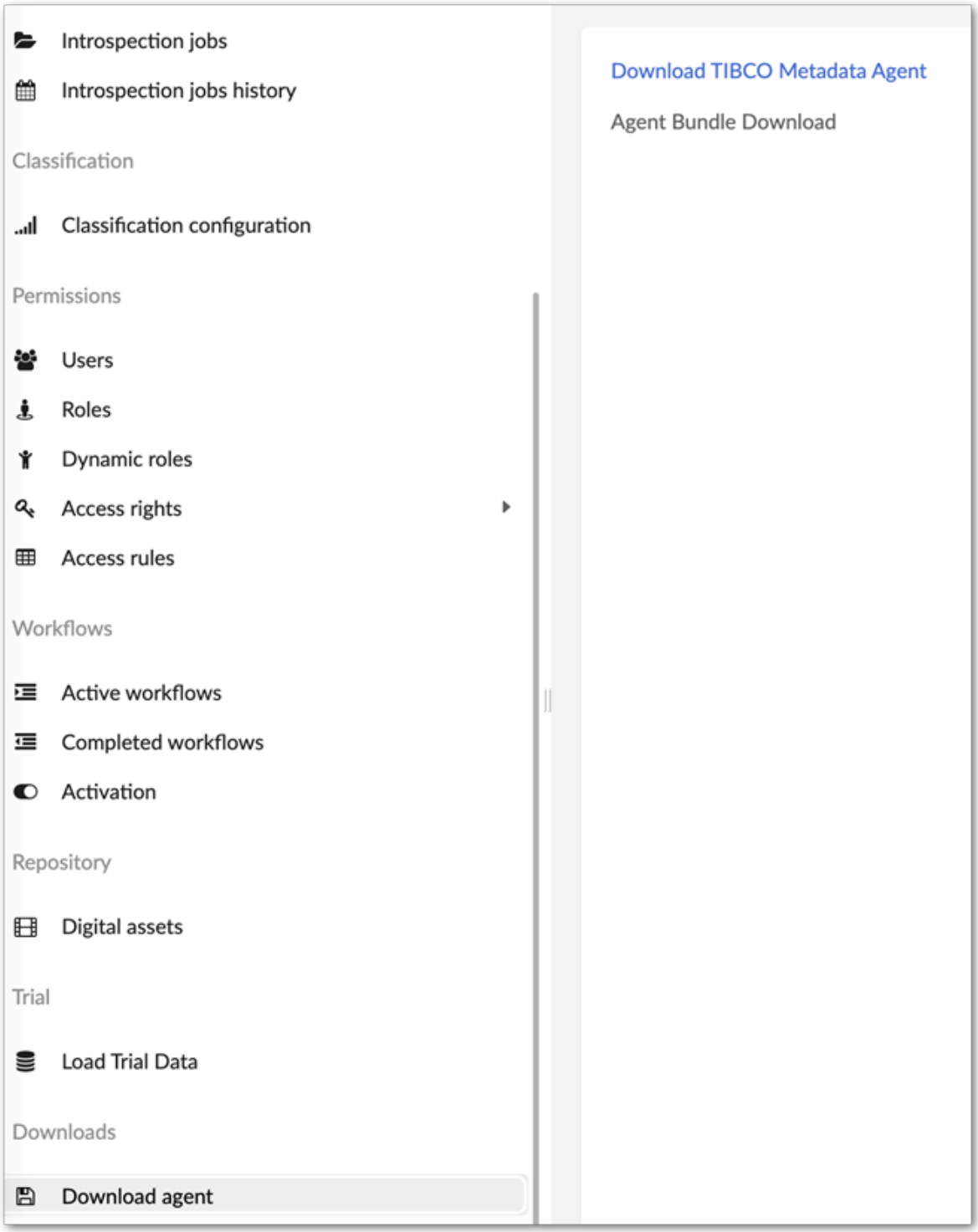
Installation process

The following steps describe how to get the TIBCO® Metadata Agent running on your system:

1. Download TIBCO® Metadata Agent package from the **Metadata Administration** perspective.
2. Unpack it on your system.
3. Fill in properties template as described below in the Properties configuration section. Please note that for some properties it is required to log into EBX® and/or obtain some information from a EBX® administrator.
4. Download and install the EBX® Metadata Management application Adapters required for your source systems.
5. Fill in properties template for each adapter.
6. Activate required adapters in the main Agent properties template.
7. Launch TIBCO® Metadata Agent as described below in the Execution section.

Packaging

The agent is available as a standalone Java application (jar file). You can download it from the **Metadata Administration** perspective.



Properties files

The main packaging of the TIBCO® Metadata Agent includes following properties files:

- application-metadata-agent.properties for configuration of Agent
- application-datasources.properties for configuration of the supported data source types per adapter
- application-tdv.properties (optional) for configuration of the TIBCO® TDV Metadata Adapter.

TIBCO® Metadata Agent properties template

```
#########################################################
#########################################################
##### ##### ##### Metadata connection ##### ##### #####
#########################################################
#########################################################
ma.metadata.context=[[METADATA BASE URL]]
ma.metadata.username=[[METADATA TECHNICAL USER NAME]]
ma.metadata.password=[[METADATA TECHNICAL USER PASSWORD MAY BE ENCRYPTED]]
ma.metadata.public-key=[[METADATA TECHNICAL USER JWT]]
ma.metadata.label=[[LABEL FOR THIS AGENT]]
####
#### Uncomment this line if you want to use encryption for agent passwords.
#### In this case, you will have to encrypt password using TIBCO Metadata Agent Tools
#### Read the documentation for more information
####
#ma.metadata.useEncryption=true

#########################################################
#########################################################
##### ##### ##### Agent properties ##### ##### #####
#########################################################
#########################################################
ma.http.port=[[AGENT PORT]]
ma.http.ws-url=[[AGENT CONTEXT URL]]

#########################################################
#########################################################
##### ##### ##### Agent log properties ##### ##### #####
#########################################################
#########################################################
#Folder where the logs will be stored
ma.logging.path=./logs
#Log level for technical logs (Spring framework, external libraries).
ma.logging.level.root=INFO
#Log level for the Agent's functional logs.
ma.logging.level.agent=INFO
#Log level for the SOAP messages. Set to TRACE to see full request body.
ma.logging.level.soap.sent=INFO
#Log level for the SOAP messages. Set to TRACE to see full response body.
ma.logging.level.soap.received=INFO

#########################################################
#########################################################
##### ##### ##### Agent proxy properties ##### ##### #####
#########################################################
#########################################################
####
#### Uncomment this line if you want to use a proxy configuration for the agent
####
#ma.proxy.useProxy=true ma.proxy.proxyHost=[[PROXY HOST]]
ma.proxy.proxyPort=[[PROXY PORT]]
ma.proxy.proxyAuthentication=[[PROXY
AUTHENTICATION ?]]
ma.proxy.proxyUsername=[[PROXY USERNAME]]
ma.proxy.proxyPassword=[[PROXY PASSWORD]]

#########################################################
#########################################################
##### ##### ##### Agent Adapter properties ##### ##### #####
#########################################################
#########################################################
#Comma-separated list of packages to scan for adapters classes
agent-adapter.base.package=com.tibco.ca.tabula.agent.tdv
#Comma separated list of profiles (corresponding to adapter technical names)
agent-adapter.active-profiles=tdv
```

Properties configuration

Prior to launching the TIBCO® Metadata Agent, you need to complete the information in the properties file to check the connection to the EBX® Metadata Management application.

Users are strongly discouraged to modify properties that are not described in the table below but are present in the properties file, especially specific URLs configured for EBX® Metadata Management application operation or the H2 database JDBC URL. A good practice is to remove old specific

properties from your prior-to-version 4 agent, in order to keep only specific properties in your configuration file.

Property name	Description
TIBCO EBX®'s metadata connection	
`ma.metadata.context`	REST Endpoint for EBX® Metadata Management application Advanced Data Services. Expected format is: `https://<host>:<port>/<contextRoot>`
`ma.metadata.username`	Metadata technical user name. Please, contact an administrator to receive this information.
`ma.metadata.password`	Metadata technical user password. Please, contact an administrator to receive this information.
`ma.metadata.public-key`	Public key (JWT) associated with the EBX® Metadata Management application technical user. Please, contact an administrator to receive this information.
`ma.metadata.label`	A user-friendly label to show in the Metadata for this agent.
`ma.metadata.useEncryption`	Define if encryption is used for `ma.tdv.datasource.password`, `ma.tdv.soap.password`, `ma.metadata.password` and `ma.proxy.proxyPassword`. If activated, all passwords must be encrypted using TIBCO Metadata Agent Tools, see below.
Agent properties	
`ma.http.port`	The port on which the agent listens for incoming REST connections.
`ma.http.ws-url`	The URL of the agent's REST endpoint. You have to set HTTPs on the agent (using reverse proxy for instance, or the EBX® Metadata Management application won't be able to connect).You also have to allow incoming connections from cloud (Internet) to agent's host & port.
Agent logging properties	
`ma.logging.path`	The folder where the agent's logs will be stored. Agent must have write permission on this folder.
`ma.logging.level.root`	Log level for the technical logs generated by the technical framework that the Agent runs on.
`ma.logging.level.agent`	Log level for the functional logs generated by the Agent.
`ma.logging.level.soap.sent`	Log level for the SOAP messages that are sent to the TDV from the Agent. By default, they are not logged. If you want to log them for debug or any other purposes, set this property to TRACE.

Property name	Description
`ma.logging.level.soap.received`	Log level for the SOAP messages that are received from the TDV to the Agent. By default, they are not logged. If you want to log them for debug or any other purposes, set this property to TRACE.
Agent proxy properties	
`ma.proxy.useProxy`	Uncomment this line if you want to use an HTTP(s) proxy for the agent.
`ma.proxy.proxyHost`	Proxy host.
`ma.proxy.proxyPort`	Proxy port
`ma.proxy.proxyAuthentication`	If authentication is needed for proxy usage, set this property to true.
`ma.proxy.proxyUsername`	Username to be used for the proxy.
`ma.proxy.proxyPassword`	Password to be used for the proxy.
Agent adapter properties	
`agent-adapter.base.package`	Comma-separated list of packages to scan for adapters classes.
`agent-adapter.active-profiles`	Comma separated list of profiles (corresponding to adapter technical names).

Once these properties are configured, you can launch the agent as a Java application, using command line or any other suitable tool (see below on this page).

The configuration file should be named `application.properties`.

Upon the first launch, the agent will query the EBX® Metadata Management application REST API for registration and will obtain a GUID to be used in all information exchanges between the two applications. Check that in the agent's log file after startup you see the message "Started MetadataAgentApplication in xx seconds." This indicates that the start up was correct and the agent is running. If you don't see this message, your need to check the log file for error messages explaining the issue preventing agent to start.

Once registered, the agent will be displayed in the EBX® **Metadata Administration** perspective as described TIBCO® Metadata Agents [p 845].

Data source types properties file

The properties file application-datasources.properties allows the agent to make connection between supported data source types, metadata management functions and the adapters to use.

Properties in this file must follow the pattern: **datasource.operation_name.data_source_type=adapter_profile** Supported operation names are as follows:

- dsmanagement
- harvesting
- sampling
- provisioning
- lineage.

Data source type name must be formatted as alphanumeric name all in lower case without any special characters. You can also define a label for data source type that will be displayed in the EBX® Metadata Management application. Only one adapter can be defined for operation - data source type.

Below is the configuration pattern:

```
datasource.label.<data source type>=
datasource.dsmanagement.<data source type>=
datasource.harvesting.<data source type>=
datasource.sampling.<data source type>=
datasource.provisioning.<data source type>=
datasource.lineage.<data source type>=
```

By default this file includes configuration for the data source types supported by TIBCO TDV Metadata Adapter installation [p 825].

138.2 EBX® Metadata Management application Connection Encryption

If `ma.metadata.useEncryption` is activated, then `ma.tdv.datasource.password`, `ma.tdv.soap.password`, `ma.metadata.password` and `ma.proxy.proxyPassword` must be encrypted into `application.properties` before starting the agent:

- As a prerequisite, `ma.metadata.public-key` must be defined in `application.properties` according to the corresponding EBX® Metadata Management application Technical User.
- Use TIBCO® Metadata Agent Tools to encrypt the original password. Launch in a terminal:

  ```
  java -jar metadata-agent-tools.jar MY_PLAIN_PASSWORD
  ```

  See the output sample below.
- Copy-paste the encrypted values into `application.properties` for each corresponding password property.

The output provided is as follows:

```
`*************************`
`Value to encrypt is MY_PLAIN_PASSWORD`
`*************************`
`Encrypted value is : I7UOEPOHtF+J4EYPTPRxR0++QG4hoXFe87uSPluI1lbi`
`*************************`
```

Caution: with this feature enabled, an encrypted password must be re-encrypted on each renewal of JWT key.

138.3 Agent execution (as a Standalone Java Application)

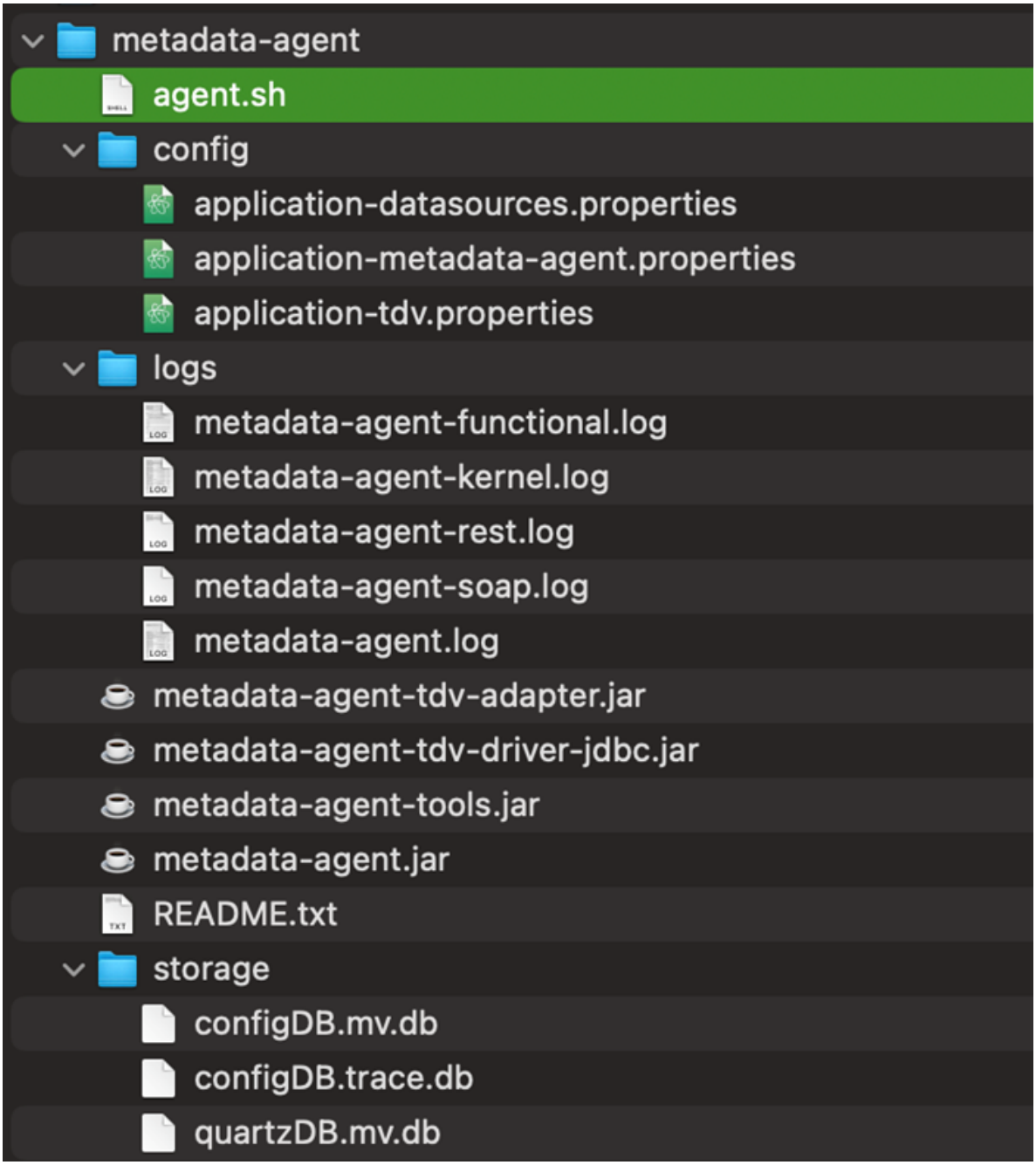
You have to configure a launcher to execute the agent, adding the configuration files into a config directory under the directory where is placed the agent.

logs and storage directories are created by the agent itself directly under the structure.

Under storage, two files are created, one for agent local database (configDB.mv.db), one for scheduling (quartzDB.mv.db). If you have previously used different names for these files in your previous configuration, please rename these files accordingly in your file system, after stopping the current agent.

To launch the agent, you can create a shell file as follows (in this example the jars are placed in a /metadata-agent directory):

You have to define the default encoding as UTF-8 for the JVM using `-Dfile.encoding=UTF-8`, if not already set.



Example on Linux (with TDV adapter activated): `java -Dloader.path=metadata-agent-tdv-adapter.jar,metadata-agent-tdv-driver-jdbc.jar -Dspring.profiles.active=tdv -jar metadata-agent.jar >> logs/metadata-agent.log 2>&1 &`

Example on Windows (with TDV adapter activated): `java -Dfile.encoding=UTF-8 -Dloader.path=metadata-agent-tdv-adapter.jar,metadata-agent-tdv-driver-jdbc.jar -Dspring.profiles.active=tdv -jar metadata-agent.jar > /logs/metadata-agent.log 2>&1`

Once started, an `application.pid` file is created containing the process id of the agent. You can stop the agent by killing this process. `kill $(cat application.pid)`

The agent can be also shut down using a HTTP POST request on a dedicated endpoint: `curl -X POST http://<agent_host>:<agent-port>/manage/shutdown`

138.4 **Troubleshooting**

The following table provides causes and resolutions to two possible errors:

Error message	Cause and resolution
Error in exchanges with the EBX® Metadata Management application: { "localizedMessage":"HTTP error 401.", "messages":["HTTP 401 Unauthorized"] }	This error appears when the agent cannot connect to the EBX® Metadata Management application using credentials provided in properties `ma.metadata.username` and `ma.metadata.password`.
Failed to connect to the EBX® Metadata Management application I/O error on GET request for "http://localhost:8081/ebx-ca-tabula-tdv/tdv-rest/health"	This error message indicates that the EBX® Metadata Management application is not responding on the `ma.metadata.context`. Check that EBX® URL is correct and it can be reached from the machine where the Agent is installed (including firewall and proxy settings)

CHAPTER 139

TIBCO TDV Metadata Adapter installation

This chapter contains the following topics:

1. Overview
2. TDV Connection rights
3. Properties template
4. Troubleshooting

139.1 Overview

Use this adapter to connect the Metadata Agent to a TIBCO® Data Virtualization (TDV) server to send its managed datasources to the EBX® Metadata Management application using the harvesting feature.

You need to install two TDV application parts:

- TDV Server - the server part of TDV. It is mandatory to install the TDV server to gather metadata from your system.
- TDV Studio - the client part of TDV. You only need to install it if you want to create your data sources and access public views from TDV locally.

Please, refer to the *Installation Guide* provided with your copy of TDV for the installation requirements and process.

The required TDV version for Metadata Agent is TDV 8.3 or higher.

Please find below an overview of the features that you may use in conjunction with the EBX® Metadata Management application.

We recommend using the corresponding TDV documentation for guidance on these features.

Create a data source

If you want the EBX® Metadata Management application to gather information about your existing data sources, you need to first configure the code in TDV. You can do it using the TDV Studio *New Data Source* function.

On the first step of creation, you will be prompted to choose a connector for your new data source.

Once you select the connector, you will be prompted to provide a label for this data source and connection details for TDV to access the physical data source on your system.

You don’t need to run the introspection on the new data source as the EBX® Metadata Management application can do it automatically. However, it is recommended to finalize the data source creation using the **Create & Introspect** button rather than **Create & Close** as this allows you to test the connection and to make sure that TDV has access to the physical data source on your system.

Once the data source is correctly configured in TDV, it becomes visible to Metadata Agent and you can run metadata harvesting for this source.

View public data sources

The Metadata Agent can create a published view of your assets and data sources by running the metadata provisioning function. This creates a public data source in TDV that you can access using TDV Studio.

Public data sources are available in TDV under the folder **Composite Data Services** that is shared by all TDV users and can be exposed to the external software via JDBC queries. The Metadata Agent creates your public data source under **Composite Data Services - Databases** with the label defined during the metadata provisioning launch.

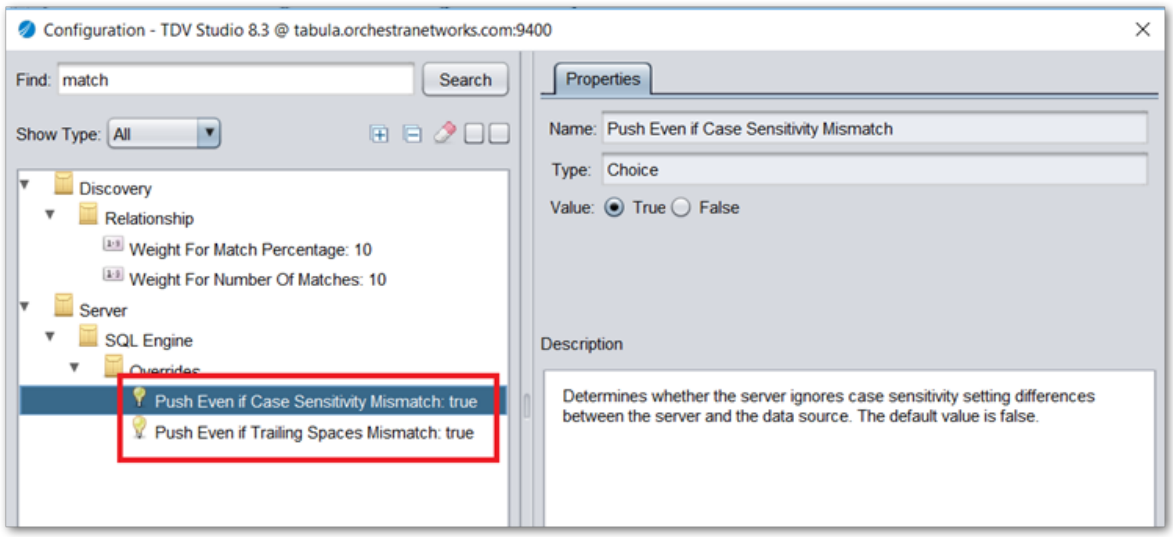
Data sources and resources corresponding to the assets that you have selected for provisioning are assembled in this public data source and their metadata is visible in TDV Studio and in the system database that third-party software can query.

Common error messages

In some cases, the interaction between Agent and TDV can fail because of the incorrect settings or incompatibility. Below is the list of common errors that may be solved by fine tuning on either TDV or Agent/EBX® Metadata Management application side.

Error message Reasons and resolution Selectable EBX_HG_ACCOUNT.OP cannot be pushed because the data source does not support upper functions and case sensitivity setting doesn’t match

This error message may appear during sampling or profiling of the data on certain data sources. To avoid this go to TDV Studio - Administration - Configuration menu and type *match* in the search bar. Look for the *Push even if Case Sensitivity Mismatch* setting and switch it to *true*. It is recommended to do the same for the *Push even if Trailing Spaces Mismatch*. Save and apply these changes to solve the issue.



139.2 TDV Connection rights

The agent must be authorized to connect to TDV instance using two connections flows:

- JDBC connection to the TDV systcode datasource using JDBC URL defined in property `ma.tdv.datasource.url`.
- SOAP connection using URL defined in property `ma.tdv.soap.resources-url`.

Check that these types of connections is authorized on your TDV instance prior launching the Agent.

See Connection Encryption details on Installing the TIBCO® Metadata Agent [p 815] if required

139.3 Properties template

The following shows a sample template:

```
#########################################################
#########################################################
#####                                               #####
#####               TDV Database                    #####
#####                                               #####
#########################################################
#########################################################
ma.tdv.datasource.url=jdbc:compositesw:dbapi@[[TDV SERVER HOST]]:9401?domain=composite&dataSource=system
ma.tdv.datasource.username=[[TDV USER]]
ma.tdv.datasource.password=[[TDV PASSWORD / MAY BE ENCRYPTED]]

#########################################################
#########################################################
#####                                               #####
#####               TDV SOAP                        #####
#####                                               #####
#########################################################
#########################################################
#Base URL for TDV SOAP interfaces
ma.tdv.soap.base-url=http://[[TDV SERVER HOST]]:9400
ma.tdv.soap.username=[[TDV USER]]
ma.tdv.soap.password=[[TDV PASSWORD / MAY BE ENCRYPTED]]
#A root folder to get private data sources. This folder will be scanned as well as its subfolders
#You can indicate several root comma separated paths
ma.tdv.soap.datasources-root-folder=/shared/examples
#Folder in which Agent will create new private datasources from TCMD. It must exist in your TDV server.
ma.tdv.soap.new-datasources-folder=/shared/examples
#Additional SQL data types not supported for profiling feature
ma.tdv.profiling.not-supported-types=
#Additional SQL data types supported by profiling feature except for the cardinality analyses
ma.tdv.profiling.not-supported-types-for-cardinality=
```

Properties configuration

Prior to launching the TIBCO® Metadata Agent with TIBCO® TDV Metadata Adapter, you need to complete the information in the properties file to check the connection to TDV.

Users are strongly discouraged to modify properties that are not described in the table below but are present in the properties file. A good practice is to remove old specific properties from your prior-to-version 4 agent, in order to keep only specific properties in your configuration file.

Property name Description

TDV Database connection

TDV SOAP connection

Private properties (should only be overwritten in particular cases)

Property Name	Description
TDV Database connection	
`ma.tdv.datasource.url`	The composite data source URL to connect the Agent to the systcode database of TDV. The expected format is `jdbc:compositesw:dbapi@<host>:<port>?domain=composite&dataSource=systcode`
`ma.tdv.datasource.username`	TDV user name with read permission on the system data sources.
`ma.tdv.datasource.password`	Corresponding password
TDV SOAP Connection	
`ma.tdv.soap.base-url`	URL of the TDV SOAP operations. The expected format is `https://<host>:<port>`
`ma.tdv.soap.username`	TDV user name with read-write permissions for the SOAP operations
`ma.tdv.soap.password`	Corresponding password
`ma.tdv.soap.datasources-root-folder`	A root folder to get private data sources. This folder will be scanned as well as its subfolders. You can indicate several comma separated paths.
`ma.tdv.soap.new-datasources-folder`	Folder in which Agent will create new private data sources from the EBX® Metadata Management application.
`ma.tdv.profiling.not-supported-types`	List of SQL data types that are not supported for profiling function. Types should be listed in lower case comma-separated.
`ma.tdv.profiling.not-supported-types-for-cardinality`	List of SQL data types that are supported for profiling, but not for the cardinality functions. Types should be listed in lower case comma-separated.
Private properties (should only be overwritten in particular cases)	
`ma.tdv.soap.public-databases-folder`	Folder for the public data sources creation. Metadata Provisioning function will create public containers under this folder. This folder must point to the TDV */services/databases* folder. This property is not defined in the public properties and must not be overwritten except for the cases where TDV upgrade includes change in the internal structure.
`ma.tdv.soap.introspect-result-timeout`	Timeout for the introspection task execution (real time in seconds is calculated as timeout * interval). Default value if property is not provided equals to 600 seconds. If this timeout is reached, Agent will stop checking state of the introspection task in TDV and will return error for ongoing harvesting request.
`ma.tdv.soap.introspect-result-interval`	Interval in seconds with which Agent checks the state of an introspection task in TDV. Default value if this property is not provided equals to 5s.

Property Name	Description
`ma.tdv.datasource.max-query-timeout`	Interval in seconds for the query timeout on the TDV datasource. By default it is equal to 600s.

Once these properties are configured, you can launch the agent as a Java application, using command line or any other suitable tool (see below on this page).

Upon each launch the adapter will test TIBCO® Data Virtualization JDBC and SOAP connection.

If the adapter start up is successful, it will be visible in the **Metdata Administration** perspective for the corresponding agent as described TIBCO® Metadata Agents [p 845].

139.4 **Troubleshooting**

Error message	Cause and resolution
Failed to initialize TDV datasource: Invalid login attempt.	This error appears when the Agent cannot connect to the TDV instance because login or password are not correct. Check the information you set up in properties *ma.tdv.datasource.username* and *ma.tdv.datasource.password*.
Failed to initialize TDV datasource: Failed to finish connect: *Connection timed out: no further information*	This kind of issue can be caused by different reasons. Here are some most common ones: - incorrect host and port for TDV datasource connection. Check the value in *ma.tdv.datasource.url* property - port not opened on the TDV server. Check that the port you indicated in *ma.tdv.datasource.url* property is opened on the server and connections are allowed from the machine where the agent is installed - firewall blocking access between TDV server and the machine where the agent is installed.
Failed to initialize TDV datasource: Cannot load driver class	Check that the driver is present into the classpath.

CHAPTER 140

TIBCO® EBX® Metadata Adapter installation

This chapter contains the following topics:

1. Overview
2. EBX® Connection rights
3. Properties template
4. Troubleshooting

140.1 Overview

Use this adapter to connect the TIBCO Metadata Agent to one or more EBX® instances to source data for the EBX® Metadata Management application using the harvesting feature.

> **Note**
>
> The Metadata Agent feature requires EBX® version 6.2.0.

140.2 EBX® Connection rights

You must authorize the agent to connect to an EBX® instance. Configure its connection as a user on the target EBX® instance and give the agent appropriate permissions.

See EBX® Metadata Management application Connection Encryption [p 822] for more details.

140.3 Properties template

The following shows a sample template:

```
#####################################################
#####################################################
#####                                           #####
#####              EBX adapter                  #####
#####                                           #####
#####################################################
#####################################################
# General properties for Tibco EBX adapter
# Do not modify
ebx.label=EBX Adapter
ebx.description=Metadata agent adapter for EBX
ebx.version=1.0.0
ebx.name=EBX Adapter
ebx.provider=Tibco
```

```
ebx.minVersion=6.2.0
ebx.defaultLocale=en-US
#This file holds properties for all the Tibco EBX instances to connect to.
#Properties below must follow the pattern ebx.<property>.<instance name>=<property value>
#Below is the configuration pattern
#ebx.base-url.<instance name>=
#ebx.path.<instance name>=
#ebx.username.<instance name>=
#ebx.password.<instance name>=admin
#ebx.public-key.<instance name>=key
#ebx.useEncryption.<instance name>=true
#Format ebx instance name as alphanumeric name all in lower case bellow.
ebx.instances=localhost
ebx.base-url.localhost=http://localhost:8080
ebx.username.localhost=admin
ebx.password.localhost=admin
ebx.locale.localhost=fr-FR
#ebx.public-key.localhost=key
#ebx.useEncryption.localhost=true
```

Properties configuration

Prior to launching the TIBCO® Metadata Agent with the TIBCO EBX® Metadata Adapter, check the connection with EBX® by updating the information in the properties file.

Attention

It is strongly discouraged to modify properties that are not described in the table below, but are present in the properties file.

Property	Description
`ebx.instances`	List of comma-seperated instances the agent will connect to. `ebx.instances=instanceId1,instanceId2`
`ebx.base-url.instanceId`	Base URL of the EBX® instance.
`ebx.username.instanceId`	User name the agent will connect as on the target EBX® instance.
`ebx.password.instanceId`	User password the agent will connect as on the target EBX® instance.
`ebx.locale.instanceId`	Optional. Locale used to harvest localised information in EBX® instance. Example: `ebx.locale.instanceId1=fr-FR`
`ebx.public-key.instanceId`	Optional. If the encryption is used for connection, fill the public key
`ebx.useEncryption.instanceId`	Optional. Activate the encrypted password connection `ebx.useEncryption.instanceId1=true`

After configuring the above properties, you can launch the agent as a Java application using the command line or any other suitable tool (see below on this page).

Upon each launch the adapter tests the TIBCO EBX® REST connection. If the adapter start up succeeds, you can view it in the EBX® Metadata Management application's **Metadata Administration** perspective for the corresponding agent as described TIBCO® Metadata Agents [p 845].

140.4 **Troubleshooting**

Error message	Cause and resolution
Failed to initialize EBX® datasource: Invalid login attempt.	This error appears when the Agent cannot connect to the EBX® instance because the login or password are not correct. Check the information you set up in properties *ebx.instance.username* and *ebx.instance.password*.
Failed to initialize EBX® datasource: Version Not Supported	This kind of issue is caused by attempting to connect to an incompatible version of EBX®. This adapter is only compatible with EBX® 6.2.0 and up.
Failed to connect to EBX® datasource	This can be caused for multiple reasons, but it's most likely an error in the *ebx.instance.url* property, or a routing issue.

CHAPTER 141

TIBCO® Metadata Agent Framework

This chapter contains the following topics:

1. Overview and purpose
2. Extending metadata extraction capabilities

141.1 Overview and purpose

TIBCO® Metadata Agent Framework allows customers to implement support for the specific data source types within TIBCO® Metadata Agent through specific adapters.

An adapter is a separate Java application that ensures one or several of EBX® Metadata Management capabilities for a given data source type(s):

- Data source management
- Metadata harvesting
- Data sampling
- Metadata provisioning
- Lineage extraction.

TIBCO® Metadata Agent Framework includes following elements:

- Java based interfaces that must be implemented in order to customize a given metadata management capability for a given data source type
- Java annotations that must be used to indicate that a capability is implemented
- Properties allowing to activate an adapter and define relationship between this adapter and supported data source types/EBX® Metadata Management capabilities
- Extensible exception and error codes to properly raise an internationalized error from the adapter to the agent.

141.2 Extending metadata extraction capabilities

To extend existing metadata extraction capabilities you need to implement an adapter using Java programming language in respect with defined interfaces contract.

The adapter must respect following rules:

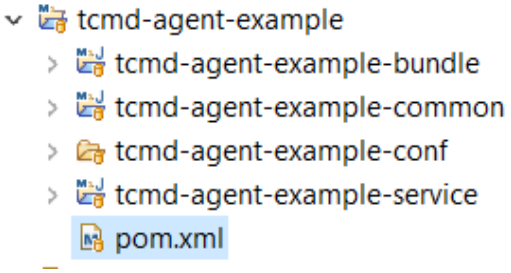
- be compiled as a Java application (jar) file

- provide mandatory registration data to the agent by implementing `addAdapterInfos()` method of the registration service
- implement at least one metadata extraction service respecting the interface contract and mandatory annotations
- be registered in the agent properties file.

Metadata Agent leverages Spring Boot 2.5.4 framework. The adapter implementation must include Spring annotations for services (`org.springframework.stereotype.Service`).

Recommended adapter project structure

The adapter can be developed in any way that is comfortable for the responsible development team. Below is a recommended structure based on Maven dependencies management and architecture best practices for Java projects.



The main project contains three module projects and one configuration project:

Module name	Description
tcmd-agent-example-bundle	Bundle project that defines how to assemble your adapter into one jar file and allows to manage dependencies in a simple way.
tcmd-agent-example-common	Common library code for your adapter. Typically this would host error management framework and some common properties and utilities as well as i18n resources.
tcmd-agent-example-conf	Project folder that hosts properties and configuration files for your adapter.
tcmd-agent-example-service	Code implementing agent services inside your adapter.

Register adapter in TIBCO® Metadata Agent Framework

In order to register an adapter in TIBCO® Metadata Agent Framework you must follow these steps:

1. Implement the registration service

The registration service for your adapter must implement the dedicated Agent interface `com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterRegistrationService` service and its main method `addAdapterInfos()`.

This method allows you to create an adapter information object `com.tibco.ebx.ca.tabula.shared.agent.dto.AdapterDTO` where you must provide the following information about your adapter:

Property	Description
name	Technical name of the adapter. It must match the value you'll use for the implementation annotations and for the Agent's property `agent-adapter.active-profiles`
version	Implementation version for your adapter.
provider	Your company name.
productVersion	Corresponding version of the supported metadata provider.
description	Adapter description that will be shown in EBX®.

2. Provide suitable annotations

In order to be recognized by TIBCO® Metadata Agent Framework each service class that you consider implemented must be annotated with `com.tibco.ebx.ca.tabula.agent.common.annotation.MetadataAdapterProfile` having as value your adapter technical name (provided in the Agent's property `agent-adapter.active-profiles` and in the attribute `name` of the Adapter information object).

Each operation in the service that you consider implemented must be annotated with `com.tibco.ebx.ca.tabula.agent.common.annotation.MetadataOperationImplemented`.

3. Add adapter name and properties to main agent's properties

Following properties must be modified to indicate that the Agent should load your adapter:

```
#Comma-separated list of packages to scan for adapters classes agent-adapter.base.package=com.tibco.ca.tabula.agent.tdv, com.mycompany.agent #Comma separated list of profiles (corresponding to adapter technical names) agent-adapter.active-profiles=tdv,myadapter
```

4. Add adapter name to the active profiles

Edit your Java launch configuration for the Agent to add your adapter name as profile:

```
-Dspring.profiles.active=tdv,myadapter
```

5. Add adapter jar and optionally properties to the Agent's CLASSPATH.

Deploy your adapter jar and optionally any properties/configuration files that are required by your code on Agent's `CLASSPATH`.

Activate your adapter for the suitable data source types in `application-datasources.properties` file as described in Installing the TIBCO® Metadata Agent [p 815].

Implementing main capabilities

The following metadata capabilities can be implemented by an adapter:

- Data source management
- Metadata harvesting
- Data profiling
- Data sampling
- Metadata provisioning
- Lineage extraction.

General implementation rules

Each custom service that implements main metadata capabilities must comply with following rules:

1. Each service class that you consider implemented must be annotated with `com.tibco.ebx.ca.tabula.agent.common.annotation.MetadataAdapterProfile` having as value your adapter technical name (provided in the Agent's property `agent-adapter.active-profiles` and in the attribute `name` of the Adapter information object).
2. Each service class must be annotated with Spring `org.springframework.stereotype.Service` annotation.
3. Each operation in the service that you consider implemented must be annotated with `com.tibco.ebx.ca.tabula.agent.common.annotation.MetadataOperationImplemented`.
4. For each object of type `com.tibco.ebx.ca.tabula.shared.agent.landing.dto.MetadataExchangeDTO` created by the adapter, you must populate at least `sourceId`, `sourceType`, `datasourceType` and `name` attributes.

Data source management

Data source management capability covers following operations with data sources in the metadata provider system:

- read all available data sources
- create a new data source
- delete an existing data source
- check availability of an existing data source.

This capability can be extended by implementing the `com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterDatasourceManagementService` service.

Metadata harvesting

Metadata harvesting capability covers following operations for data sources and underlying resources in the metadata provider system:

- create a new data source and get all metadata for it
- get metadata for a resource/data source or a list of resources/data sources
- get list of children resources for a root resource/data source.

This capability can be extended by implementing the `com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterHarvestingService` service.

Data profiling

Data profiling information can be added to a metadata exchange object.

This capability can be extended by implementing the `com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterHarvestingService` service, method `addProfilingToAsset`.

Data sampling

Sample data can be requested from a source system for a given data source/resource.

This capability can be extended by implementing the `com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterSamplingService` service.

Metadata provisioning

Metadata provisioning capability covers operations that are for goal to provide some existing data sources/resources to a `public` space on the metadata provider system.

This capability also allows to create a new data source/resource based on several existing ones.

Following operations are supported:

- provision metadata (create a new public source and publish resources in there)
- modify already provisioned metadata resources
- delete already provisioned metadata resources
- check that a resource has already be provisioned
- check whether a given resource is supported to be source of provisioned resource.

This capability can be extended by implementing the `com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterProvisioningService` service.

Lineage extraction

Lineage information can be extracted from the source system if the source system supports this. This information can also be added to the Metadata exchange objects and pushed through TIBCO® Metadata Agent to {{ site.PRODUCT_NAME_R }} and stored as technical flows.

This capability can be extended by implementing the `com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterLineageService` service.

Exceptions management

A specific exceptions management framework is provided as part of TIBCO® Metadata Agent Framework.

This framework includes the following elements:

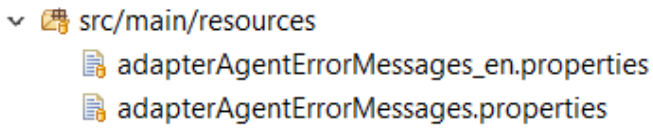
- `com.tibco.ebx.ca.tabula.agent.common.adapter.exception.MetadataAdapterException`: internationalized exception class that you should implement to throw specific exceptions from your code to TIBCO® Metadata Agent and {{ site.PRODUCT_NAME_R }} to handle.
- `com.tibco.ebx.ca.tabula.agent.common.exception.MetadataExceptionReason` : an interface to be implemented by enumeration classes to define custom exception reasons.

To implement your specific exception, you need to extend the generic exception class with suitable constructors and override `getErrorBundle()` method to return the name of your error messages

file. This file is expected to be available inside `src/main/resources` folder with following naming convention:

```
<error bundle name>_<language 2 letters code>.properties
```

Language code is optional, if not indicated the bundle will be considered applicable for English language.



To add specific error reasons for your exception(s), you need to create an `enum` in your project that implements `MetadataExceptionReason` interface and define your error messages keys into it.

```
public enum MyMetadataExceptionReason implements MetadataExceptionReason {

    AGENT_ADAPTER_ERROR("agent.adapter.error");

    private MyMetadataExceptionReason(final String ckey) {
        this.key = ckey;
    }

    private String key;

    @Override
    public String getKey() {
        return key;
    }
}
```

If you don’t wish to define specific exceptions, you can use generic technical exception provided by TIBCO® Metadata Agent Framework represented by the class 'com.tibco.ebx.ca.tabula.agent.common.adapter.exception.MetadataAdapterTechnicalException'

This exception can be instantiated using a string message or a `Throwable` object.

Logging

No specific logging framework is provided as of today.

TIBCO Metadata Agent leverages slf4j logging framework to write functional messages in suitable log files.

For consistency, we recommend you using the same framework for your adapter and avoid using `System.out.print/println` to write messages.

CHAPTER 142

Administration of TIBCO® Metadata Agent

This chapter contains the following topics:

1. EBX® Metadata Management application subscription
2. Agent Migration and Backup

142.1 EBX® Metadata Management application subscription

New connection for a previously deleted agent

On the first start, the agent subscribes to the corresponding EBX® system. As a result of this interaction, a GUID will be stored in the agent storage and used for all exchanges between the agent and EBX®. If you delete a managed agent in the EBX® **Administration** perspective, the GUID stored on the agent's side will no longer be valid. To recreate a deleted agent, you have to delete the agent's H2 database (`configDB.mv.db`) and restart the agent. It will automatically subscribe to the EBX® Metadata Management application and obtain a new GUID.

Public key update

The Public key (JWT) is a unique token used for additional security in exchanges between the agent and the EBX® Metadata Management application.

It is valid 60 days and expires automatically after. Connections with an expired JWT are not accepted by the EBX® Metadata Management application and a specific exception is generated to notify the agent that its public key is no longer valid: JWT for the Agent with id xxx has expired or is not valid. Please, get update the JWT in the Agent's properties.

After seeing this message you need to obtain a new JWT from a EBX® administrator, copy-paste it in the public-key property in the agent's properties file and restart the agent.

142.2 Agent Migration and Backup

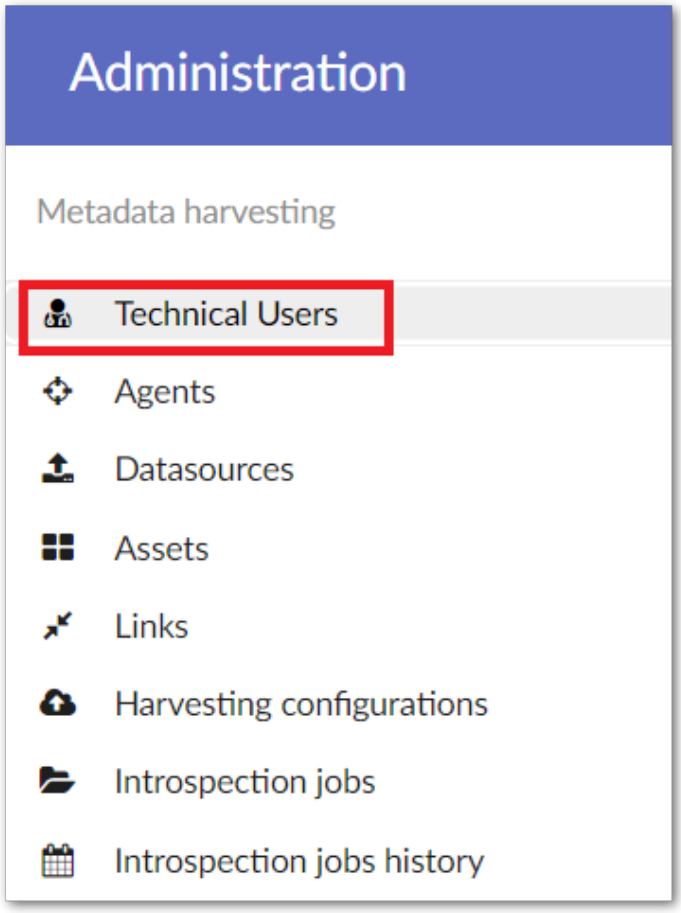
If you need to migrate your existing Agent and preserve its connection to the EBX® Metadata Management application, please, follow the process below:

1. Stop agent.
2. Backup agent’s database and Quarz Scheduler database by copying files `configDB.mv.db` and `quartzDB.mv.db`
3. Backup your `application.properties` file.
4. Install the agent on the new instance.
5. Copy databases backup into `[[Installation folder]]/storage`.
6. Copy properties file into `[[Installation folder]]/conf`.
7. Update `ma.http.ws-url` and `ma.http.port`.
8. Update other properties if necessary.
9. Restart agent. It will automatically update registration information in EBX®.

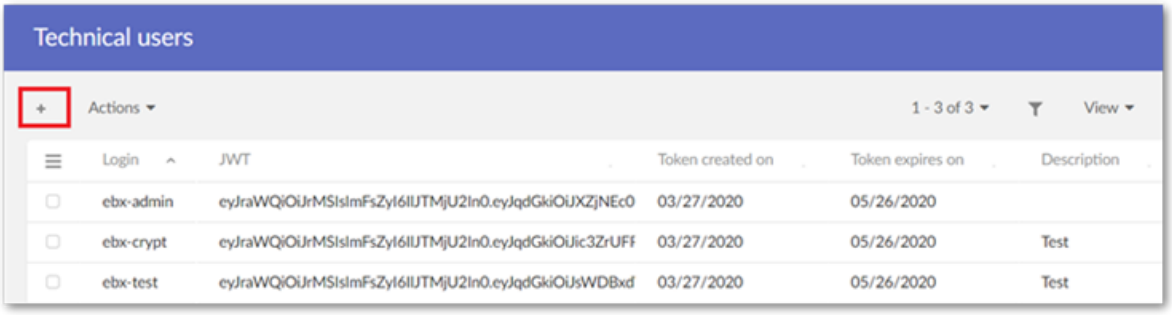
CHAPTER 143

Technical Users

Technical users are a specific type of user that do not have UI access and are solely meant to use the Metadata agent or another technical system. You can access the list of your technical users by selecting the **Metadata Administration** perspective and going to the **Technical Users** section.



Under **Technical Users** in **Metadata administration** perspective, you can select an existing user to specify that they are authorized to make a connection between the agent and the EBX® Metadata Management application.



The system automatically generates a **JSON Web Token** for this user. If this technical user will be associated with a Metadata Agent, you must copy-paste the **JWT** into the Metadata Agent's properties files, otherwise the communication between the EBX® Metadata Management application and your Metadata Agent won't be allowed.

Please, note that JWT is valid for 60 days and will be automatically regenerated by the EBX® Metadata Management application once expired.

When expired, you'll get an expiration message on agent operations (harvesting, creating data sources from TDV, …) and the agent will ask to EBX® to generate a new JWT Key.

If you get this expiration message, you'll have to

- stop your agent.
- get the new JWT key from EBX®.
- copy/paste into agent `application.properties`.
- restart the agent.

CHAPTER 144

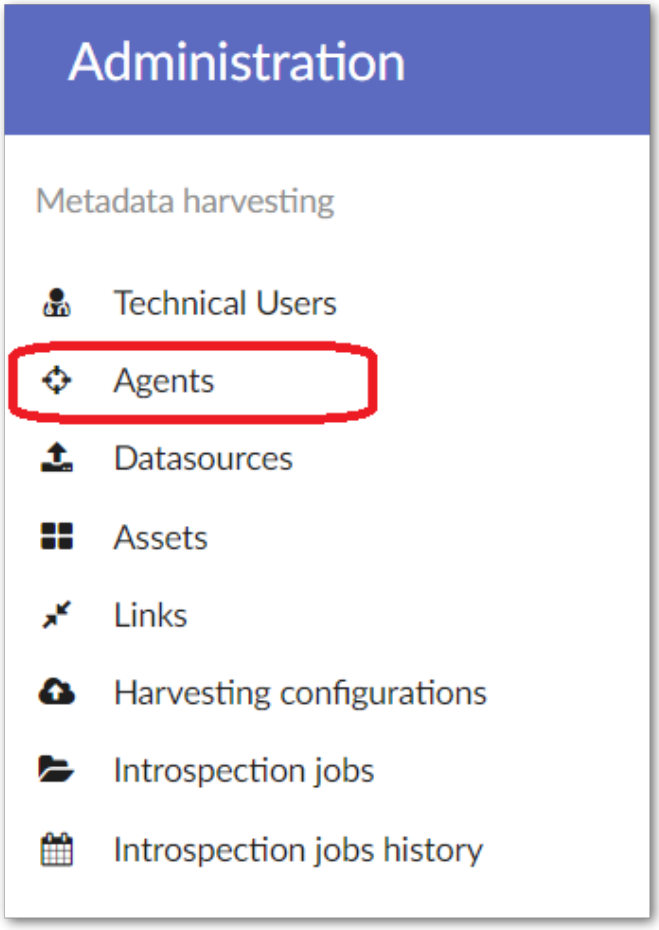
TIBCO® Metadata Agents

This chapter contains the following topics:

1. Overview
2. Metadata Agent registration
3. Set Metadata Agent as favorite
4. Disable Metadata Agent
5. Delete Metadata Agent

144.1 **Overview**

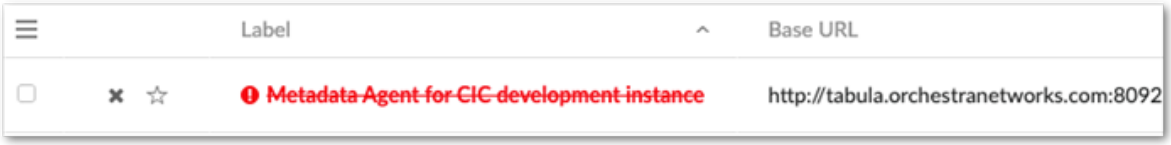
In the **Metadata Administration** perspective select **Agents** to access the list of your registered Metadata Agents.



By default, only active Metadata Agents are visible. If you want to see the inactive Metadata Agents, go to the **View** menu and switch to **Deactivated TIBCO Metadata Agents**.

The records in this table are created and updated automatically when a Metadata Agent starts and contacts Metadata to subscribe or check that the subscription is up-to-date.

If an agent is not available, it will appear in red. On mouse over, the reason of the unavailability is displayed.



144.2 Metadata Agent registration

When a Metadata Agent subscribes to the EBX® Metadata Management application the following elements are created in the **Agents** table:

Module name	Description
GUID	The GUID generated by the EBX® Metadata Management application for the Agent during the subscription.
Label	A label that will be used to present Metadata Agents in user interactions.
Base URL	Agent's base URL.
Agent API Version	REST API version of this agent. This defines a list of the operations available.
Timezone	Agent's timezone that will be used to convert dates for scheduled jobs.
JWT	JSON Web Token corresponding to the technical user with which Metadata Agent connects to the EBX® Metadata Management application. See Technical Users [p 843] for more details.
Technical User Name	Technical user login. See Technical Users [p 843] for more details.
Landing dataspace	Identifier of the dataspace that Metadata Agent uses to store raw metadata retrieved from TDV.
Destination Dataspace	Identifier of the target dataspace where the metadata will be integrated.
TDV JDBC URL	JDBC URL of the TDV system data source.
TDV Host	Host name where the TDV instance is situated.
TDV Port	Port number on which the TDV instance is listening for the JDBC connection.
TDV Version	TDV Version.
Published views folder	Path to the TDV published views folder. See Administration of TIBCO® Metadata Agent [p 841]
Operations	List of the API operations available in Metadata Agent.

These elements are verified and updated if necessary each time the Metadata Agent instance is restarted.

On subscription, an instance corresponding to the TDV system data source is created in the EBX® Metadata Management application to allow harvesting of the published view available in TDV.

144.3 Set Metadata Agent as favorite

Several Metadata Agents can be subscribed to the same EBX® Metadata Management application instance. In order to define one of them to be used in priority for the instances that are not yet linked to another agent, select the **Select as favorite** icon near Metadata Agent:

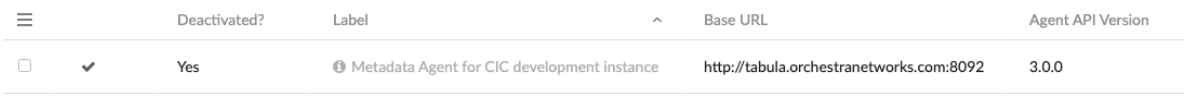


To change your favorite Metadata Agent select the **Set as favorite** icon for another Metadata Agent. You can only have one favorite agent.

144.4 Disable Metadata Agent

To disable Metadata Agent and all related objects (Managed data sources, assets, links and introspection jobs), select the **Disable** icon near the Metadata Agent label.

This Metadata Agent will no longer be visible in your list of registered agents and it won't be possible to run any operation with this agent. To see a disabled agent, go to the **View** menu and switch to **Deactivated TIBCO Metadata Agents**.



From this view, you can re-enable a Metadata Agent by clicking on the **Enable** button near it. However, the related managed objects won't be re-enabled with the agent.

144.5 Delete Metadata Agent

To delete permanently a Metadata Agent from the EBX® Metadata Management application, select this agent and select the **Delete** icon above the table:

Note: This operation cannot be undone. |



This operation will also delete related objects from the system:

- Managed data sources
- Managed assets
- Managed links

- Introspection Jobs and corresponding job executions history

CHAPTER 145

Items managed by Metadata Agent

This chapter contains the following topics:

1. Overview
2. Managed data sources
3. Managed assets
4. Managed links
5. Relation of the managed items with Metadata objects

145.1 Overview

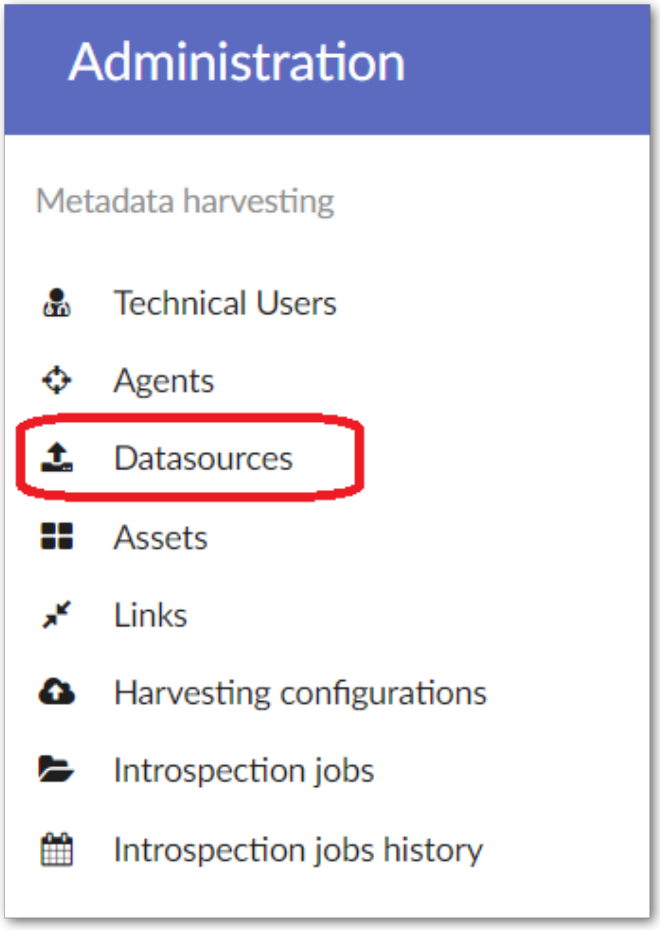
Metadata Agent allows you to gather raw metadata from the data sources for three managed objects categories:

1. Managed data source - corresponds to the TDV data source defined in the private part or to the folder containing TDV published view.
2. Managed assets - children resources of the data source for which metadata can be retrieved.
3. Managed links - relations between resources, such as foreign keys for relational databases.

You can access a list of the objects managed by Metadata Agents in the **Administration** perspective.

145.2 **Managed data sources**

In the **Administration** perspective select **Datasources** to access the list of the data sources managed by subscribed Metadata Agents.

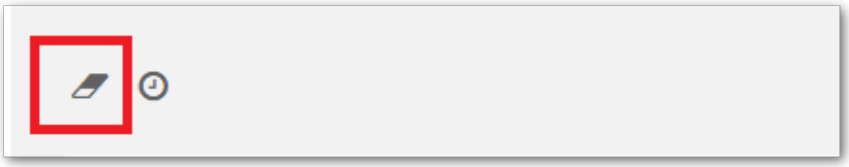


The following information is available for each managed data source:

Field	Description
Agent identifier	The identifier of Metadata Agent that is connected to this data source.
Data source Identifier in TDV	Resource path of this data source in TDV instance. Instance
Identifier in the EBX® Metadata Management application	If the managed data source is already connected to an instance in the EBX® Metadata Management application, the corresponding identifier will appear in this field.
Status	Data source status: • `NEW` - represents a new resource that was not transferred to the EBX® Metadata Management application yet. • `STORED` - represents a data source that is already stored in the EBX® Metadata Management application as an instance. • `DELETED` - represents a data source that was deleted in TDV instance.
Created at	Creation date for this record.
Last updated	Last update date for this record.
Is disabled?	A data source is disabled if the corresponding agent is disabled.
Datasource Type	Technical data source type in TDV. See Administration of TIBCO® Metadata Agent [p 841] for the full list of the supported data source types.

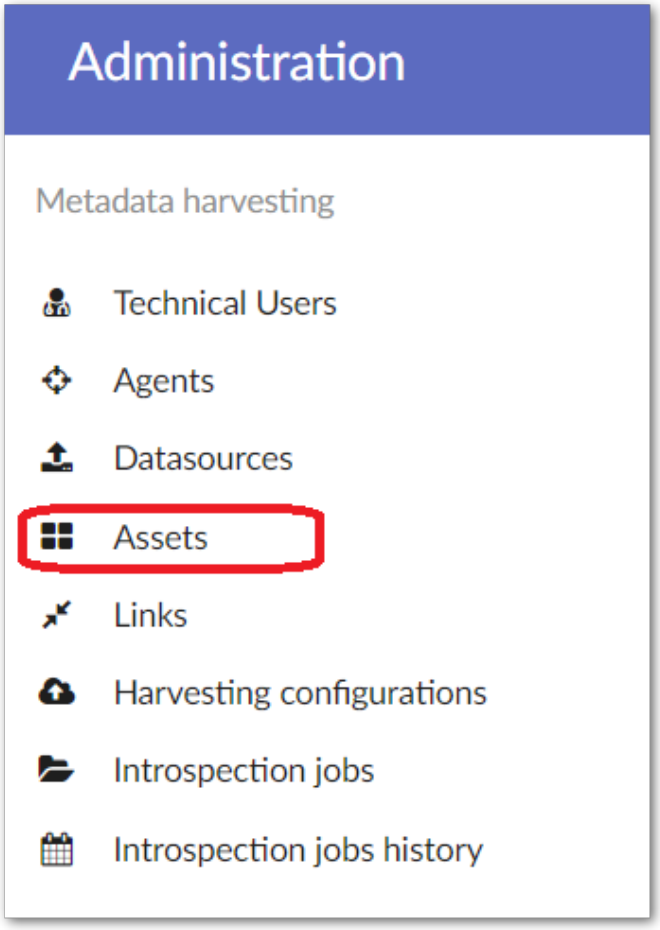
Clean deleted data sources

To delete all data sources with the DELETED status from the system, select the **Clean deleted** icon above the table. This will permanently delete these data sources from this table. Related links and assets will also be deleted.



145.3 **Managed assets**

In the **Administration** perspective select **Assets** to access the list of the assets managed by subscribed Metadata Agents.

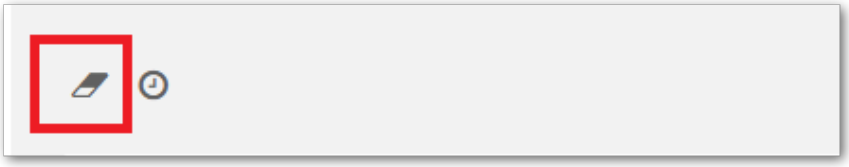


The following information is available for each managed data source:

Field	Description
Datasource	The data source to which the asset belongs.
Agent identifier	The identifier of Metadata Agent that is connected to this asset.
Resource ID in TDV	Resource path of this asset in TDV instance.
Asset ID in the EBX® Metadata Management application	If the managed asset is already connected to an asset in the EBX® Metadata Management application the corresponding identifier will appear in this field.
Status	Asset status: • `NEW` - represents a new resource that was not transferred to the EBX® Metadata Management application yet. • `STORED` - represents a resource that is already stored in the EBX® Metadata Management application as an asset. • `DELETED` - represents a resource that was deleted in TDV instance.
Created at	Creation date for this record.
Last updated	Last update date for this record.
Is disabled?	An asset is disabled if the corresponding agent is disabled.
TDV Parent asset Id	Technical path to the parent element in TDV.

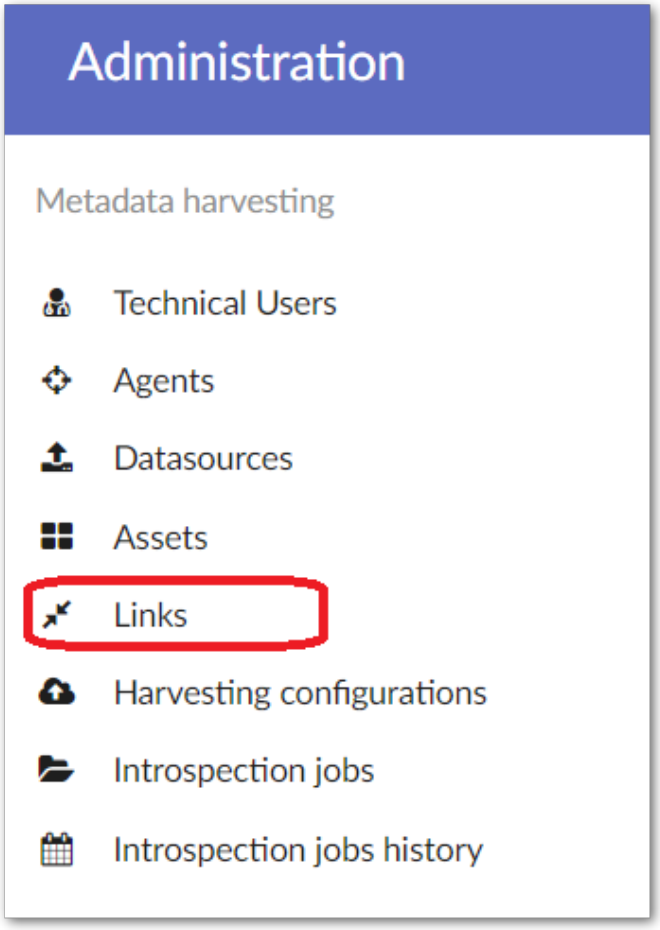
Clean deleted assets

To remove all assets with the DELETED status from the system, select the **Clean deleted** icon above the table. This will permanently delete these assets from this table. Related links will also be deleted.



145.4 **Managed links**

In the **Administration** perspective select **Links** to access the list of the links managed by subscribed Metadata Agents.

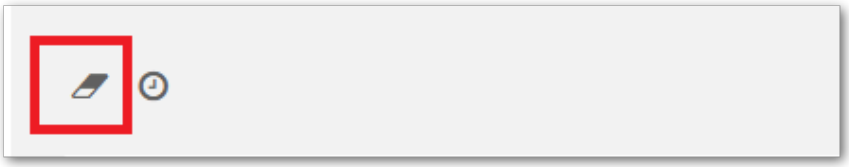


Following information is available for each managed data source:

Field	Description
Agent identifier	The identifier of Metadata Agent that is connected to this link.
ID in TDV	Name of the foreign key.
PK Asset	A managed asset that represents a primary key in this link relation.
FK Asset	A managed asset that represents a foreign key in this link relation.
Link ID in the EBX® Metadata Management application	If this managed link is already stored in the EBX® Metadata Management application as a link, the corresponding identifier will appear in this field.
Status	Link status: • `NEW` - represents a new resource that was not transferred to the EBX® Metadata Management application yet. • STORED - represents a resource that is already stored in the EBX® Metadata Management application as a link. • DELETED - represents a resource that was deleted in TDV instance.
Is disabled?	A link is disabled if the corresponding agent is disabled.
TDV Parent Id	Identifier of the parent link object.

Clean deleted links

To remove all links with the DELETED status from the system, select the **Clean deleted** icon above the table. This will permanently delete these links from this table.



145.5 Relation of the managed items with Metadata objects

Each stored managed item has a relationship with an item in the main Metadata data model.

The link is created during harvesting of the metadata through execution of the **Harvest metadata** service or **Create instance from datasource** service. Once the metadata is stored, the status of the managed item becomes STORED and the identifier of the corresponding EBX® Metadata Management application object is attached to it.

If an EBX® Metadata Management application object linked to a managed item is deleted, the link is broken and managed item's status becomes NEW again.

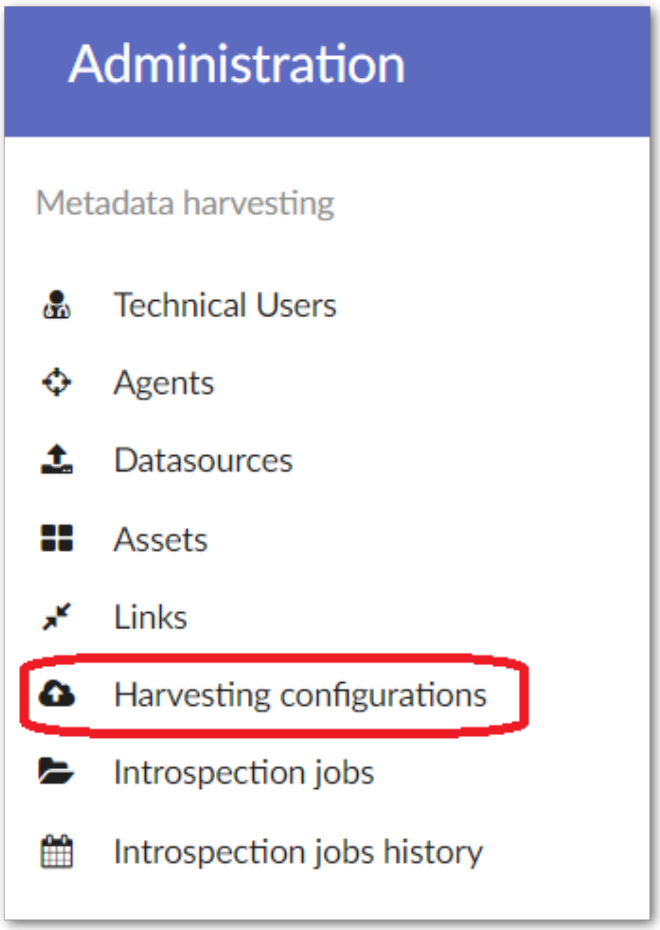
If a physical data source or resource is deleted in TDV, during the next harvesting executed for the corresponding object in the EBX® Metadata Management application the managed item will receive the DELETED status and the linked EBX® Metadata Management application object will be marked as disabled.

For the instances and assets that have a linked managed item, attribute name is read-only as it is used for technical exchanges between the EBX® Metadata Management application and TDV through Metadata Agent.

CHAPTER 146

Harvesting configurations

In the **Metadata Administration** perspective select **Harvesting configurations** to access the list of the harvesting configurations.



Harvesting configurations are created using the **Harvest metadata** service available from the **Instance** and **Asset** tables.

The **Metadata Administration** perspective provides a technical representation of the harvesting configuration and you can delete one or several configurations using the **Delete** icon above the table.

This chapter contains the following topics:

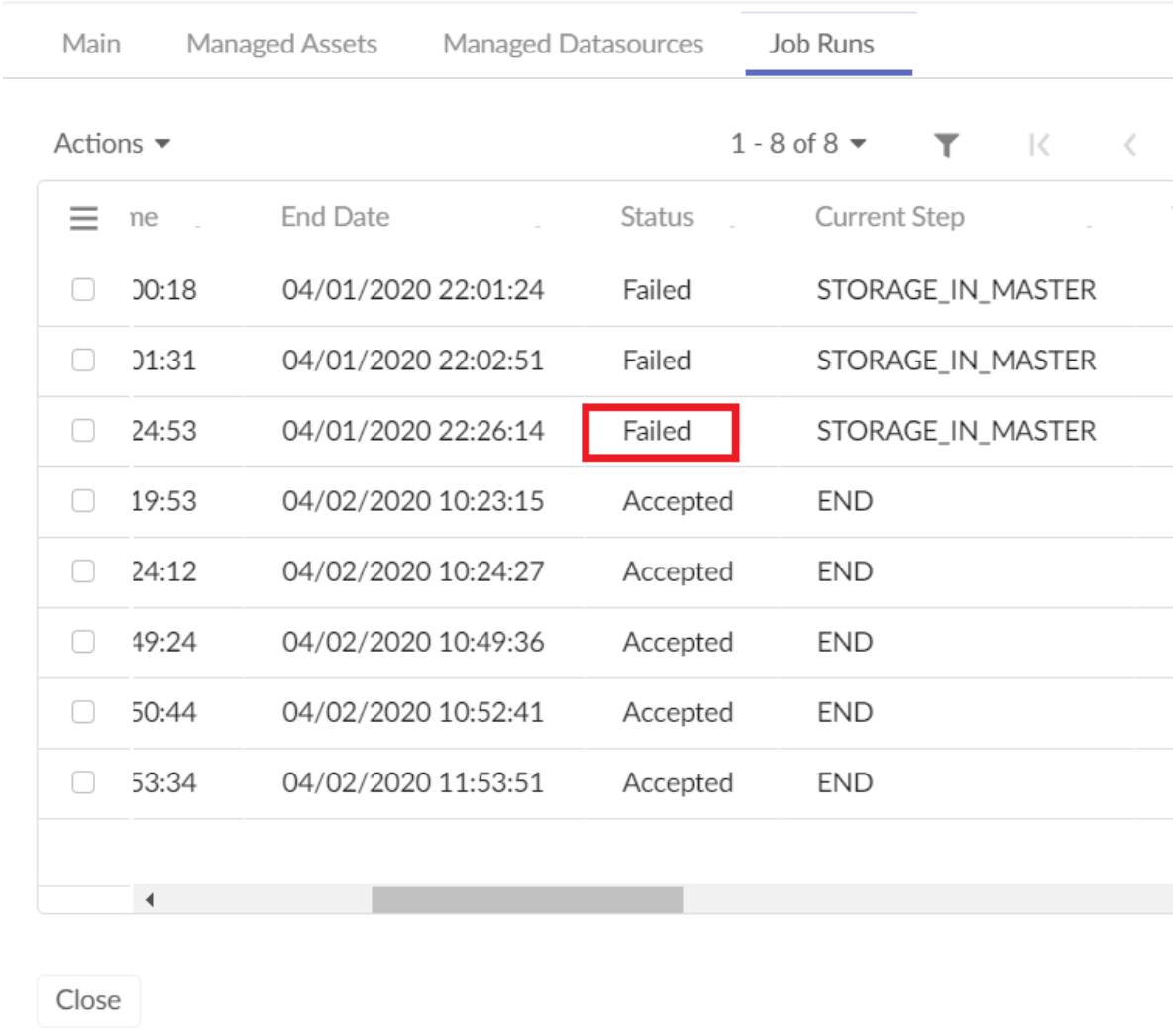
1. Troubleshooting harvesting run
2. Introspection Jobs
3. Introspection Jobs History

146.1 Troubleshooting harvesting run

The **Harvesting configurations** table may be useful in case you need to troubleshoot one of the harvesting runs. Double click on the configuration for which a failed run was reported to access more details.

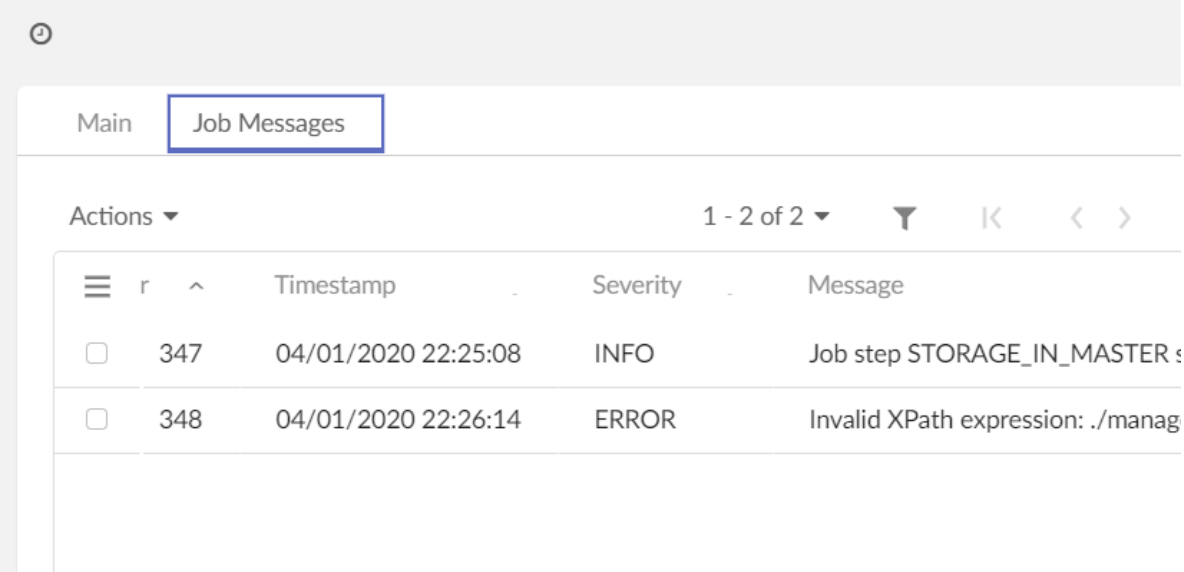
Click on the **Preview** icon near the introspection job to view its details.

Switch to the **Job Runs** tab to access the list of the executions for this job.



Double click on the latest job run that has a Failed status to access its details.

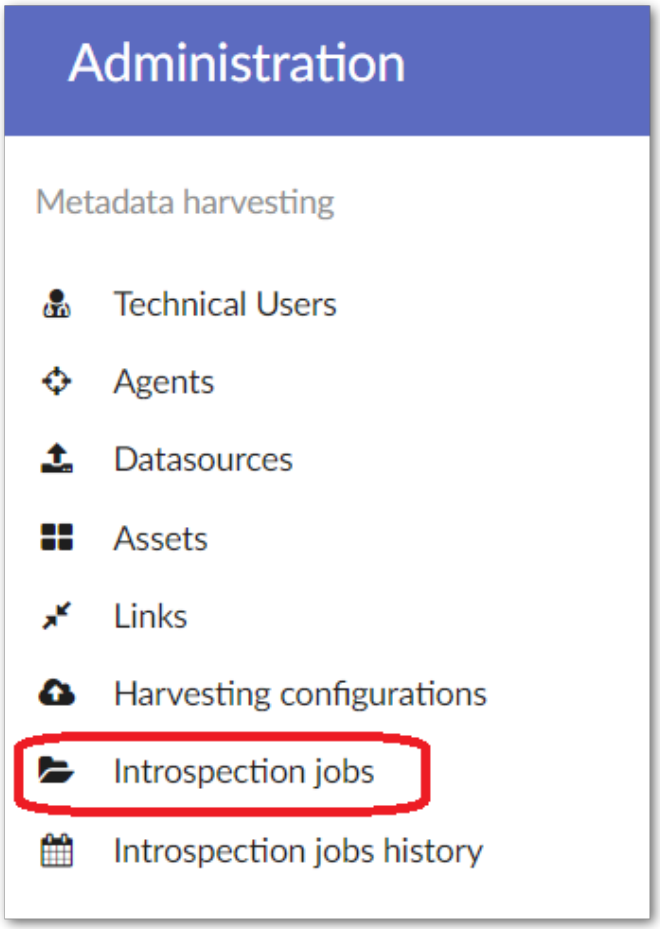
Switch to the **Job Messages** tab to access the log of important messages raised during the job execution.



The messages having a severity of ERROR are there to help you find the root cause of the run fail (impossible to connect to the agent, impossible to store objects, failed to complete run because of the server failure etc).

146.2 **Introspection Jobs**

In the **Administration** perspective select **Introspection jobs** to access the list of the harvesting jobs.



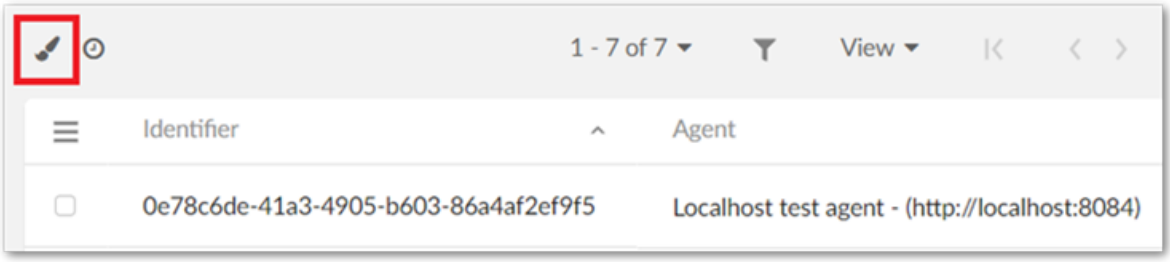
The following information is available for each job to help you to track the state of each job and, if necessary, troubleshoot the execution:

Field	Description
Identifier	A unique technical identifier used in exchanges between the EBX® Metadata Management application and Metadata Agent to identify this job.
Agent	Metadata Agent related to this job.
Start date and time	Date and time when the first job run started.
Interval (minutes)	Interval (in minutes) between the executions of this job if it is a scheduled job.
Repeat count	Number of executions for a scheduled job.
Next run	Date and time when the next execution of the job will happen.
Status	Execution status: • Created - the job is created but not run yet. • Scheduled - the job is scheduled in the corresponding Metadata Agent. • In progress - this status is given to a job once the first run starts and until the last run ends. • Failed - the last run failed. • Success - all runs of this job completed without errors.
Auto merge in workflow	Indicates whether the metadata harvested by this job will be automatically merged in the Master data or user's confirmation is required.
Activate sampling?	If true - sampling is expected to be launched during this job's execution for all assets that are eligible but don't have a sample attached yet.
Force sampling?	If true - sampling is expected to be launched during this job's execution for all assets that are eligible and overwrite any existing samples.
Automatic classification suggestion	If true - classification is expected to be launched during this job's execution for all assets that do not have a classification yet.
Force refresh of classification suggestion	If true - classification is expected to be launched during this job's execution for all assets.
Classify assets automatically	if true - classification suggestion will be automatically applied on assets.
Is disabled?	A job is considered disabled if the corresponding agent is disabled.

Field	Description
Harvesting Configuration ID	Harvesting configuration identifier for history purposes.
Managed Assets	List of the root assets that this job will harvest. For each root asset in the job there can be a list of children assets to exclude from harvesting scope.
Managed Datasources	List of the data sources that this job will harvest. For each of them, there can be a list of children assets to exclude from harvesting scope.
Job Runs	List of the executions of the current job.

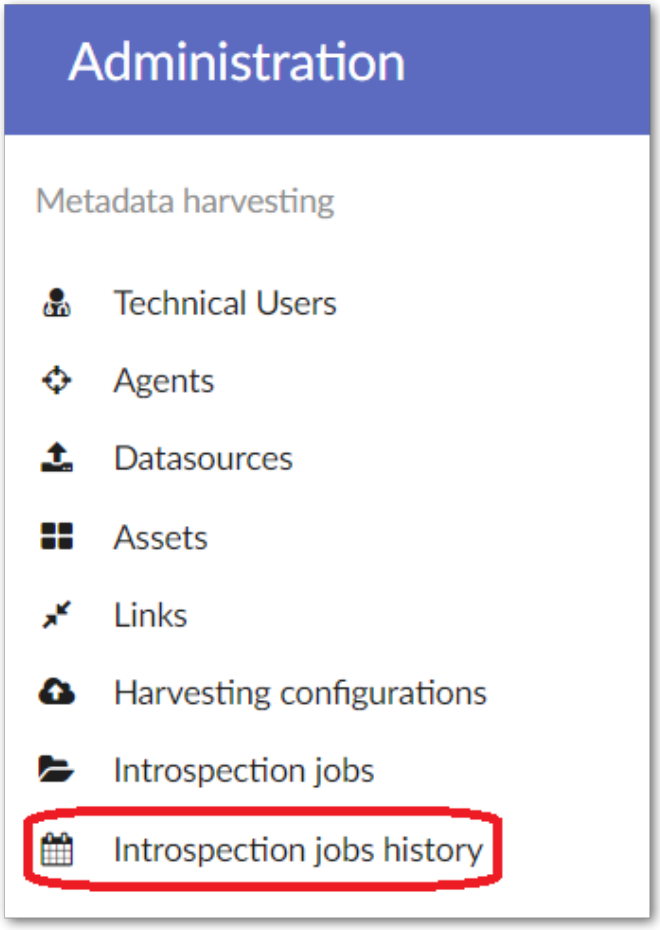
Clean failed jobs

To remove Introspection jobs that are in a Failed status select the **Clean failed jobs** icon above the table. This will delete all jobs and the corresponding job runs with related job messages.



146.3 Introspection Jobs History

In the **Administration** perspective select **Introspection jobs history** to access the list of the harvesting jobs' executions.



For each job execution, the following information is available to help you to track the state of each job and, if necessary, troubleshoot the execution:

Field	Description
Identifier	Autoincremented identifier for the run.
Introspection Job	Introspection job.
Iteration number	For manual job executions (launched directly by a user from **Harvest metadata** service), the iteration number is always **0**. For scheduled runs, the iteration number corresponds to the order in which runs are executed.
Start date and time	Start date and time for this run.
End Date	End date and time for this run.
Status	Execution status: - execution started. • **In progress** • **Failed** - execution failed. • **Success** - scheduled run has been executed successfully and metadata has been automatically merged into Master Data. • **Pending for approval** - manual run has been executed successfully and results are pending to be accepted by the user. • **Accepted** - results of the manual run have been accepted by the user and merged into Master Data. • **Rejected** - results of the manual run have been rejected by the user and deleted.
Current Step	Current execution step: • **STORAGE IN MASTER** - integration of the harvested metadata into child dataspace of the Master Data. • **SAMPLING** - requesting samples (if sampling activated). • **CLASSIFICATION** - classification of assets (if activated). • **PURGE_LANDING** - a purge of the Metadata Agent's landing dataspace. • **END** - execution end.
EBX® Metadata Management application Child dataspace	EBX® Metadata Management application Child dataspace created for this run.
Automatically merge results	Indicates whether the metadata harvested by this job run will be automatically merged in the Master data or user's confirmation is required.
Harvesting Configuration ID	Harvesting configuration identifier for history purposes.
Harvested Datasources	List of the data sources harvested by this run.

Field	Description
Harvested Assets	List of the assets harvested by this run.
Harvested Relations	List of the links harvested by this run.
Job Messages	List of the important messages logged by this job run. Typically are logged execution steps and errors occurred.

Security Guide

CHAPTER 147

Security Best Practices

Here is a list of best practices that are considered useful to enforce a good security level for the EBX® setup. These best practices apply to EBX® and to other environments, their configuration, protocols and policies. While these practices are commonly regarded as beneficial, they may not be relevant to your particular infrastructure and security policy.

This chapter contains the following topics:

1. Encryption algorithms
2. HTTPS
3. Installation
4. Web Server
5. Application Server
6. Java
7. Database
8. Archive directory
9. User directory and Administration rights
10. Permissions

147.1 Encryption algorithms

The Web Server or Application Server may specify encryption algorithms when setting HTTPS parameters. Some recommendations on these algorithms are provided in section HTTPS [p 870]. Password and fields having osd:password as a type store a hash of their value, using the SHA_512 algorithm. This is, notably, the case for the password of users of the default directory.

147.2 HTTPS

It is recommended to use HTTPS for communication with clients (GUI and REST or SOAP). All HTTP traffic should be redirected to HTTPS.

A secure cipher suite and protocols should be used whenever possible. This applies, for example, to Web Servers, Application Servers, and JDBC connections.

TLS v1.2 should be the main protocol, because it is the only version that offers modern authenticated encryption (also known as AEAD).

Several obsolete cryptographic primitives must be avoided:

- Anonymous Diffie-Hellman (ADH) suites do not provide authentication,
- NULL cipher suites provide no encryption,
- Export cipher suites are insecure when negotiated in a connection, but they can also be used against a server that prefers stronger suites (the FREAK attack),
- Suites with weak ciphers (typically of 40 and 56 bits) use encryption that can easily be broken,
- RC4 is insecure,
- 3DES is slow and weak,

On the other hand, being too restrictive on allowed cyphers may prevent some clients from connecting, as they may not be able to negotiate a HTTPS connection.

The following configuration is compatible with browsers supported by EBX®.

- Cipher suites: ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256
- Versions: TLSv1.2

147.3 Installation

Deployed components, such as as Web Server and Application Server, should be installed using a non-root or unprivileged user, and following the principle of least privilege whenever possible. For example, only necessary ports and protocols should be opened.

147.4 Web Server

If you have to expose web applications on the internet, it is a good practice to protect them with a Web Server in a demilitarized zone, while EBX® and the database server can be in a production zone. Consider the following practices for your configuration.

The secure cipher suite and protocols should be set according to the above section HTTPS [p 870].

Do not use the default configuration, and remove any banner that might also expose the version and type of web server.

For example, on Apache2, to remove the banner (default page returned at the root), just remove the folder `/var/www/html`.

Also, on Apache2, to remove headers identifying the Web Server, the value of `ServerTokens` and `ServerSignature` from the file `security.conf` should have the following values:

```
# ServerTokens
# This directive configures what you return as the Server HTTP response
# Header. The default is 'Full' which sends information about the OS-Type
# and compiled in modules.
# Set to one of:  Full | OS | Minimal | Minor | Major | Prod
# where Full conveys the most information, and Prod the least.
ServerTokens Prod

# Optionally add a line containing the server version and virtual host
# name to server-generated pages (internal error documents, FTP directory
# listings, mod_status and mod_info output etc., but not CGI generated
# documents or custom error documents).
# Set to "EMail" to also include a mailto: link to the ServerAdmin.
# Set to one of:  On | Off | EMail
```

```
ServerSignature Off
```

Use the Web Server to set restrictions with HTTP security headers. Note that headers related to the origin impact authorized URLs for all resources returned by EBX®. That includes the content of fields of the URL type (example: image of avatar).

Here is a list of security headers and how to set them for EBX®. First, configure EBX® to not set any HTTP security headers. To do so, set the property `ebx.security.headers.activated` to `false` or unset.

X-XSS-Protection

The `x-xss-protection` header is designed to enable the cross-site scripting (XSS) filter built into modern web browsers. Here is what the header should look like.

```
x-xss-protection: 1; mode=block
```

Enable in Nginx

```
proxy_hide_header x-xss-protection;
add_header x-xss-protection "1; mode=block" always;
```

Enable in Apache2

```
header always unset x-xss-protection
header always set x-xss-protection "1; mode=block"
```

x-Frame-Options

The `x-frame-options` header provides clickjacking protection by not allowing iframes to load on the site. Be aware, this may not be compatible with your configuration if EBX® is integrated through frames for example. Here is what the header should look like:

```
x-frame-options: SAMEORIGIN
```

Enable in Nginx

```
add_header x-frame-options "SAMEORIGIN" always;
```

Enable in Apache2

```
header always set x-frame-options "SAMEORIGIN"
```

X-Content-Type-Options

The `x-content-type-options` header prevents Internet Explorer and Google Chrome from sniffing a response away from the declared content-type. This helps reduce the danger of drive-by downloads and helps treat the content properly. Here is what the header looks like.

```
x-content-type-options: nosniff
```

Enable in Nginx

```
add_header X-Content-Type-Options "nosniff" always;
```

Enable in Apache2

```
header always set X-Content-Type-Options "nosniff"
```

Strict-Transport-Security

The `strict-transport-security` header is a security enhancement that restricts web browsers to access web servers solely over HTTPS. This ensures the connection cannot be established through an insecure HTTP connection which could be vulnerable to attacks. Here is what the header should look like:

```
strict-transport-security: max-age=31536000; includeSubDomains
```

Enable in Nginx

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
```

Enable in Apache2

```
header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"
```

Content-Security-Policy

The `content-security-policy` HTTP header provides an additional layer of security. This policy helps prevent attacks such as Cross Site Scripting (XSS) and other code injection attacks by defining content sources which are approved and thus allowing the browser to load them. Here is what the header shuould look like. Make sure to adapt it with your domain name (`server.company.com` in the example).

```
content-security-policy: default-src 'self'; font-src * data: server.company.com; img-
src * data: server.company.com; script-src * 'unsafe-inline' 'unsafe-eval'; style-src
 * 'unsafe-inline';
```

Enable in Nginx

```
add_header Content-Security-Policy "default-src 'self'; font-src * data:
server.company.com; img-src * data: server.company.com; script-src * 'unsafe-inline'
'unsafe-eval'; style-src * 'unsafe-inline';" always;
```

Enable in Apache2

```
header always set Content-Security-Policy "default-src 'self'; font-src * data:
server.company.com; img-src * data: server.company.com; script-src * 'unsafe-inline'
'unsafe-eval'; style-src * 'unsafe-inline';"
```

Referrer-Policy

The Referrer-Policy HTTP header governs which referrer information should be included with requests made. The Referrer-Policy tells the web browser how to handle referrer information that is sent when a user clicks on a link that leads to another page. Here is what it should look like:

```
Referrer-Policy: strict-origin
```

Enable in Nginx

```
add_header Referrer-Policy: "strict-origin" always;
```

Enable in Apache2

```
header always set Referrer-Policy "strict-origin"
```

Permissions-Policy

The Permissions-Policy HTTP header provides mechanisms for web developers to explicitly declare what functionality can and cannot be used in iframes. A set of "policies" that restrict what APIs the site's code can access or modify is defined and overwrites the browser's default behavior for certain features. Here is what it should look like:

```
Permission-Policy: "*=()"
```

Enable in Nginx

```
add_header Permissions-Policy: "*=()" always;
```

Enable in Apache2

```
Header always set Permissions-Policy "*=()"
```

Please note that this value may change depending on your needs. If your EBX® installation has customized frontend code that uses any relevant Api you may need to adapt it.

147.5 Application Server

As for Web Servers, the same best practice applies: do not expose technical information on the Application Server. For example, for Tomcat, it is recommended to fill the attribute `server` of `connector` in `server.xml` with a generic value as `AppServer`.

```
<Connector port="8080" enableLookups="false" protocol="HTTP/1.1" useBodyEncodingForURI="true"
```

```
            allowBackslash="true" encodedSolidusHandling="passthrough" server="AppServer"/>
```

If the Application Server is exposed through HTTPS, the secure cipher suite and Protocols should be set according to the above section HTTPS [p 870].

If there is a Web Server, it is also recommended to use ports higher than 1024 and let the Web Server do proxy.

If there is no Web Server, security headers should be set by the Application Server as described above.

147.6 Java

It is recommended to follow the security best practices from Oracle. Last supported patches should also be applied as soon as they are available, especially when they include security patches. Consider using the Server JRE for server systems, such as application servers or other long-running back-end processes. The Server JRE is the same as the regular JRE except that it does not contain the web-browser plugins.

EBX® allows a very high level of customization through custom code. All integrated Java modules are considered by EBX® as trusted. Hence, all development on top of EBX® should be reviewed and validated. As an example, developers should not generate HTML from values coming from the database without proper escaping. For more details on this, see the Cross Site Scripting prevention on the OWASP site. Here is a proper escaping example: the name of a store is encoded before being displayed in an HTML form. The `StringEscapeUtils` class included in Apache Commons Lang is used for string encoding.

```
public class StoreMainPane implements UIFormPane
{
 public static final String STORE_NAME_STYLE = "font-weight: bold; padding-top:20px; padding-bottom:20px";

 @Override
 public void writePane(final UIFormPaneWriter writer, final UIFormContext context)
 {

  String storeName = (String) context.getValueContext().getValue(Paths._Store._Name);

  writer.add("<div ").addSafeAttribute("style", STORE_NAME_STYLE).add(">");
  writer.addSafeInnerHTML("Data stored for " + storeName);
  writer.add("</div>");

  // ...
 }
}
```

147.7 Database

Databases should be encrypted at rest and in transit. If there is a private key for encryption, it should not be stored in the same location as the data files. Regarding the JDBC connection, consider configuring the JDBC driver to use SSL/TLS. Contact your database administrator for detailed instructions. You should always use the last supported version or RDBMS, including drivers.

147.8 Archive directory

On the server, the archive directory [p 664] must be properly secured and/or encrypted. Indeed, any archive exported from the EBX® instance will be created there, and these archives are neither encrypted nor protected by password. As a consequence, any user with an access to these files will be able to see the content regardless of any permission defined in EBX®.

147.9 User directory and Administration rights

For production and test platforms, EBX® must be integrated with a custom directory [p 706] to enforce the password policy of your company. The default directory can be used only for development platforms.

According to the Separation of Duties best practice, administrators can manage users and grant access but should not have any functional rights.

147.10 Permissions

Special care is required when defining permissions in EBX®. Persons in charge of this are expected to be aware of the content of the permission documentation [p 487], and especially the information provided in the Important considerations about permissions [p 489] section.

Developer Guide

Introduction

CHAPTER 148

Packaging TIBCO EBX® modules

An EBX® module is a standard Jakarta EE web application, packaging various resources such as XML Schema documents, Java classes and static resources.

Since EBX® modules are web applications they benefit from features such as class-loading isolation, WAR or EAR packaging, and Web resources exposure.

This chapter contains the following topics:

1. Module structure
2. Module declaration
3. Module registration
4. Packaged resources

148.1 Module structure

An EBX® module contains the following files:

`/WEB-INF/ebx/module.xml`	This mandatory document defines the main properties and services of the module. See Module declaration [p 880].
`/WEB-INF/web.xml`	This is the standard Jakarta EE deployment descriptor. It can perform the registration of the EBX® module when the application server is launched. See Module registration [p 880].
`/META-INF/MANIFEST.MF`	Optional. If present, EBX® reports the 'Implementation-Title' and 'Implementation-Version' values to *Administration > Technical configuration > Modules and data models*.
`/www/`	This optional directory contains all packaged resources, which are accessible via public URL. See Packaged resources [p 882].

148.2 Module declaration

A module is declared using the document `/WEB-INF/ebx/module.xml`. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:ebx-schemas:module_2.4"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ebx-schemas:module_2.4 https://schema.orchestranetworks.com/module_2.4.xsd">
 <name>moduleTest</name>
</module>
```

See the associated schema for documentation about each property. The main properties are as follows:

Element	Description	Required
`name`	Defines the unique identifier of the module in the server instance. The module name usually corresponds to the name of the web application (the name of its directory).	Yes.
`publicPath`	Defines a path other than the module's name identifying the web application in public URLs. This path is added to the URL of external resources of the module when computing absolute URLs. If this field is not defined, the public path is the module's `name`, defined above.	No.
`services`	Declares user services using the legacy API. See *Declaration and configuration* of legacy user services. From the version 5.8.0, it is strongly advised to use the new user services [p 1027].	No.
`beans`	Declares reusable Java bean components. See the workflow package [p 1013].	No.
`ajaxComponents`	Declares Ajax components. See **Declaring an Ajax component in a module** `UIAjaxComponent.declareInModule`[API] in the Java API.	No.

148.3 Module registration

In order to be identifiable by EBX®, a module must be registered at runtime when the application server is launched. For a web application, every EBX® module must:

- contain a Java class with the annotation `@WebListener` extending the class `ModuleRegistrationListener`[API].

> **Attention**
> When using the `@WebListener` annotation, ensure that the application server is configured to activate the servlet 3.0 annotation scanning for the web application. See JSR 315: JavaTM Servlet 3.0 Specification for more information.

or:

- contain a Servlet extending the class `ModuleRegistrationServlet`[API];
- make a standard declaration of this servlet in the deployment descriptor `/WEB-INF/web.xml`;

- ensure that this servlet will be registered at server startup by adding the following standard element to the deployment descriptor: `<load-on-startup>1</load-on-startup>`.

Additional recommendations and information:

- The method `handleRepositoryStartup` in `ModuleRegistrationServlet`[API] allows setting the logger associated with the module and defining additional behavior such as common JavaScript and CSS resources.
- The specific class extending `ModuleRegistrationServlet` must be located in the web application (under `/WEB-INF/classes` or `/WEB-INF/lib`; due to the fact that this class is internally used as a hook to the application's class-loader, to load Java classes used by the data models associated with the module).
- The application server startup process is asynchronous and web applications / EBX® modules are discovered dynamically. The EBX® repository initialization depends on this process and will wait for the registration of all used modules up to an unlimited amount of time. As a consequence, if a used module is not deployed for any reason, it must be declared in the EBX® main configuration file. For more information, see the property Declaring modules as undeployed [p 598].
- All module registrations and unregistrations are logged in the `log.kernel` category.
- If an exception occurs while loading a module, the cause is written in the application server log.
- Once the servlet is out of service, the module is unregistered and the data models and associated datasets become unavailable. Note that hot deployment/undeployment is not supported [p 525].

Deployment descriptor example

Here is an example of a Jakarta EE deployment descriptor (`/WEB-INF/web.xml`):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-
app_6_0.xsd"
      version="6.0">
 <servlet>
    <servlet-name>InitEbxServlet</servlet-name>
    <servlet-class>com.foo.RegisterServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
 </servlet>
</web-app>
```

Registration example

Here is an implementation example of the `ModuleRegistrationServlet`:

```
package com.foo;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import com.onwbp.base.repository.*;

public class RegisterServlet extends ModuleRegistrationServlet
{

 public void handleRepositoryStartup(ModuleContextOnRepositoryStartup aContext)
  throws OperationException
 {
  // Perform module-specific initializations here
  ...

  // Declare custom resources here
  aContext.addExternalStyleSheetResource(MyCompanyResources.COMMON_STYLESHEET_URL);
  aContext.addExternalJavaScriptResource(MyCompanyResources.COMMON_JAVASCRIPT_URL);

  aContext.addPackagedStyleSheetResource("myModule.css");
  aContext.addPackagedJavaScriptResource("myModule.js");
```

```
 }

 public void handleRepositoryShutdown()
 {
  // Release resources of the current module when the repository is shut down here
  ...
 }

  public void destroyBeforeUnregisterModule()
 {
  // Perform operations when this servlet is being taken out of service here
  ...
 }

}
```

148.4 Packaged resources

The packaged resources are files and documents that can be directly accessed from client browsers and can be managed and specified either as `osd:resource` fields or via the Java API. They have various types and can also be localized.

See also

`ResourceType`[API]

Type `osd:resource` [p 904]

Directory structure

The packaged resources must be located under the following directory structure:

1. On the first level, the directory `/www/` must be located at the root of the module (web application).
2. On the second level, the directory must specify the localization. It can be:
   - `common/` should contain all the resources to be used by default, either because they are locale-independent or as the default localization (in EBX®, the default localization is `en`, namely English);
   - `{lang}/` when localization is required for the resources located underneath, with `{lang}` to be replaced by the actual locale code; it should correspond to the locales supported by EBX®; for more information, see Configuring EBX® localization [p 581].
3. On the third level, the directory must specify the resource type. It can be:
   - `jscripts/` for JavaScript resources;
   - `stylesheets/` for Cascading Style Sheet (CSS) resources;
   - `html/` for HTML resources;
   - `icons/` for icon typed resources;
   - `images/` for image typed resources.

Example

In this example, the image `logoWithText.jpg` is the only resource that is localized:

```
/www
 ├── common
 │   ├── images
 │   │   ├── myCompanyLogo.jpg
 │   │   └── logoWithText.jpg
 │   ├── jscripts
 │   │   └── myCompanyCommon.js
 │   └── stylesheets
```

```
│       └── myCompanyCommon.css
├── de
│   └── images
│       └── logoWithText.jpg
└── fr
    └── images
        └── logoWithText.jpg
```

CHAPTER 149

Mapping to Java

This chapter contains the following topics:

1. How to access data from Java?
2. Transactions and concurrency
3. Mapping of data types
4. Java bindings

149.1 How to access data from Java?

Read access

Data can be read from various generic Java classes, mainly `Adaptation`[API] and `ValueContext`[API]. The getter methods for these classes return objects that are typed according to the mapping rules described in the section Mapping of data types [p 887].

Write access

Data updates must be performed in a well-managed context:

- In the context of a procedure execution, by calling the methods `setValue...` of the interface `ValueContextForUpdate`[API], or
- During the user input validation, by calling the method `setNewValue` of the class `ValueContextForInputValidation`[API].

Modification of mutable objects

According to the mapping that is described in the Mapping of data types [p 887] section, some accessed Java objects are mutable objects. These are instances of `List` , `Date` or any JavaBean. Consequently, these objects can be locally modified by their own methods. However, such modifications will remain local to the returned object unless one of the above setters is invoked and the current transaction is successfully committed.

149.2 Transactions and concurrency

Concurrency

At the dataspace level	In a single dataspace, the system supports running only one single read-write `Procedure` and multiple concurrent `ReadOnlyProcedures`. Concurrent accesses outside any `Procedure` are also supported.
At the repository level	At the repository level, concurrency is limited for only some specific operations. For example (non-exhaustive list): • A data model publication excludes many operations. • A dataspace merge excludes write operations on the two dataspaces involved in the merge.

Query snapshot isolation

The following table defines the properties related to query isolation. Note that a *query result* is represented in the Java API by either a `QueryResult`[API] or a `RequestResult`[API]:

Queries outside of a `Procedure`	Data is frozen at the time of fetching the query result. More precisely, a query result accesses only committed data as of the last committed transaction at the time of fetching this result. The content of this result never changes afterwards. A query outside of a `Procedure` can be considered as a self-containing `ReadOnlyProcedure`.
Queries inside of a `Procedure`, in the same dataspace as the `Procedure` underlying dataspace	The query result reflects the last committed state before the Procedure starts and the changes that occurred in the `Procedure` previously to the query result fetch. The content of this result never changes afterwards, whatever happens in the `Procedure`.
Queries inside of a `Procedure`, in another dataspace	The consistency is guaranteed at the repository level, so the query result reflects the last committed state before the `Procedure` starts. The content of this result never changes after the query is fetched, whatever happens in the whole repository.

Adaptation objects

In Java, a persistent dataset or a persistent record are both represented by an instance of the `Adaptation` class.

The following table defines the properties related to `Adaptation` objects.

Immutability	An `Adaptation` object instance is immutable. Therefore, the client code should not "hold" an `Adaptation` object for a long time (in particular beyond a transaction boundaries). However, it is possible to invoke the method `Adaptation.getUpToDateInstance`[API].
Fetch	If an `Adaptation` is fetched from a `QueryResult`, then the snapshot isolation rules described in the previous section apply. Otherwise, if an `Adaptation` is fetched from a running `Procedure`, it reflects the last committed state before the Procedure starts. Otherwise, outside of a `QueryResult` or a running `Procedure`, the `Adaptation` reflects the state of the record on its fetch-time. **See also** *`AdaptationHome.findAdaptationOrNull`*[API] *`AdaptationTable.lookupAdaptationByPrimaryKey`*[API]

149.3 Mapping of data types

This section describes how XML Schema type definitions and element declarations are mapped to Java types.

Simple data types

Basic rules for simple data types

Each XML Schema simple type corresponds to a Java class; the mapping is documented in the table XML Schema built-in simple types [p 900].

See also *`SchemaNode.createNewOccurrence`*[API]

Multiple cardinality on a simple element

If the attribute `maxOccurs` is greater than 1, then the element is an aggregated list and the corresponding instance in Java is an instance of `java.util.List`.

Elements of the list are instances of the Java class that is determined from the mapping of the simple type (see previous section).

Complex data types

Complex type definitions without a class declaration

By default (no attribute `osd:class`), a terminal node of a complex type is instantiated using an internal class. This class provides a generic JavaBean implementation. However, if a custom client Java code

must access these values, use a custom JavaBean. To do so, use the `osd:class` declaration described in the next section.

You can transparently instantiate, read and modify the mapped Java object, with or without the attribute `osd:class`, by invoking the methods `SchemaNode.createNewOccurrence`[API], `SchemaNode.executeRead`[API] and `SchemaNode.executeWrite`[API].

Mapping of complex types to custom JavaBeans

You can map an XML Schema complex type to a custom Java class. This is done by adding the attribute `osd:class` to the complex node definition. Unless the element has `xs:maxOccurs > 1`, you must also specify the attribute `osd:access` for the node to be considered a *terminal* node. If the element has `xs:maxOccurs > 1`, it is automatically considered to be terminal.

The custom Java class must conform to the JavaBean protocol. This means that each child of the complex type must correspond to a JavaBean property of the class. Additionally, each JavaBean property must be a read-write property, and its implementation must ensure that the value set by the setter method is returned, as-is, by the getter method. Contextual computations are not allowed in these methods.

Example

In this example, the Java class `com.carRental.Customer` must define the methods `getFirstName()` and `setFirstName(String)`.

A JavaBean can have a custom user interface within TIBCO EBX®, by using a `UIBeanEditor`[API].

```
<xs:element name="customer" osd:access="RW">
  <xs:complexType name="subscriber" osd:class="com.carRental.Customer">
 <xs:sequence>
   <xs:element name="firstName" type="xs:string"/>
   ...
 </xs:sequence>
  </xs:complexType>
</xs:element>
```

Multiple cardinality on a complex element

If the attribute `maxOccurs` is greater than 1, then the corresponding instance in Java is:

- An instance of `java.util.List` for an aggregated list, where every element in the list is an instance of the Java class determined by the mapping of simple types [p 900], or
- An instance of `AdaptationTable`[API], if the property `osd:table` is specified.

149.4 Java bindings

Java bindings support generating Java types that reflect the structure of the data model. The Java code generation can be done in the user interface. See Generating Java bindings [p 891].

Benefits

Ensuring the link between XML Schema structure and Java code provides a number of benefits:

- **Development assistance:** Auto-completion when you type an access path to parameters, if it is supported by your IDE.
- **Access code verification:** All accesses to parameters are verified at code compilation.
- **Impact verification:** Each modification of the data model impacts the code compilation state.

- **Cross-referencing:** By using the reference tools of your IDE, you can easily verify where a parameter is used.

Consequently, it is strongly encouraged that you use Java bindings.

XML declaration

The specification of the Java types to be generated from the data model is included in the main schema.

Each binding element defines a generation target. It must be located at, in XPath notation, `xs:schema/xs:annotation/xs:appinfo/ebxbnd:binding`, where the prefix `ebxbnd` is a reference to the namespace identified by the URI `urn:ebx-schemas:binding_1.0`. Several binding elements can be defined if you have different generation targets.

The attribute `targetDirectory` of the element `ebxbnd:binding` defines the root directory used for Java type generation. Generally, it is the directory containing the project source code, `src`. A relative path is interpreted based on the current runtime directory of the VM, as opposed to the XML schema.

See bindings XML Schema.

XML bindings example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:ebxbnd="urn:ebx-schemas:binding_1.0">
 <xs:annotation>
  <xs:appinfo>
   <!-- The bindings define how this schema will be represented in Java.
   Several <binding> elements may be defined, one for each target. -->
   <ebxbnd:binding
    targetDirectory="../_ebx-demos/src-creditOnLineStruts-1.0/">
    <javaPathConstants typeName="com.creditonline.RulesPaths">
     <nodes root="/rules" prefix="" />
    </javaPathConstants>
    <javaPathConstants typeName="com.creditonline.StylesheetConstants">
     <nodes root="/stylesheet" prefix="" />
    </javaPathConstants>
   </ebxbnd:binding>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:schema>
```

Java constants can be defined for XML schema paths. To do so, generate one or more interfaces from a schema node, including the root node `/`. The example generates two Java path constant interfaces, one from the node `/rules` and the other from the node `/stylesheet` in the schema. Interface names are described by the element `javaPathConstants` with the attribute `typeName`. The associated node is described by the element `nodes` with the attribute `root`.

CHAPTER 150

Tools for Java developers

TIBCO EBX® provides Java developers with tools to facilitate use of the EBX® API, as well as integration with development environments.

This chapter contains the following topics:

1. Activating the development tools
2. Data model refresh tool
3. Generating Java bindings
4. Path to a node
5. Web component link generator

150.1 Activating the development tools

To activate the development tools, run EBX® in *development mode*. This is specified in the EBX® main configuration file EBX® run mode [p 599] using the property `backend.mode=development`.

150.2 Data model refresh tool

When editing the data model directly as an XML Schema document without using the data-modeling tool provided by EBX®, you can refresh it without restarting the application server.

In the 'Administration' area, select **Select > Technical configuration > Development tools > Refresh updated data models** (or **Refresh all data models**).

> **Attention**
> Since the operation is critical regarding data consistency, refreshing the data models acquires a global exclusive lock on the repository. This means that most other operations (data access and update, validation, etc.) will wait until the completion of the data model refresh.

150.3 Generating Java bindings

The Java types specified by Java bindings can be generated from a dataset or a data model, by selecting **Actions > Generate Java** in the navigation pane.

See also *Java bindings [p 888]*

150.4 **Path to a node**

The field 'Data path' is displayed in the documentation pane of a node. This field indicates the path to the node, which can be useful when writing XPath formulas.

> **Note**
>
> This field is always available to administrators.

150.5 **Web component link generator**

The 'Web component link generator' service is a user interface designed to create HTTP requests that call EBX® web components. To launch this service, select **Actions > Web component link generator** in the navigation pane.

CHAPTER 151

Terminology changes

A new TIBCO EBX® release can introduce new vocabulary for users. To preserve the backward compatibility, these terminology changes do not usually impact the API. Consequently, Java class names, method names, data services operation names, etc. still use the older version terminology. This chapter purpose is to facilitate the correspondence of the old term in the API to the new terms.

See also *Glossary* *[p 33]*

This chapter contains the following topics:

1. Terminology changes in version 5.9
2. Terminology changes in version 5.0

151.1 Terminology changes in version 5.9

New term	Term prior to version 5.9.0
D3 primary node	D3 master node
D3 replica node	D3 slave node

151.2 Terminology changes in version 5.0

The following table summarizes the mappings between the version 5.0.0 terminology and previous terminology:

New term	Term prior to version 5.0.0
Dataset	Adaptation instance
Child dataset	Child adaptation instance
Data model	Data model
Dataspace	Branch
Snapshot	Version
Dataspace or snapshot	Home
Data Workflow	Workflow instance
Workflow model	Workflow definition
Workflow publication	Workflow
Field	Attribute
Inherited field	Inherited attribute
Record	Record/occurrence
Validation rule	Constraint
Simple/advanced control	Simple/advanced constraint

Data model

CHAPTER 152

Introduction

A data model is a structural definition of the data to be managed in the TIBCO EBX® repository. Data models contribute to EBX®'s ability to guarantee the highest level of data consistency and to facilitate data management.

Specifically, the data model is a document that conforms to the XML Schema standard (W3C recommendation). Its main features are as follows:

- A rich library of well-defined simple data types [p 899], such as integer, boolean, decimal, date, time;
- The ability to define additional simple types [p 901] and complex types [p 901];
- The ability to define simple lists of items, called aggregated lists [p 910];
- Validation constraints [p 935] (facets), for example: enumerations, uniqueness constraints, minimum/maximum boundaries.

EBX® also uses the extensibility features of XML Schema for other useful information, such as:

- Predefined types [p 902], for example: locale, resource, html;
- Definition of tables [p 913] and foreign key constraints [p 919];
- Mapping data in EBX® to Java beans;
- Advanced validation constraints [p 935] (extended facets), such as dynamic enumerations;
- Extensive presentation information [p 955], such as labels, descriptions, and error messages.

> **Note**
>
> EBX® supports a subset of the W3C recommendations, as some features are not relevant to Master Data Management.

This chapter contains the following topics:

1. Editing the data model
2. References
3. Relationship between datasets and data models
4. Pre-requisite for XML Schemas
5. Conventions
6. Schemas with reserved names

152.1 Editing the data model

There are two different ways to define a data model:

- The data model can be defined using an XML Schema editor or through the data model assistant. The data model assistant has the advantage of being integrated into the EBX® user interface, abstracting the verbose underlying XML. For more information, see Introduction to data models [p 44].

  The data model assistant allows using features that are not documented to be used outside of the DMA; e.g. Toolbars and Widgets.
- By using an external XML Schema document editor.

152.2 References

For an introduction to XML Schema, see the W3Schools XML Schema Tutorial.

See also

XML Schema Part 0: Primer

XML Schema Part 1: Structures

XML Schema Part 2: Datatypes

152.3 Relationship between datasets and data models

Each root dataset is associated with a single data model. At the dataspace creation, an associated data model is selected, on which to base the dataset.

See also *Creating a dataset [p 125]*

152.4 Pre-requisite for XML Schemas

In order for an XML Schema to be accepted by EBX®, it must include a global element declaration that includes the attribute `osd:access="--"`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
   <xs:import namespace="urn:ebx-schemas:common_1.0"
     schemaLocation="https://schema.orchestranetworks.com/common_1.0.xsd"/>
   <xs:element name="root" osd:access="--">
  ...
  </xs:element>
</xs:schema>
```

152.5 Conventions

By convention, namespaces are always defined as follows:

Prefix	Namespace
`xs:`	`http://www.w3.org/2001/XMLSchema`
`osd:`	`urn:ebx-schemas:common_1.0`
`fmt:`	`urn:ebx-schemas:format_1.0`
`usd:`	`urn:ebx-schemas:userServices_1.0`
`emd:`	`urn:ebx-schemas:entityMappings_1.0`

152.6 Schemas with reserved names

Several data models in EBX® have reserved names.

All references to other data models (using the attribute `schemaLocation` for an import, include or redefine) that end with one of the following strings are reserved:

- common_1.0.xsd
- org_1.0.xsd
- coreModel_1.0.xsd
- session_1.0.xsd
- userServices_1.0.xsd
- entityMappings_1.0.xsd

These XSD files correspond to the schemas provided for the module `ebx-root-1.0`, at the path `/WEB-INF/ebx/schemas`. The attribute `schemaLocation` can reference the files at this location or a copy, if the file names are identical. This is useful if you want to avoid a module dependency on `ebx-root-1.0`.

For security reasons, EBX® uses an internal definition for these schemas to prevent any modification.

CHAPTER 153

Data types

This chapter details the data types supported by TIBCO EBX®.

See also *Tables and relationships* *[p 913]*

This chapter contains the following topics:

1. XML Schema built-in simple types
2. XML Schema named simple types
3. XML Schema complex types
4. Extended simple types defined by EBX®
5. Complex types defined by EBX®
6. Aggregated lists
7. Including external data models

153.1 XML Schema built-in simple types

The table below lists all the simple types defined in XML Schema that are supported by EBX®, along with their corresponding Java types.

XML Schema type	Java class	Notes
`xs:string`	`java.lang.String`	The default strategies defined for string fields are detailed in the section Default strategy for string fields [p 514].
`xs:boolean`	`java.lang.Boolean`	
`xs:decimal`	`java.math.BigDecimal`	A `totalDigits` facet with a value equal to 15 is added by default to decimal fields that are contained in a mapped table (historized or replicated table). However, this facet can be overwritten with a greater value in the data model.
`xs:dateTime`	`java.util.Date`	
`xs:time`	`java.util.Date`	The date portion of the returned `Date` is always set to '1970/01/01'.
`xs:date`	`java.util.Date`	The time portion of the returned `Date` is always the beginning of the day, that is, '00:00:00'.
`xs:anyURI`	`java.net.URI`	
`xs:Name` (`xs:string` restriction)	`java.lang.String`	
`xs:int` (`xs:decimal` restriction)	`java.lang.Integer`	
`xs:integer` (`xs:decimal` restriction)	`java.lang.Integer`	This mapping does not comply with the XML Schema recommendation. Although the XML Schema specification states that `xs:integer` has no value space limitation, this value space is, in fact, restricted by the Java specifications of the `java.lang.Integer` object.

The mapping between XML Schema types and Java types are detailed in the section Mapping of data types [p 887].

153.2 XML Schema named simple types

Named simple types can be defined when designing a data model for redefining an existing built-in simple type. A *named simple type* can be reused in the data model.

Restrictions:

- In the data model, only the element `restriction` is allowed in a named simple type, and even then, only derivation by restriction is supported. Notably, the elements `list` and `union` are not supported.
- Facet definition is not cumulative. That is, if an element and its named type both define the same kind of facet, then the facet defined in the type is overridden by the local facet definition.

  However, this restriction does not apply to programmatic facets defined by the element `osd:constraint`. For `osd:constraint`, if an element and its named type both define a programmatic facet with different Java classes, the definition of these facets will be cumulative.

  Contrary to the XML Schema Specification, EBX® is not strict regarding the definition of a facet of the same kind in an element and its named type. That is, the value of a same kind of facet defined in an element is not checked according to the one defined in the named type. However, in the case of static enumerations defined both in an element and its type, the local enumeration will be replaced by the intersection between these enumerations.
- It is not possible to define different types of enumerations on both an element and its named type. For instance, you cannot specify a static enumeration in an element and a dynamic enumeration in its named type.
- It is not possible to simultaneously define a pattern facet in both an element and its named type.

153.3 XML Schema complex types

Complex types can be defined when designing a data model. A *named complex type* can be reused in the data model.

Restrictions:

- In the data model, only the element `sequence` is allowed. Notably, attribute definition is not supported.
- Type extensions are not supported in the current version of EBX®.

153.4 Extended simple types defined by EBX®

EBX® provides pre-defined simple data types:

XML Schema type	Java class
`osd:text` (`xs:string` restriction)	`java.lang.String`
`osd:html` (`xs:string` restriction)	`java.lang.String`
`osd:email` (`xs:string` restriction)	`java.lang.String`
`osd:password` (`xs:string` restriction)	`java.lang.String`
`osd:color` (`xs:string` restriction)	`java.lang.String`
`osd:resource` (`xs:anyURI` restriction)	internal class
`osd:locale` (`xs:string` restriction)	`java.util.Locale`
`osd:dataspaceKey` (`xs:string` restriction)	`java.lang.String`
`osd:datasetName` (`xs:string` restriction)	`java.lang.String`

The above types are defined by the internal schema `common-1.0.xsd`. They are defined as follows:

`osd:text`

This type represents textual information. Its default user interface in EBX® consists of a dedicated editor with several lines for input and display.

The use of this data type is intended for very long texts; this is why the `'Text'` search strategy is associated with this type, and cannot be modified. As a consequence, fields of type `osd:text` cannot benefit from sorting; moreover, their indexing strategy and search results differ from those of the `xs:string` type. See Default strategy for string fields [p 514] for more information.

```
<xs:simpleType name="text">
 <xs:restriction base="xs:string" />
</xs:simpleType>
```

`osd:html`

This represents a character string with HTML formatting. A WYSIWYG editor is provided in EBX®.

```
<xs:simpleType name="html">
 <xs:restriction base="xs:string" />
</xs:simpleType>
```

`osd:email`

This represents an email address as specified by the RFC822 standard.

```
<xs:simpleType name="email">
 <xs:restriction base="xs:string" />
</xs:simpleType>
```

`osd:password`

This represents a hashed or encrypted password. A specific editor is provided in EBX®.

```
<xs:element name="password" type="osd:password" />
```

The default editor performs a hash computation using the SHA-512 algorithm. This encryption function is also available from a Java client using the method `DirectoryDefault.encryptString`[API].

It is also possible for the default editor to use a different encryption mechanism by specifying a class that implements the interface `Encryption`[API].

```
<xs:element name="password" type="osd:password">
 <xs:annotation>
  <xs:appinfo>
   <osd:uiBean class="com.orchestranetworks.ui.UIPassword">
    <encryptionClass>package.EncryptionClassName</encryptionClass>
   </osd:uiBean>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

It is possible to specify some salt by referencing a path to another field, and by using a class the implements the interface `HashComputation`[API].

```
<xs:element name="password" type="osd:password">
 <xs:annotation>
  <xs:appinfo>
   <osd:uiBean class="com.orchestranetworks.ui.UIPassword">
    <encryptionClass>package.HashClassName</encryptionClass>
    <saltPath>../login</saltPath>
   </osd:uiBean>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

`osd:locale`

This represents a geographical, political or cultural location.

The locale type is translated into Java by the class `java.util.Locale`.

```
<xs:simpleType name="locale">
 <xs:restriction base="xs:string">
  <xs:enumeration value="ar" osd:label="Arabic" />
  <xs:enumeration value="ar_AE" osd:label="Arabic (United Arab
 Emirates)" />
  <xs:enumeration value="ar_BH" osd:label="Arabic (Bahrain)" />
  <xs:enumeration value="ar_DZ" osd:label="Arabic (Algeria)" />
  <xs:enumeration value="ar_EG" osd:label="Arabic (Egypt)" />
  <xs:enumeration value="ar_IQ" osd:label="Arabic (Iraq)" />
   ...
  <xs:enumeration value="vi_VN" osd:label="Vietnamese (Vietnam)" /
>
  <xs:enumeration value="zh" osd:label="Chinese" />
  <xs:enumeration value="zh_CN" osd:label="Chinese (China)" />
  <xs:enumeration value="zh_HK" osd:label="Chinese (Hong Kong)" />
  <xs:enumeration value="zh_TW" osd:label="Chinese (Taiwan)" />
 </xs:restriction>
</xs:simpleType>
```

`osd:color`

This represents a character string with hexadecimal RGB color formatting. A color picker `UIComponent` is provided in EBX®.

```
<xs:simpleType name="color">
 <xs:restriction base="xs:string" />
</xs:simpleType>
```

`osd:resource`

This represents a resource packaged in a module. For more information, see Packaged resources [p 882]. This type requires the definition of the facet FacetOResource [p 940].

```
<xs:simpleType name="resource">
 <xs:restriction base="xs:anyURI" />
</xs:simpleType>
```

`osd:dataspaceKey`

This type represents a reference to a dataspace.

```
<xs:element name="dataspaceField" type="osd:dataspaceKey" />
```

A specific editor is provided in EBX® that displays the dataspaces that can be referenced.

It is possible to specify the dataspaces that can be referenced using the element `osd:dataspaceSet` under `xs:annotation/xs:appInfo`. If the element `osd:dataspaceSet` is not defined, then by default, only open branches can be referenced.

```
<xs:element name="dataspaceField" type="osd:dataspaceKey">
 <xs:annotation>
  <xs:appinfo>
```

```
   <osd:dataspaceSet>
    <include>
     <pattern>a pattern</pattern>
     <type>all | branch | version</type>
     <includeDescendants>none | allDescendants |
 allBranchDescendants | allSnapshotDescendants | branchChildren |
 snapshotChildren</includeDescendants>
    </include>
    <exclude>
     <pattern>a pattern</pattern>
     <type>all | branch | version</type>
     <includeDescendants>none | allDescendants |
 allBranchDescendants | allSnapshotDescendants | branchChildren |
 snapshotChildren</includeDescendants>
    </include>
    <filter osd:class="com.foo.MyDataspaceFilter">
     <param1>...</param1>
     <param2>...</param2>
    </filter>
   </osd:dataspaceSet>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

- includes

  Specifies the dataspaces that can be referenced by this field. An `include` must at least be defined.

  `pattern`: Specifies a pattern that filters dataspaces. The pattern is checked against the name of the dataspaces. This property is mandatory.

  `type`: Specifies the type of dataspaces that can be referenced by this field. If not defined, this restriction is applied to branches. If `all` then branches and snapshots are included. If `branch` then only branches are included. If `snapshot` then only snapshots are included. If not set, this property is `branch` by default.

  `includeDescendants`: Specifies if children or descendants of the dataspaces that match the specified pattern are included in the set. If `none` then neither children nor descendants of the dataspaces that match the specified pattern are included. If `allDescendants` then all descendants of the dataspaces that match the specified pattern are included. If `allBranchDescendants` then all descendant branches of the dataspaces that match the specified pattern are included. If `allSnapshotDescendants` then all descendant snapshots of the dataspaces that match the specified pattern are included. If `directBranchChildren` then only direct branches of the dataspaces that match the specified pattern are included. If `directSnapshotChildren` then only direct snapshots of the dataspaces that match the specified pattern are included. If not set, this property is `none` by default.

- excludes

  Specifies the dataspaces that cannot be referenced by this field. Excludes are ignored if no includes are defined.

`pattern`: Specifies a pattern that filters dataspaces. The pattern is checked against the name of the dataspaces. This property is mandatory.

`type`: Specifies the type of dataspaces that can be referenced by this field. If not defined, this restriction is applied to branches. If `all` then branches and snapshots are excluded. If `branch` then only branches are excluded. If `snapshot` then only snapshots are excluded. If not set, this property is `branch` by default.

`includeDescendants`: Specifies if children or descendants of the datasets that match the specified pattern are excluded from the set. If `none` then neither children nor descendants of the dataspaces that match the specified pattern are excluded. If `allDescendants` then all descendants of the dataspaces that match the specified pattern are excluded. If `allBranchDescendants` then all descendant branches of the dataspaces that match the specified pattern are excluded. If `allSnapshotDescendants` then all descendant snapshots of the dataspaces that match the specified pattern are excluded. If `directBranchChildren` then only direct branches of the dataspaces that match the specified pattern are excluded. If `directSnapshotChildren` then only direct snapshots of the dataspaces that match the specified pattern are excluded. If not set, this property is `none` by default.

- `filter`

  Specifies a filter to accept or reject dataspaces in the context of a dataset or a record. This filter is only used in the dedicated input component that is associated to this field. That is, this filter is not used when validating this field. A specific constraint can be used to perform specific controls on this field.

  The attribute `osd:class` specifies a Java bean that implements the interface `DataspaceSetFilter`[API].

It is also possible to customize validation messages and the control policy associated with this type using the element `validation` under `xs:annotation/xs:appInfo/osd:dataspaceSet`. See Facet validation message with severity [p 958] and Control policy [p 945] for more information.

`osd:datasetName`

This type represents a reference to a dataset.

```
<xs:element name="dataset" type="osd:datasetName" />
```

A specific editor provided in EBX® displays the datasets that can be referenced.

It is also possible to specify the datasets that can be referenced using the element `osd:datasetSet` under `xs:annotation/xs:appInfo`:

```
<xs:element name="datasetField" type="osd:datasetName">
 <xs:annotation>
  <xs:appinfo>
   <osd:datasetSet>
    <branch>productsBranch</branch>
    <version>productsVersion</version>
    <dataspaceSelector>../dataspaceField</dataspaceSelector>
    <pattern>a pattern</pattern>
    <filter osd:class="com.foo.MyDatasetFilter">
     <param1>...</param1>
     <param2>...</param2>
    </filter>
   </osd:datasetSet>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

- `branch`

  Specifies the source branch. Only datasets contained in this branch will be able to be selected by a field of the type Dataset identifier (osd:datasetName).

- `version`

  Specifies the source snapshot. Only datasets contained in this snapshot will be able to be selected by a field of the type Dataset identifier (osd:datasetName).

- `dataspaceSelector`

  Specifies a field in the same data model that defines the dataspace containing the datasets that can be referenced. The specified field must be of type `xs:string` or `osd:dataspaceKey`. The value of this field must comply with the representation of a persistent identifier of a dataspace or snapshot.

  The referred node must respect the restrictions existing for dynamic facets.

  See also

  `HomeKey.format`[API]

  Dynamic constraints [p 939]

- `includes`

  Specifies the datasets that can be referenced by this field.

  `pattern`: Specifies a pattern that filters datasets. The pattern is checked against the name of the datasets. This property is mandatory.

  `includeDescendants`: Specifies if children or descendants of the datasets that match the specified pattern are included in the set. If `none` then neither children nor descendants of the datasets that match the specified pattern are excluded. If `directChildren`

then only direct children of the datasets that match the specified pattern are excluded. If `allDescendants` then all descendants of the datasets that match the specified pattern are excluded. If not set, this property is `none` by default.

- `excludes`

  Specifies the datasets that cannot be referenced by this field. Excludes are ignored if no includes are defined.

  `pattern`: Specifies a pattern that filters datasets. The pattern is checked against the name of the datasets. This property is mandatory.

  `includeDescendants`: Specifies if children or descendants of the datasets that match the specified pattern are included in the set. If `none` then neither children nor descendants of the datasets that match the specified pattern are excluded. If `directChildren` then only direct children of the datasets that match the specified pattern are excluded. If `allDescendants` then all descendants of the datasets that match the specified pattern are excluded. If not set, this property is `none` by default.

- `filter`

  Specifies a filter to accept or reject datasets in the context of a dataset or record. This filter is only used in the dedicated input component that is associated to this field. That is, this filter is not used when validating this field. A specific constraint can be used to perform specific controls on this field.

  The attribute `osd:class` specifies a Java bean that implements the interface `DatasetSetFilter`[API]. A validation message is added to the associated field if an input dataspace reference does not match this filter.

One of the elements `branch`, `version` or `dataspaceSelector` must be defined.

It is also possible to customize validation messages and the control policy associated with this type using the element `validation` under `xs:annotation/xs:appInfo/osd:datasetSet`. See Facet validation message with severity [p 958] and Control policy [p 945] for more information.

153.5 Complex types defined by EBX®

EBX® provides pre-defined complex data types:

XML Schema type	Description
`osd:UDA`	**User Defined Attribute:** This type allows any user, according to their access rights, to define a value associated with an attribute defined in a dictionary called a UDA Catalog.
`osd:UDACatalog`	**Catalog of User Defined Attributes:** This type consists of a table in which attributes can be specified. This catalog is used by all `osd:UDA` elements declared in the same data model.

`osd:UDA`

A User Defined Attribute (UDA) supports both the `minOccurs` and `maxOccurs` attributes, as well as the attribute `osd:UDACatalogPath`, which specifies the path of the corresponding catalog.

```
<xs:element name="firstUDA" type="osd:UDA" minOccurs="0"
 maxOccurs="unbounded" osd:UDACatalogPath="//insuranceCatalog" />
<xs:element name="secondUDA" type="osd:UDA" minOccurs="1"
 maxOccurs="1"
 osd:UDACatalogPath="/root/userCatalog" />
<xs:element name="thirdUDA" type="osd:UDA" minOccurs="0"
 maxOccurs="1"
 osd:UDACatalogPath="//userCatalog" />
```

In the manager, when working with a UDA, the editor will adapt itself to the type of the selected attribute.

`osd:UDACatalog`

Internally, a catalog is represented as a table. The parameters `minOccurs` and `maxOccurs` must be specified.

Several catalogs can be defined in the same data model.

```
<xs:element name="insuranceCatalog" type="osd:UDACatalog"
 minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:documentation xml:lang="en-US">Insurance Catalog.</
xs:documentation>
  <xs:documentation xml:lang="fr-FR">Catalog assurance.</
xs:documentation>
 </xs:annotation>
</xs:element>
<xs:element name="userCatalog" type="osd:UDACatalog" minOccurs="0"
 maxOccurs="unbounded">
 <xs:annotation>
  <xs:documentation xml:lang="en-US">User catalog.</
xs:documentation>
  <xs:documentation xml:lang="fr-FR">Catalogue utilisateur.</
xs:documentation>
 </xs:annotation>
</xs:element>
```

Only the following types are available for creating new attributes:

- xs:string
- xs:boolean
- xs:decimal

- xs:dateTime
- xs:time
- xs:date
- xs:anyURI
- xs:Name
- xs:int
- osd:html
- osd:email
- osd:password
- osd:locale
- osd:text

Restrictions on User Defined Attributes and Catalogs

The following features are unsupported on UDA elements:

- Facets
- Functions using the `osd:function` property
- UI bean editors using the `osd:uiBean` property
- The `osd:checkNullInput` property
- History features
- Replication
- Inheritance features, using the `osd:inheritance` property

As UDA catalogs are internally considered to be tables, the restrictions that apply to tables also exist for `UDACatalog` elements.

153.6 **Aggregated lists**

In XML Schema, the maximum number of times an element can occur is determined by the value of the `maxOccurs` attribute in its declaration. If this value is strictly greater than 1 or is `unbounded`, the data can have multiple occurrences. If no `osd:table` declaration is included, this element is called an *aggregated list*.

In Java, it is then represented as an instance of the class `java.util.List`.

The following is an example of an aggregated list that defines the pricing of a loan product, depending on the amount borrowed.

```
<xs:element name="pricing" minOccurs="0" maxOccurs="unbounded"
 osd:access="RW">
 <xs:annotation>
  <xs:documentation>
   <osd:label>Pricing</osd:label>
   <osd:description>Pricing grid </osd:description>
  </xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element name="amount" type="xs:int">
    <xs:annotation>
     <xs:documentation>
```

```
      <osd:label>Amount borrowed</osd:label>
     </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="monthly" type="xs:int">
    <xs:annotation>
     <xs:documentation>
      <osd:label>Monthly payment </osd:label>
     </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="cost" type="xs:int">
    <xs:annotation>
     <xs:documentation>
      <osd:label>Cost</osd:label>
     </xs:documentation>
    </xs:annotation>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Aggregated lists have a dedicated editor in EBX®. This editor allows you to add or to delete occurrences.

Attention

The addition of an `osd:table` declaration to an element with `maxOccurs > 1` is a very important consideration that must be taken into account during the design process. An aggregated list is severely limited with respect to the many features that are supported by tables. Some features unsupported on aggregated lists that are supported on tables are:

- Performance and memory optimization;
- Lookups, filters and searches;
- Sorting, view and display in hierarchies;
- Identity constraints (primary keys and uniqueness constraints);
- Detailed permissions for creation, modification, deletion and particular permissions at the record level;
- Detailed comparison and merge.

Thus, *aggregated lists should be used only for small volumes of simple data (one or two dozen occurrences), with no advanced requirements*. For larger volumes of data or more advanced functionalities, it is strongly advised to use an `osd:table` declaration.

For more information on table declarations, see Tables and relationships [p 913].

153.7 Including external data models

Including another data model in your current model allows you to use the reusable types that are defined in that data model. You can thus use the inclusion of external data models to share data types between multiple XML Schema Documents.

To include another XML Schema Document in your model, thereby including the data types that it defines, specify the `xs:include` element as follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
   <xs:include  schemaLocation="./schemaToInclude.xsd"/>
 ...
</xs:schema>
```

The attribute `schemaLocation` is mandatory and must specify either an absolute or a relative path to the XML Schema Document to include.

The inclusion of XML Schema Documents is not namespace aware, thus all included data types must belong to the same namespace. As a consequence, including XML Schema Documents that define data types of the same name is not supported.

EBX® includes extensions with specific URNs for including embedded data models and data models packaged in modules.

To include an embedded data model in a model, specify the URN defined by EBX®. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
 <xs:include schemaLocation="urn:ebx:publication:myPublication"/>
 ...
</xs:schema>
```

To include a data model packaged in a module, specify the specific URN defined by EBX®. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
 <xs:include schemaLocation="urn:ebx:module:aModuleName:/WEB-INF/ebx/schema/myDataModel.xsd"/>
 ...
</xs:schema>
```

See `SchemaLocation`[API] for more information about specific URNs supported by EBX®.

> **Note**
>
> If the packaged data model uses Java resources, the class loader of the module containing the data model will be used at runtime for resolving these resources.

CHAPTER 154

Tables and relationships

This chapter contains the following topics:

1. Tables
2. Foreign keys
3. Associations
4. Linked fields

154.1 Tables

Overview

TIBCO EBX® supports the features of relational database tables, including the handling of large volumes of records, and identification by primary key.

Tables provide many benefits that are not offered by aggregated lists [p 910]. Beyond relational capabilities, some features that tables provide are:

- filters and searches;
- sorting, views and hierarchies;
- identity constraints: primary keys, foreign keys [p 919] and uniqueness constraints [p 937];
- specific permissions for creation, modification, and deletion;
- dynamic and contextual permissions at the individual record level;
- detailed comparison and merge;
- ability to have inheritance at the record level (see dataset inheritance [p 482]);
- performance and memory optimization.

See also

Entity mappings [p 92]

Foreign keys [p 919]

Associations [p 923]

Linked fields [p 933]

Working with existing datasets [p 165]

Simple tabular views *[p 135]*

Hierarchical views *[p 135]*

History *[p 461]*

Declaration

A table element, which is an element with *maxOccurs > 1*, is declared by adding the following annotation:

```
<xs:annotation>
 <xs:appinfo>
  <osd:table>
   <primaryKeys>/pathToField1 /pathToField...n</primaryKeys>
  </osd:table>
 </xs:appinfo>
</xs:annotation>
```

> **Note**
>
> A default entity name is associated to the table if it does not have one explicitly set in the data model. The table's name is used as its default entity name. Entity names allow to refer to tables in SQL requests and Data Service operations using unique names instead of their paths by defining suffixes for WSDL operations and aliases for SQL requests. See Entity mappings [p 92] for more informations.

Common properties

Element	Description	Required
`primaryKeys`	Specifies the primary key fields of the table. Each field of the primary key must be denoted by its absolute XPath notation that starts just under the root element of the table. If there are multiple fields in the primary key, the list is delimited by whitespace. **Note:** Whitespaces in primary keys of type `xs:string` are handled differently. See Whitespace handling for primary keys of type string [p 949].	Yes.

`defaultLabel`	Defines the end-user display of records. Multiple variants can be specified:
	• A static non-localized expression is defined using the `defaultLabel` element, for example:
	`<defaultLabel>Product: ${./productCode}</defaultLabel>`
	• Static localized expressions are specified using the `defaultLabel` element with the attribute `xml:lang`, for example:
	`<defaultLabel xml:lang="fr-FR">Produit : ${./productCode}</defaultLabel>`
	`<defaultLabel xml:lang="en-US">Product: ${./productCode}</defaultLabel>`
	• A JavaBean that implements the interface `UILabelRenderer`[API] and/or the interface `UILabelRendererForHierarchy`[API]. The JavaBean is specified by means of the attribute `osd:class`, for example:
	`<defaultLabel osd:class="com.wombat.MyLabel"></defaultLabel>`
	Attention
	Since the end-user display should be sortable, only fields that use sortable search strategies are allowed in static expressions.
	Note: The priority of the tags when displaying the user interface is the following:
	1. `defaultLabel` tags with a JavaBean (but it is not allowed to define several renderers of the same type);
	2. `defaultLabel` tags with a static localized expression using the `xml:lang` attribute;
	3. `defaultLabel` tags with a static non-localized expression.
	Attention: Access rights defined on associated datasets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
		No.

`recordForm`	Defines a specific component for customizing the record form in a dataset. This component is defined using a JavaBean that extends `UIForm`[API] or implements `UserServiceRecordFormFactory`[API]. The JavaBean is specified by means of the attribute `osd:class`, for example: `<recordForm osd:class="com.wombat.MyRecordForm"/>`	No.

Example

Below is an example of a product catalog:

```
<xs:element name="Products" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
   <xs:documentation>
    <osd:label>Product Table </osd:label>
    <osd:description>List of products in Catalog </osd:description>
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
  <xs:annotation>
   <xs:appinfo>
    <osd:table>
     <primaryKeys>./productRange /productCode</primaryKeys>
     <index name="indexProductCode">/productCode</index>
    </osd:table>
   </xs:appinfo>
  </xs:annotation>
   <xs:sequence>
    <xs:element name="productRange" type="xs:string"/><!-- key -->
    <xs:element name="productCode" type="xs:string"/><!-- key -->
    <xs:element name="productLabel" type="xs:string"/>
    <xs:element name="productDescription" type="xs:string"/>
    <xs:element name="productWeight" type="xs:int"/>
    <xs:element name="productType" type="xs:string"/>
    <xs:element name="productCreationDate" type="xs:date"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Catalogs" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
   <xs:documentation>
    <osd:label>Catalog Table</osd:label>
    <osd:description>List of catalogs</osd:description>
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:annotation>
    <xs:appinfo>
     <osd:table>
      <primaryKeys>/catalogId</primaryKeys>
     </osd:table>
    </xs:appinfo>
   </xs:annotation>
   <xs:sequence>
    <xs:element name="catalogId" type="xs:string"/><!-- key -->
    <xs:element name="catalogLabel" type="xs:string"/>
    <xs:element name="catalogDescription" type="xs:string"/>
    <xs:element name="catalogType" type="xs:string"/>
    <xs:element name="catalogPublicationDate" type="xs:date"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

Properties related to dataset inheritance

The following properties are only valid in the context of dataset inheritance:

Element	Description	Required
`onDelete-deleteOccultingChildren`	Specifies whether, upon record deletion, child records in occulting mode are also to be deleted. Valid values are: `never` or `always`.	No, default is `never`.
`mayCreateRoot`	Specifies whether root record creation is allowed. The expression must follow the syntax below. See definition modes [p 482].	No, default is `always`.
`mayCreateOverwriting`	Specifies whether records are allowed to be overwritten in child datasets. The expression must follow the syntax below. See definition modes [p 482].	No, default is `always`.
`mayCreateOcculting`	Specifies whether records are allowed to be occulted in child datasets. The expression must follow the syntax below. See definition modes [p 482].	No, default is `always`.
`mayDuplicate`	Specifies whether record duplication is allowed. The expression must follow the syntax below.	No, default is `always`.
`mayDelete`	Specifies whether record deletion is allowed. The expression must follow the syntax below.	No, default is `always`.

The `may...` expressions specify when the action is possible, though the ultimate availability of the action also depends on the user access rights. The expressions have the following syntax:

```
expression ::= always | never | <condition>*
condition ::= [root:yes | root:no]
"always": the operation is "always" possible (but user rights may restrict this).
"never": the operation is never possible.
"root:yes": the operation is possible if the record is in a root instance.
"root:no": the operation is not possible if the record is in a root instance.
```

If the record does not define any specific conditions, the default is used.

See also *Dataset inheritance [p 481]*

Using toolbars

It is possible to define the toolbars to display in the user interface using the element `defaultView/toolbars` under `xs:annotation/appinfo/osd:table`. A toolbar allows to customize the buttons and menus to display when displaying a table view, a hierarchical view, or a record form.

The table below presents the elements that can be defined under `defaultView/toolbars`.

Element	Description	Required
`tabularViewTop`	Defines the toolbar to use on top of the default table view.	No.
`tabularViewRow`	Defines the toolbar to use on each row of the default table view.	No.
`recordTop`	Defines the toolbar to use in the record form.	No.
`hierarchyViewTop`	Defines the toolbar to use in the default hierarchy view of the table.	No.

See also *Toolbars [p 971]*

Example

Below is an example of custom toolbars used by a product catalog:

```
<xs:element name="Products" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
   <xs:documentation>
    <osd:label>Product Table </osd:label>
    <osd:description>List of products in Catalog </osd:description>
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
  <xs:annotation>
   <xs:appinfo>
    <osd:table>
     <primaryKeys>./productRange /productCode</primaryKeys>
    <defaultView>
     <toolbars>
      <tabularViewTop>toolbar_name_for_tabularViewTop</tabularViewTop>
      <tabularViewRow>toolbar_name_for_tabularViewRow</tabularViewRow>
      <recordTop>toolbar_name_for_recordTop</recordTop>
      <hierarchyViewTop>toolbar_name_for_hierarchyViewTop</hierarchyViewTop>
     </toolbars>
    </defaultView>
   </osd:table>
   </xs:appinfo>
  </xs:annotation>
 ...
  </xs:complexType>
</xs:element>
```

> **Note**
>
> If a toolbar does not exist or is not available for a specific location then no toolbar will be displayed in the user interface in the corresponding location.

Record metadata

EBX® automatically adds record metadata to a table's structure. The metadata nodes enable retrieval of the following information about a record:

- Creator.
- Creation time.
- Last modifier.
- Time of last modification.

- Unique identifier of a record (UUID). This metadata being technical it cannot be used in a custom table view, in the structured search tools and quick search.
- The definition mode of a record. This metadata being technical it cannot be used in a custom table view, in the structured search tools and quick search.

An example use for record metadata might be to display or fetch all the records created by a specific user since a specific date. This is possible:

- For business users in the user interface by:
  - Using the quick search feature to locate records by using the metadata fields as part of the search criteria.
  - Defining table views that include the desired metadata fields.
- Via the API by querying with XPath or SQL.

Attention

Please take note of the following points regarding metadata:

- Metadata are defined under a group with the reserved name *ebx-metadata*. As a consequence, the metadata won't be available in a table if it contains a group, directly under its root, with the same name. A warning displays when compiling the data model if a table defines a group with this reserved name.
- Metadata are read-only.
- Metadata are not included when exporting the content of a table to an XML or CSV file.
- By default, metadata fields are not including when instrospecting a data model using the API. You can explicitly included them using the dedicated **API** `SchemaNode.getNodeChildren`[API].

See also

Metadata fields in SQL *[p 1259]*

Technical data *[p 1240]*

`SchemaMetadataConstants`[API]

`Constants.Data`[API]

154.2 Foreign keys

Declaration

A reference to a table [p 913] is defined using the extended facet `osd:tableRef`.

The node holding the `osd:tableRef` declaration must be of type `xs:string`.

At the instantiation, any value of the node identifies a record in the target table using its **primary key syntax** `PrimaryKey.syntax`[API].

This extended facet is also interpreted as an enumeration whose values refer to the records in the target table.

Element	Description	Required
`tablePath`	XPath expression that specifies the target table.	Yes.
`container`	Reference of the dataset that contains the target table.	Only if the dataspace element is defined. Otherwise, default is the current dataset.
`branch`	Reference of the dataspace that contains the container dataset.	No, default is the current dataspace or snapshot.

`display`	Custom display for presenting the selected foreign key in the current record and the sorted list of possible keys. Two variants can be specified, either pattern-based expressions, or a JavaBean if the needs are very specific:
	• Static expressions are specified using the `display` and `pattern` elements. These static expressions can be localized using the additional attribute `xml:lang` on the `pattern` element, for example:
	`<display>`
	`<pattern>Product : ${./productCode}</pattern>`
	`<pattern xml:lang="fr-FR">Produit : ${./productCode}</pattern>`
	`<pattern xml:lang="en-US">Product: ${./productCode}</pattern>`
	`</display>`
	Attention
	Since the display pattern should be sortable, only fields that use sortable search strategies are allowed.
	• A JavaBean that implements the interface `TableRefDisplay`[API]. It is specified using the attribute `osd:class`. For example:
	`<display osd:class="com.wombat.MyLabel"></display>`
	Attention
	Quick search and sort operations in the user interface will use the raw value instead of the label of the records if a JavaBean is defined or if the target table defines a **programmatic label** `UILabelRenderer`[API] for its records. A static expression must be defined to use in these operations the label of the records.
	It is not possible to define both variants on the same foreign key element.
	Attention: Access rights defined on associated datasets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
		No, if the `display` property is not specified, the table's record rendering [p 915] is used.

Element	Description	Required

`filter`	Specifies an additional constraint that filters the records of the target table. Two types of filters are available:
	• An XPath filter is an XPath predicate in the target table context. It is specified using the `predicate` element. For example:
	`<filter><predicate>type = ${../refType}</predicate></filter>`
	A localized validation message can be specified using the element `validationMessage`, which will be displayed to the end-user at the validation time if a record is not accepted by the filter.
	A specific severity level can be defined in a nested `severity` element. The default severity is 'error'.
	Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a `message` element with no `xml:lang` attribute.
	In the user interface, the XPath filter is applied to filter a table according to the value of a foreign key field. That is, if a foreign key field specifies an XPath filter in a data model, then it will be reused in the filter pane to restrict the set of values in the associated combo-box displayed in the filter pane. However, the predicate used by the filter pane will only take into account the non-contextual parts of the predicate.
	• A programmatic filter is a JavaBean that implements the interface `TableRefFilter`[API]. It is specified using the attribute `osd:class`. For example:
	`<filter osd:class="com.wombat.MyFilter"></filter>`
	Additional validation messages can be specified during the setup of the programmatic filter.
	In the user interface, programmatic filters are not applied to filter the set of values in the associated combo-box displayed in the filter pane. However, it is possible to specify an additional XPath predicate that will be used in the filter pane of the user interface. This XPath predicate is specified during the setup of the programmatic filter using the method `TableRefFilterContext.setFilterForSearch`[API].
	Note
	The attributes `osd:class` and the property `predicate` cannot be set simultaneously. The validation search XPath functions are forbidden on a `tableRef` filter.
	See also
	JavaBean specifications `Package com.orchestranetworks.schema.JavaBeans`[API]
	`JavaBeanVersion`[API]
		No.

`validation`	Specifies localized validation messages for the `osd:tableRef` and error management policy. A specific severity level can be defined in a nested `severity` element. The default severity is 'error'. An error management policy can be defined in a nested `blocksCommit` element. The error management policy that	No.

Element	Description	Required
	blocks all operations does not apply to filters. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected, and a validation error will be reported. Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a `message` element with no `xml:lang` attribute.	

Attention

You can create a dataset which has a foreign key to a container that does not exist in the repository. However, the content of this dataset will not be available until the container is created. After the creation of the container, a data model refresh is required to make the dataset available. When creating a dataset that refers to a container that does not yet exist, the following limitations apply:

- Triggers defined at the dataset level are not executed.
- Default values for fields that are not contained in tables are not initialized.
- During an archive import, it is not possible to create a dataset that refers to a container that does not exist.

Example

The example below specifies a foreign key in the 'Products' table to a record of the 'Catalogs' table.

```
<xs:element name="catalog_ref" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:tableRef>
     <tablePath>/root/Catalogs</tablePath>
     <display>
       <pattern xml:lang="en-US">Catalog: ${./catalogId}</pattern>
       <pattern xml:lang="fr-FR">Catalogue : ${./catalogId}</pattern>
     </display>
      <validation>
       <severity>error</severity>
       <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
       <message>A default error message</message>
       <message xml:lang="en-US">A localized error message</message>
       <message xml:lang="fr-FR">Un message d'erreur localisé</message>
     </validation>
    </osd:tableRef>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

See also

Table definition *[p 913]*

Primary key syntax `PrimaryKey.syntax`[API]

Extraction of foreign keys (XPath predicate syntax) *[p 447]*

Associations *[p 923]*

View for advanced selection *[p 965]*

`SchemaNode.getFacetOnTableReference`[API]

`SchemaFacetTableRef`[API]

154.3 Associations

Overview

An association provides an abstraction over an existing relationship in the data model, and allows an easy model-driven integration of *associated objects* in the user interface and in data services.

Several types of associations are supported:

- *'By foreign key'* specifies the inverse relationship of an existing foreign key field [p 919].
- *'Over a link table'* specifies a relationship based on an intermediate link table (such tables are often called "join tables"). This link table has to define two foreign keys, one referring to the 'source' table (the table holding the association element) and another one referring to the 'target' table.
- *'By an XPath predicate'* specifies a relationship based on an XPath predicate.

For an association, it is also possible to:

- Filter associated objects by specifying an additional XPath filter.
- Configure a tabular view to define the fields that must be displayed in the associated table.
- Define how associated objects are to be rendered in forms.
- Hide/show associated objects in the data service 'select' operation. See Hiding a field in Data Services [p 964].
- Specify the minimum and maximum number of associated objects that are required.
- Add validation constraints using XPath predicates for restricting associated objects.

See also

`SchemaNode.getAssociationLink`[API]

`SchemaNode.isAssociationNode`[API]

`AssociationLink`[API]

Declaration

Associations are defined in the data model using the XML Schema element `osd:association` under `xs:annotation/appInfo`.

Restrictions:

- An association must be a simple element of type *xs:string*.
- An association can only be defined inside a table.

> **Note**
>
> The "official" cardinality constraints (`minOccurs="0"  maxOccurs="0"`) are required because, from an instance of XML Schema, the corresponding node is absent. In other words, an association has no value and is considered as a "virtual" element as far as XML and XML Schema is concerned.

The table below presents the elements that can be defined under `xs:annotation/appInfo/osd:association`.

Element	Description	Required
`tableRefInverse`	Defines the properties of an association that is the inverse relationship of a *foreign key*. The element `fieldToSource` defines the foreign key that refers to the source table of the association. The element `fieldToSource` is mandatory and must specify a foreign key field that refers to the table containing the association. The element `fieldToSource` can be defined on a foreign key list (`maxOccurs` > 1), in that case, the list should be declared as unique with a blocking uniqueness constraint (`onInsertUpdateOrDelete`). **See also** *Blocking and non-blocking constraints* [p 945] *Uniqueness constraints* [p 937]	Yes if the association is the inverse relationship of a *foreign key*, otherwise no.
`linkTable`	Defines the properties of an association over a link table. The element `table` specifies the link table used by the association. The element `table` is mandatory and must refer to an existing table. **Important:** In order to be used by an association, a link table must define a primary key that is composed of auto-incremented fields and/or the foreign key to the source or target table of the association. The element `fieldToSource` defines the foreign key that refers to the source table of the association. The element `fieldToSource` is mandatory and must specify a foreign key field that refers to the table containing the association. The element `fieldToTarget` defines the foreign key that refers to the target table of the association. The element `fieldToTarget` is mandatory and must specify a foreign key field.	Yes if the association is over a link table, otherwise no.
`xpathLink`	Defines the properties of an association that is based on an *XPath predicate*. The `predicate` element specifies the criteria of the association, relative to the current node. Examples: `/root/Products[catalog_ref =${../catalogId}]` or `//Products[catalog_ref = ${../catalogId}]` or `../Products[catalog_ref =${../catalogId}]`. The path to the predicate, for example `../Products`, specifies the target table of the association. This part of the path is resolved with respect to the current table. It is not possible to refer to a table using a relative path if the association targets a table in another dataset. If the association depends on fields of the source table, the XPath expression predicate must include references to the elements on which it depends	Yes if the association is based on an *XPath predicate*, otherwise no.

Element	Description	Required
	using the notation `${<relative-path>}` where `relative-path` is a path that identifies the element relative to the association node. See EBX® XPath supported syntax [p 441]. **Note** The validation search XPath functions are forbidden on an XPath link.	
`filter`	Defines an XPath predicate to filter associated objects using the `predicate` element. For example: `<filter><predicate>type = ${../refType}</predicate></filter>` It is only possible to use fields from the source and the target tables when defining an XPath filter. That is, if it is an association over a link table, it is not possible to use fields of the link table in the XPath filter. **Error message on creation**: in the user interface, the record creation is blocked when a user submits a new associated record that does not comply with the filter. The error message can be customized using the element `checkOnAssociatedRecordCreation/message`. Each localized message variant is defined in a nested message element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a message element with no `xml:lang` attribute. See Examples [p 931] for more information on this property. **Note** The validation search XPath functions are forbidden for association filter.	No.
`xpathFilter`	**Note:** Deprecated. This property has been replaced by the property `filter`. Defines an XPath predicate to filter associated objects.	No.
`recordForm`	Defines a specific component for customizing the form of an associated record. This component is defined using a JavaBean that implements `UserServiceAssociationRecordFormFactory`[API]. The JavaBean is specified by means of the attribute `osd:class`, for example: `<recordForm osd:class="com.wombat.MyRecordFormFactory"/>`	No.

It is possible to refer to another dataset. For that, the following properties must be defined either under the element `tableRefInverse`, `linkTable` or `xpathLink` depending on the type of the association:

Element	Description	Required
`schemaLocation`	Defines the data model containing the fields used by the association. The data model is defined using a specific URN that allows referring to embedded data models and data models packaged in modules. See `SchemaLocation`[API] for more information about specific URNs supported by EBX®.	Yes.
`dataSet`	Defines the dataset used by the association. This dataset must use the data model specified by the element `schemaLocation`.	Yes.
`dataSpace`	Defines the dataspace containing the dataset used by the association.	No.

Important:

- When creating a dataset, you can create a dataset that defines an association to a container that does not yet exist in the repository. However, the content of this dataset will not be available immediately upon creation. After the absent container is created, a data model refresh is required in order to make the dataset available. When creating a dataset that refers to a container that does not yet exist, the following limitations apply:
  - Triggers defined at the dataset level are not executed.
  - Default values on fields outside tables are not initialized.
  - During an archive import, it is not possible to create a dataset that refers to a container that does not exist.
- The consistency of an association to another dataset is checked in the operational phase. If an association is not consistent in the context of a specific dataset then:
  - It is considered as read-only from a permission perspective. That is, the association is read-only in the user interface and the data services are hidden. Also, in the user interface it is not possible to create, delete, attach, detach, or move an associated record. However, it is possible to modify an existing associated record.
  - It is excluded from the validation of the container dataset, table or record.

User interface integration

It is possible to define how associated objects are to be rendered in forms, using the element `osd:defaultView/displayMode` under `xs:annotation/appinfo`.

Possible values are:

- `inline`, specifies that associated records are to be rendered in the form at the same position of the association in the data model.
- `tab`, specifies that associated records are to be rendered in a specific tab.
- `link`, specifies that associated records are to be rendered in a modal window.

By default, associated records are rendered `inline` if this property is not defined.

The following example specifies that associated objects are to be rendered `inline` in the form:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
  <osd:association>
   <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
  </osd:association>
  <osd:defaultView>
   <displayMode>inline</displayMode>
  </osd:defaultView>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

The following example specifies that associated objects are to be rendered in a specific tab:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
  <osd:association>
   <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
  </osd:association>
  <osd:defaultView>
   <displayMode>tab</displayMode>
  </osd:defaultView>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

Using toolbars

It is possible to define the toolbars to display in the user interface using the element `osd:defaultView/toolbars` under `xs:annotation/appinfo`. A toolbar allows to customize the buttons and menus to display when displaying the tabular view of an association.

The table below presents the elements that can be defined under `osd:defaultView/toolbars`.

Element	Description	Required
`tabularViewTop`	Defines the toolbar to use in the default table view of this association.	No.
`tabularViewRow`	Defines the toolbar to use for each row of the default view of this association.	No.

The following example shows how to use toolbars from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
   </osd:association>
   <osd:defaultView>
    <toolbars>
     <tabularViewTop>toolbar_name_for_tabularViewTop</tabularViewTop>
     <tabularViewRow>toolbar_name_for_tabularViewRow</tabularViewRow>
    </toolbars>
   </osd:defaultView>
  </xs:appinfo>
 </xs:annotation>
```

```
</xs:element>
```

> **Note**
>
> It is only possible to use the toolbars defined in the data model containing the target table of the association. That is, if the target table of the association is defined in another data model, then it is only possible to reference a toolbar defined in this data model and not in the one holding the association.

See also *Toolbars [p 971]*

Customized view of associated objects

A specific tabular view can be specified to define the fields that must be displayed in the target table. If a tabular view is not defined, all columns that a user is allowed to view, according to the granted access rights, are displayed. A tabular view is defined using the element `osd:defaultView/tabularView` under `xs:annotation/appinfo`.

The table below shows the elements that can be defined under `osd:defaultView/tabularView`.

Element	Description	Required
`column`	Define a field of the target table to display. The specified path must be absolute from the target table and must refer to an existing field. Several `column` elements can be defined to specify the fields that are to be displayed.	No.
`sort`	Define a field that can be used to sort associated objects. Several`sort` elements can be defined to specify the fields that can be used to sort associated objects. The element `nodePath` defines the path of the field that can be used to sort associated objects. The element `isAscending` specifies whether the sort order is ascending (true) or descending (false).	No.

The following example shows how to define a tabular view from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
   </osd:association>
   <osd:defaultView>
    <tabularView>
     <column>/productRange</column>
     <column>/productCode</column>
     <column>/productLabel</column>
     <column>/productDescription</column>
     <sort>
        <nodePath>/productLabel</nodePath>
        <isAscending>true</isAscending>
     </sort>
    </tabularView>
   </osd:defaultView>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Actions in the user interface

In the user interface, it is possible to perform the following actions:

- **Create**: it allows directly creating an object in the target table of the association. When a new object is created, it is automatically associated with the current record.
- **Duplicate**: allows to duplicate an object in the target table of the association. When a new object is created, it is automatically associated with the current record.
- **Associate**: associates an existing object with the current record. In the case of an association over a link table, a record in the link table is automatically created to materialize the link between the current record and the existing object.
- **Move**: associates the selected objects to a different record than the current one. In the case of an association over a link table, the previous link record is automatically deleted and a new record in the link table is automatically created to materialize the link between the selected objects and their new parent record.
- **Delete**: deletes selected associated objects in the target table of the association.
- **Detach**: breaks the semantic link between the current record and the selected associated objects. In the case of an association over a link table, the records in the link table are automatically deleted, to break the links between the current record and associated objects.

> **Note**
>
> The actions *associate* and *detach* are not available when the association is defined using an **XPath predicate** (element *xpathLink*).

Customized view for actions

A published view, tabular or hierarchical, can be specified to define how objects should be displayed when performing an action through the user interface. A published view is defined using the element `osd:defaultView/associationViews` under `xs:annotation/appinfo`.

The table below shows the elements that can be defined under `osd:defaultView/associationViews`.

Element	Description	Required
`viewForAssociateAction`	Define a published view to be used when displaying the objects in the target table to be associated with the current record. The specified view must be published and created upon the target table of the association.	No.
`viewForMoveAction`	Define a published view to be used when moving an associated object to another record of the current table. The specified view must be published and created upon the current table.	No.

The following example shows how to define views from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
```

```
   </osd:association>
   <osd:defaultView>
    <associationViews>
     <viewForAssociateAction>view_name_for_catalogs</viewForAssociateAction>
     <viewForMoveAction>view_name_for_products</viewForMoveAction>
    </associationViews>
   </osd:defaultView>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Validation

Some controls can be defined on associations, in order to restrict associated objects. These controls are defined under the element `osd:association`.

The table below presents the controls that can be defined under xs:annotation/appInfo/osd:association.

Element	Description	Required
`minOccurs`	Specifies the minimum number of associated objects that are required for this association. This minimum number is defined using the element `value` and must be a positive integer.	No, by default the minimum is not restricted.
`maxOccurs`	Specifies the maximum number of associated objects that are allowed for this association. This maximum number is defined by the element `value` and must be either a positive integer or the raw string `unbounded` which indicates that this maximum is not restricted. The maximum number of associated objects must be greater than the minimum number of associated objects.	No, by default the maximum is not restricted.
`constraint`	Defines an XPath predicate for restricting associated records. It is only possible to use fields from the source and the target table when defining an XPath predicate. That is, if it is an association over a link table, it is not possible to use fields of the link table in the XPath predicate. In associated datasets, a validation message of the specified severity is added and displayed to the end-user at the validation time when an associated record does not comply with the specified constraint.	No.
`validation`	A validation message can be defined under the elements `minOccurs`, `maxOccurs` and `constraint`, using the element `validation`. The severity of the validation message is specified using the element `severity`. Possible severities are: `error`, `warning` and `info`. If the severity is not specified then, by default, the severity `error` is used. A localized validation message can be specified using the element `message`, which will be displayed to the end-user at the validation time if an association does not comply with this constraint. Each localized message variant is defined in a nested message element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a message element with no `xml:lang` attribute.	No.

Data services integration

It is possible to define whether associated objects must be hidden in the Data service `select` operation. For this, the property `osd:defaultView/hiddenInDataServices` under `xs:annotation/xs:appinfo` can be set on the association. Setting the property to 'true' will hide associated objects in the Data service `select` operation. If this property is not defined then, by default, associated objects will be shown in the Data service `select` operation.

See also

Hiding a field in Data Services [p 964]

Association field [p 1080]

Examples

For example, the product catalog data model defined previously [p 916] specifies that a product belongs to a catalog (explicitly defined by a foreign key in the 'Products' table). The reverse relationship (that a catalog has certain products) is not easily represented in XML Schema, unless the 'Catalogs' table includes the following association that is the inverse of a foreign key:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
   <osd:association>
   <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
   </osd:association>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

For an association over a link table, we can consider the previous example and bring some updates. For instance, the foreign key in the 'Products' table is deleted and the relation between a product and a catalog is redefined by a link table (named 'Catalogs_Products') that has a primary key composed of two foreign keys: one that refers to the 'Products' table (named 'productRef') and another to the 'Catalogs' table (named 'catalogRef'). The following example shows how to define an association over a link table from this new relationship:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
  <osd:association>
   <linkTable>
     <table>/root/Catalogs_Products</table>
     <fieldToSource>./catalogRef</fieldToSource>
     <fieldToTarget>./productRef</fieldToTarget>
   </linkTable>
  </osd:association>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

The following example shows an association that refers to a foreign key in another dataset. In this example, the 'Products' and 'Catalogs' tables are not in the same dataset:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
   <osd:association>
   <tableRefInverse>
    <schemaLocation>urn:ebx:module:aModuleName:/WEB-INF/ebx/schema/products.xsd</schemaLocation>
    <dataSet>Products</dataSet>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
   </osd:association>
```

```
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

The following example defines an XPath filter to associate only products of the 'Technology' type:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
   <osd:association>
   <tableRefInverse>
    <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
   <filter>
    <predicate>./productType = 'Technology'</predicate>
    <checkOnAssociatedRecordCreation>
     <message>A default message</message>
                   <message xml:lang="en-US">A localized message</message>
                   <message xml:lang="fr-FR">Un message localisé</message>
    </checkOnAssociatedRecordCreation>
   </filter>
   </osd:association>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

The following example specifies the minimum number of products that are required for a catalog:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
      <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
     <minOccurs>
     <value>1</value>
     <validation>
      <severity>warning</severity>
      <message xml:lang="en-US">One product should at least be associated to this catalog.</message>
      <message xml:lang="fr-FR">Un produit doit au moins être associé à ce catalogue.</message>
     </validation>
    </minOccurs>
   </osd:association>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

The following example specifies that a catalog must contain at most ten products:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
    <maxOccurs>
     <value>10</value>
     <validation>
      <severity>warning</severity>
      <message xml:lang="en-US">Too much products for this catalog.</message>
      <message xml:lang="fr-FR">Ce catalogue a trop de produits.</message>
     </validation>
    </maxOccurs>
   </osd:association>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

154.4 Linked fields

Overview

Linked fields provide the ability to simulate a multi-table view by aggregating the fields of a main table with some fields of another table over an existing relationship. It allows an easy model-driven integration of some fields from records that are referred by other ones using relationships.

See also *SchemaLinkedField*[API]

Following relationships can be traversed:

- Foreign key constraints [p 919].
- Associations 'By foreign key' [p 923].

Important: these relationships must be "single valued relationships". That is, a record of the table that defines the relationship must refer to only one record in the target table of the relationship.

In details:

- A foreign key is considered as a single valued relationship if the foreign key field defines a *maxOccurs* equal to 1.
- An association by foreign key is considered as a single valued relationship if the referred foreign key field is the unique primary key field of the target table or if an uniqueness constraint is defined on this foreign key field.

> **Note**
>
> A linked field inherits some properties from its target field (foreign key constraint, enumeration, value labelling, search strategies, search mode, formatting).

Declaration

A linked field, which is an element with *minOccurs = 0* and *maxOccurs = 0*, is declared by adding the following annotation:

```
<xs:element name="catalog_ref_label" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:function linkedField="../catalog_ref/catalogLabel"/>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Attribute *linkedField* defined in element *osd:function* defines a path composed of steps which refer to a single-valued relationship in the container table and of steps to a field in the target table of the relationship.

Important:

- If the path is absolute (starts with a "/") then it will be resolved from the container table node.
- If the path is relative then it will be resolved from the current field.

In the context of the product catalog data model defined previously [p 916], the field *catalog_ref_label* is added in the *Products* table. The first step *catalog_ref* refers to the foreign key that indicates that a product belongs to a catalog. The last step *catalogLabel* refers to the label of the referred catalog. This field belongs to the *Catalog* table that is the target table of the foreign key constraint.

Restrictions:

- Currently, linked fields only allow to target one level of single valued relationships. As a consequence:
  - The single-valued relationship cannot be defined by another linked field.
  - A linked field cannot target another linked field.
- A linked field must define properties *minOccurs* and *maxOccurs*. equal to 0.
- A linked field must define a data type that is compatible with the one of the target field.
- A linked field cannot be a part of the primary key of the container table.
- A linked field must target a terminal field.
- A linked field cannot target an aggregated list or a field under an aggregated list.
- A linked field cannot refer to a single-valued relationship that is also a linked field.
- Constraints should not be defined on a linked field. If constraints are declared on a linked field, a warning message will be thrown and the declared constraints will be ignored.
- For a linked field, the property `osd:disableValidation` is always considered as `true` and cannot be overwritten.

CHAPTER 155

Constraints

Facets allow you to define data constraints in your data models. TIBCO EBX® supports XML Schema constraining facets and provides extended and programmatic facets for advanced data controls.

This chapter contains the following topics:

1. XML Schema supported facets
2. Extended facets
3. Programmatic facets
4. Control policy

155.1 XML Schema supported facets

The tables below show the facets that are supported by different data types.

Key:

- **X** - Supported
- **1** - The `whiteSpace` facet can be defined, but is not interpreted by EBX®
- **2** - In XML Schema, boundary facets are not allowed on the type `string`. Nevertheless, EBX® allows such facets as extensions.
- **3** - The `osd:resource` type only supports the facet `FacetOResource`, which is required. See Extended Facets [p 939].
- **4** - `osd:dataspaceKey`, `osd:datasetName` and `osd:color` types do not support facets.

Only Programmatic constraints [p 943] are supported on these types.

	length	minLength	max Length	pattern	enumeration	white Space
xs:string	X	X	X	X	X	1
xs:boolean				X		1
xs:decimal				X	X	1
xs:dateTime				X	X	1
xs:time				X	X	1
xs:date				X	X	1
xs:anyURI	X	X	X	X	X	1
xs:Name	X	X	X	X	X	1
xs:integer				X	X	1
osd:resource [p 902][3]						1
osd:dataspaceKey [p 904][4]						1
osd:datasetName [p 906][4]						1
osd:color [p 904][4]						1

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
xs:string			2	2	2	2
xs:boolean						
xs:decimal	X	X	X	X	X	X
xs:dateTime			X	X	X	X
xs:time			X	X	X	X
xs:date			X	X	X	X

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
xs:anyURI						
xs:Name			**2**	**2**	**2**	**2**
xs:integer	**X**	**X**	**X**	**X**	**X**	**X**
osd:resource [p 902][3]						
osd:dataspaceKey [p 904][4]						
osd:datasetName [p 906][4]						
osd:color [p 904][4]						

Example:

```
<xs:element name="loanRate">
 <xs:simpleType>
  <xs:restriction base="xs:decimal">
   <xs:minInclusive value="4.5" />
   <xs:maxExclusive value="17.5" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Uniqueness constraint

It is possible to define a uniqueness constraint, using the standard XML Schema element xs:unique. This constraint indicates that a value or a set of values has to be unique inside a table.

Case-sensitivity to be used to check string fields defined in the uniqueness constraint can be specified. Possible value are for case sensitivity are: "sensitive" or "insensitive". By default, uniqueness constraints are case-sensitive.

> **Note**
>
> Applying the case-insensitive option can lead to performance problems if it is used to check tables with large data.

Example:

In the example below, a uniqueness constraint is defined on the 'publisher' table, for the target field 'name'. This means that no two records in the 'publisher' table can have the same name. The option case-sensitive is used to check the unicity of the field "name".

```
<xs:element name="publisher">
 ...
 <xs:complexType>
  <xs:sequence>
   ...
   <xs:element name="name" type="xs:string" />
   ...
  </xs:sequence>
 </xs:complexType>
 <xs:unique name="uniqueName">
  <xs:annotation>
   <xs:appinfo>
    <osd:validation>
```

```
      <caseSensitivity>sensitive</caseSensitivity>
     <severity>error</severity>
     <message>Name must be unique in table.</message>
     <message xml:lang="en-US">Name must be unique in table.</message>
     <message xml:lang="fr-FR">Le nom doit être unique dans la table.</message>
    </osd:validation>
   </xs:appinfo>
  </xs:annotation>
  <xs:selector xpath="." />
  <xs:field xpath="name" />
 </xs:unique>
</xs:element>
```

A uniqueness constraint has to be defined within a table and has the following properties:

Property	Description	Mandatory
`name` attribute	Identifies the constraint in the data model.	Yes
`xs:selector` element	Indicates the table to which the uniqueness constraint applies using a restricted XPath expression ('..' is forbidden). It can also indicate an element within the table (without changing the meaning of the constraint).	Yes
`xs:field` element	Indicates the field in the context whose values must be unique, using a restricted XPath expression. It is possible to indicate that a set of values must be unique by defining multiple `xs:field` elements.	Yes

> **Note**
>
> Undefined values (null values) are ignored on uniqueness constraints applied to single fields. On multiple fields, undefined values are taken into account. That is, sets of values are considered as being duplicated if they have the same defined and undefined values.

Additional localized validation messages can be defined using the element `osd:validation` under the elements `annotation/appinfo`. If no custom validation messages are defined, a built-in validation message will be used.

The uniqueness constraint can also be applied on an simple aggregated list: in this case, each value of the list has to be unique in the scope of the list and not in the scope of the table.

Example:

In the example below, a uniqueness constraint is defined on the 'title' table, for the target field 'printedEditions'. This means that an edition can appear only once in the list. The option case-insensitive will be used to search the field "title".

```
<xs:element name="title">
 ...
 <xs:complexType>
  <xs:sequence>
   ...
   <xs:element name="printedEditions" type="xs:string" minOccur="0" maxOccur="5"/>
   ...
  </xs:sequence>
 </xs:complexType>
 <xs:unique name="uniquePrintedEditions">
  <xs:annotation>
   <xs:appinfo>
    <osd:validation>
     <caseSensitivity>insensitive</caseSensitivity>
     <severity>error</severity>
     <message xml:lang="en-US">An edition must be referenced only once by this title</message>
     <message xml:lang="fr-FR">Une édition ne peut être référencée qu'une seule fois par ce livre</message>
    </osd:validation>
   </xs:appinfo>
```

```
  </xs:annotation>
  <xs:selector xpath="." />
  <xs:field xpath="printedEditions"/>
 </xs:unique>
</xs:element>
```

Limitations:

1. The target of the `xs:field` element must be in a table.
2. The uniqueness constraint does not apply to computed fields.
3. The uniqueness constraint cannot be applied on multiple fields that contains an aggregated list.
4. The uniqueness constraint cannot be applied on embedded lists.

See also ***Uniqueness constraint in the Java API*** `UniquenessConstraint`[API]

155.2 Extended facets

EBX® provides additional constraints that are not specified in XML Schema, but that are useful for managing master data.

In order to guarantee XML Schema conformance, these extended facets are defined under the element `annotation/appinfo/otherFacets`.

Foreign keys

EBX® allows to create a reference to an existing table by means of a specific facet. See Foreign keys [p 919] for more information.

Dynamic constraints

Dynamic constraint facets retain the semantics of XML Schema, but the value attribute is replaced with a path attribute that allows fetching the value from another element. The available dynamic constraints are:

- `length`
- `minLength`
- `maxLength`
- `maxInclusive`
- `maxExclusive`
- `minInclusive`
- `minExclusive`

Using these facets, the data model can be modified dynamically.

Example:

```
<xs:element name="amount">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:minInclusive path="/domain/Loan/Pricing/AmountMini/amount" />
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
   ...
</xs:element>
```

In this example, the boundary of the facet `minInclusive` is not statically defined. The value of the boundary comes from the node */domain/Loan/Pricing/AmountMini/amount*.

Restrictions:

- Target field cannot be an aggregated list. That is, it cannot define `maxOccurs` = 1.
- Data type of the target field must be compatible with the facet. That is, it must be:
  - of type `integer` for facets `length`, `minLength` and `maxLength`.
  - compatible with the data type of the field holding the facet for facets `maxInclusive`, `maxExclusive`, `minInclusive` and `minExclusive`.
- Target field cannot be in a table if the field holding the facet is not in a table.
- Target field must be in the same table or outside a table if the field holding the facet is in a table.
- If the target field is under one or more aggregated lists, the field holding the facet must also be under these aggregated lists. That is: the field holding the facet must be in the same list occurrence as the target field, or in a parent occurrence, so that the target field refers to a single value, from an XPath perspective.

FacetOResource constraint

This facet must be defined for every definition using the type `osd:resource`, to specify the subset of available packaged resource files as an enumeration. For more information on this type, see osd:resource type [p 904]. It has the following attributes:

`moduleName`	Indicates, using an alias, the EBX® module that contains the resource. If the resource is contained in the current module, the alias must be preceded by "wbp". Otherwise, the alias must be one of the values defined in the element `<dependencies>` in the file `module.xml`.
`resourceType`	Represents the resource type that is one of the following values: 'Image', 'JavaScript', 'Style sheet', 'HTML'.
`relativePath`	Indicates in which directory the resources will be located. This directory must be located under the directory that corresponds to the resource type. For example, for an "Image" type resource, the directory www/common/images/, located at the same level as the directory WEB-INF/ of the target module, will be used and the relative path will have to be defined from this. Furthermore, if a resource is defined in a localized directory (www/fr/ for example), it will only be taken into account if another resource with the same name is defined in the directory www/common/.

This facet has the same behavior as an enumeration facet: the values are collected by recursively listing all the files in the local path in the specified resource type directory in the specified module.

Example:

```
<xs:element name="promotion" type="osd:resource">
 <xs:annotation>
```

```
  <xs:appinfo>
   <osd:otherFacets>
    <osd:FacetOResource osd:moduleName="wbp"
     osd:resourceType="ext-images" osd:relativePath="promotion/" />
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</ xs:element>
```

For an overview of the standard directory structure of an EBX® module (Jakarta EE web application), see Module structure [p 879].

Excluding values

excludeValue constraint

This facet verifies that a value is not the same as the specified excluded value.

In this example, the empty string is excluded from the allowed values.

Example:

```
<xs:element name="roleName">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:excludeValue value="">
     <osd:validation>
      <severity>error</severity>
       <message>Please select address role(s).</message>
     </osd:validation>
    </osd:excludeValue>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType type="xs:string" />
</xs:element>
```

excludeSegment constraint

This facet verifies that a value is not included in a range of values. Boundaries are excluded.

Example:

In this example, values between 20000 and 20999 are not allowed.

```
<xs:element name="zipCode">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:excludeSegment minValue="20000" maxValue="20999">
     <osd:validation>
      <severity>error</severity>
       <message>Postal code not valid.</message>
     </osd:validation>
    </osd:excludeSegment>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType type="xs:string" />
</xs:element>
```

Enumeration constraint defined using another node

Attention

This kind of constraint is obsolete. You should use foreign key constraint [p 919]. It has limitations, in particular Quick search and sort operations in the user interface will use the raw value of the field instead of the labels defined through the constraint.

By default, an enumeration facet is described statically in XML Schema.

The content of an enumeration facet can also be provided dynamically by a list of simple elements in the data model.

Example:

In this example, the content of an enumeration facet is sourced from the node `CountryList`.

```
<xs:annotation>
 <xs:appinfo>
  <osd:otherFacets>
   <osd:enumeration osd:path="../CountryList" />
  </osd:otherFacets>
 </xs:appinfo>
</xs:annotation>
```

The referred node `CountryList`:

- Must be an aggregated list, that is, `maxOccurs` > 1.
- Must be a list of elements of the same type as the node with the enumeration facet.
- Must be a node outside a table if the node with the enumeration facet is not inside a table.
- Must be a node outside a table or in the same table as the node with the enumeration facet if the node with this enumeration is inside a table.
- If the target field is under one or more aggregated lists, the field holding the facet must also be under these aggregated lists. That is: the field holding the facet must be in the same list occurrence as the target field, or in a parent occurrence, so that the target field refers to a single value, from an XPath perspective.

Example:

```
<xs:element name="FacetEnumBasedOnList">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="CountryList" maxOccurs="unbounded">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="DE" osd:label="Germany" />
      <xs:enumeration value="AT" osd:label="Austria" />
      <xs:enumeration value="BE" osd:label="Belgium" />
      <xs:enumeration value="JP" osd:label="Japan" />
      <xs:enumeration value="KR" osd:label="Korea" />
      <xs:enumeration value="CN" osd:label="China" />
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
   <xs:element name="CountryChoice" type="xs:string">
    <xs:annotation>
     <xs:appinfo>
      <osd:otherFacets>
       <osd:enumeration osd:path="../CountryList" />
      </osd:otherFacets>
     </xs:appinfo>
    </xs:annotation>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

155.3 Programmatic facets

A programmatic constraint can be added to any XML element declaration for a simple type.

In order to guarantee XML Schema conformance, programmatic constraints are specified under the element `annotation/appinfo/otherFacets`.

Programmatic constraints

A programmatic constraint is defined by a Java class that implements the interface `Constraint`[API].

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol.

Example:

In the example below, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

```
<xs:element name="amount">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraint class="com.foo.CheckAmount">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraint>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

See also

JavaBean specifications *`Package com.orchestranetworks.schema.JavaBeans`*[API]

`JavaBeanVersion`[API]

Programmatic enumeration constraints

An enumeration constraint adds an ordered list of values to a basic programmatic constraint. This facet allows selecting a value from a list. It is defined by a Java class that implements the interface `ConstraintEnumeration`[API].

Example:

```
<xs:element name="amount">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraintEnumeration class="com.foo.CheckAmountInEnumeration">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraintEnumeration>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Attention

Quick search and sort operations in the user interface will use the raw value of the field instead of its label. Consider whether this enumeration can be replaced by a foreign key constraint [p 919] to a table that defines the same set of values.

Constraint on 'null' values

In some cases, a value is only mandatory if some conditions are satisfied, for example, if another field has a given value. In this case, the standard XML Schema attribute `minOccurs` is insufficient because it is static.

In order to check if a value is mandatory according to its context, the following requirements must be satisfied:

1. A programmatic constraint must be defined by a Java class (see above).
2. This class must implement the interface `ConstraintOnNull`[API].
3. The XML Schema cardinality attributes must specify that the element is optional (`minOccurs="0"` and `maxOccurs="1"`).

> **Note**
>
> By default, constraints on 'null' values are not checked upon user input. In order to enable a check at the input, the 'checkNullInput' property [p 948] must be set. Also, if the element is terminal, the dataset must also be activated.

Example:

```
<xs:element name="amount" minOccurs="0" maxOccurs="1">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraint class="com.foo.CheckIfNull">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraint>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

See also

JavaBean specifications *`Package com.orchestranetworks.schema.JavaBeans`*[API]

`JavaBeanVersion`[API]

Constraints on table

A constraint on table is defined by a Java class that implements the interface `ConstraintOnTable`[API]. It can only be defined on table nodes.

As additional parameters can be defined. the implemented Java class must conform to the JavaBean protocol.

Example:

In the example below, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:appinfo>
   <osd:table>
    <primaryKeys>/key</primaryKeys>
   </osd:table>
   <osd:otherFacets>
    <osd:constraint class="com.foo.checkTable">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraint>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
```

```
</xs:element>
```

Attention

For performance reasons, constraints on tables are only checked when getting the validation report of a dataset or table. This means that these constraints are not checked when updates, such as record insertions, deletions or modifications, occur on tables. However, the internal incremental validation framework will optimize the validation cost of these constraints if dependencies are defined. For more information, see Data validation [p 617].

See also

JavaBean specifications `Package com.orchestranetworks.schema.JavaBeans`[API]

`JavaBeanVersion`[API]

155.4 Control policy

Blocking and non-blocking constraints

When an update in the repository is performed, and this update adds a validation error according to a given constraint, it is possible to specify whether the new error blocks the update (and cancels the transaction) or if it is considered as non-blocking (so that the update can be committed and the error

can be corrected later). The element `blocksCommit` within the element `osd:validation` allows this specification, with the following supported values:

`onInsertUpdateOrDelete`	Specifies that the constraint must always remain valid after an operation (dataset update, dataset deletion, record creation, update or deletion). In this case, any operation that would violate the constraint is rejected and the values remain unchanged. This is the default and mandatory policy for primary key constraints, data type conversion constraints (an integer or a date must be well-written) and also structural constraints in mapped tables.
`onUserSubmit-checkModifiedValues`	Specifies that the constraint must remain valid whenever a user modifies the associated value and submits a form. In this case, any form input that would violate the constraint is rejected and the values remain unchanged. This is the default policy for all blocking constraints mentioned in the previous case. For example, a foreign key constraint is by default not blocking (a record referred to by other records can be deleted, etc.), except in the context of a form submit.
`never`	Specifies that the constraint must never block operations. In this case, any operation that would violate the constraint is allowed. In the context of the user interface, this constraint does not block the form submission if the user sets a value that violates this constraint.

On foreign key constraints, the control policy that blocks all operations does not apply to filtered records. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected and a validation error occurs.

It is not possible to specify a control policy on structural constraints that are defined in mapped tables. That is, this property is not available for fixed length, maximum length, maximum number of digits, and decimal place constraints due to the validation policy of the underlying RDBMS blocking constraints.

This property does not apply to archive imports and when merging dataspaces. That is, all blocking constraints, except structural constraints, are always disabled when importing archives and merging dataspaces.

See also

Facet validation message with severity *[p 958]*

Foreign keys *[p 919]*

XML Schema facet

The control policy is described by the element `osd:validation` in `annotation/appinfo` under the definition of the facet.

Example:

```
<xs:element name="zipCode">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:minInclusive value="1000">
    <xs:annotation>
     <xs:appinfo>
      <osd:validation>
       <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
      </osd:validation>
     </xs:appinfo>
    </xs:annotation>
   </xs:minInclusive>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

XML Schema enumeration facet

The control policy is described by the element `osd:enumerationValidation` in `annotation/appinfo` under the definition of the field.

Example:

```
<xs:element name="Gender">
 <xs:annotation>
  <xs:appinfo>
   <osd:enumerationValidation>
    <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
   </osd:enumerationValidation>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="0" osd:label="male" />
   <xs:enumeration value="1" osd:label="female" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

EBX® facet

The control policy is described by the element `osd:validation` under the definition of the facet (which is defined in `annotation/appinfo/otherFacets`).

The control policy with values `onInsertUpdateOrDelete` and `onUserSubmit-checkModifiedValues` is only available on `osd:excludeSegment`, `osd:excludeValue` and `osd:tableRef` EBX® facets.

The control policy with the value `never` can be defined on all EBX® facets.

On programmatic constraints, the control policy with the value `never` can only be set directly during the setup of the corresponding constraint. See `ConstraintContext.setBlocksCommitToNever`[API] and `ConstraintContextOnTable.setBlocksCommitToNever`[API] in the Java API for more information.

Example:

```
<xs:element name="price" type="xs:decimal">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:minInclusive path="../priceMin">
     <osd:validation>
      <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
     </osd:validation>
    </osd:minInclusive>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Check 'null' input

According to the EBX® default validation policy, in order to allow temporarily incomplete input, a mandatory element is not checked for completion upon user input. Rather, it is verified at the dataset validation only. If completion must be checked immediately upon user input, the element must additionally specify the attribute `osd:checkNullInput="true"`. This property is ignored if defined on an aggregated list (`maxOccurs` > 1).

> **Note**
>
> A value is mandatory if the data model specifies a mandatory element, either statically, using `minOccurs="1"`, or dynamically, using a constraint on 'null'. For terminal elements, mandatory values are only checked for an activated dataset. For non-terminal elements, the dataset does not need to be activated.

Example:

```
<xs:element name="amount" osd:checkNullInput="true" minOccurs="1">
 ...
</xs:element>
```

See also

Constraint on 'null' *[p 943]*

Whitespace management *[p 948]*

Empty string management *[p 950]*

EBX® whitespace management for data types

According to XML Schema (see https://www.w3.org/TR/xmlschema-2/#rf-whiteSpace), whitespace handling must follow one of the procedures *preserve*, *replace* or *collapse*:

preserve	No normalization is performed, the value is unchanged.
replace	All occurrences of `#x9` (tab), `#xA` (line feed) and `#xD` (carriage return) are replaced with `#x20` (space).
collapse	After the processing according to the replace procedure, contiguous sequences of `#x20` are then collapsed to a single `#x20`, and any leading or trailing `#x20`s are removed.

General whitespace handling

EBX® complies with the XML Schema recommendation:

- For fields of type `xs:string`, whether a primary key element or not, whitespaces are always preserved and an empty string is never converted to `null`.

- For other fields (non-`xs:string` type), whitespaces are always collapsed and empty strings are converted to `null`.

Attention

Exceptions:

- For fields of type `osd:html` or `osd:password`, whitespaces are always preserved and empty strings are converted to `null`.
- For fields of type `xs:string` that define the property `osd:checkNullInput="true"`, an empty string is interpreted as `null` at user input by EBX®.

Whitespace handling upon user input

The rules described in the previous section are applied in the user interface, but leading and trailing whitespaces are removed upon user input. That is, in the user interface, whitespaces are by default always trimmed upon user input. Other input methods (Import XML/CSV, Data services, API updates) are not trimmed from the user interface.

Attention

Exceptions:

- For fields of type `osd:password`, whitespaces are not trimmed upon user input.
- For foreign key fields, whitespaces are not trimmed upon user input.

It is possible to indicate in a data model that whitespaces should not be trimmed upon user input. The attribute `osd:trim="disable"` can be set on the fields that allow leading and trailing whitespaces upon user input.

Example:

```
<xs:element name="field" osd:trim="disable" type="xs:string">
 ...
</xs:element>
```

Whitespace handling for primary keys of type string

For primary key columns of type `xs:string`, a default EBX® constraint is defined. This constraint forbids empty strings and non-collapsed whitespace values when creating a record. That is, any record creation that would violate this constraint is rejected.

However, if the primary key node specifies its own `xs:pattern` facet, this facet overrides the default EBX® constraint. For example, the specific pattern `".*"` would accept any string, although this is not recommended.

The default constraint allows handling certain ambiguities. For example, it would be difficult for a user to distinguish between the following strings: "12 34" and "12 34". For generic values, this would not create conflicts, however, errors would occur for primary keys.

See also *Tables and relationships* *[p 913]*

Empty string management

Default conversion

For nodes of type `xs:string`, no distinction is made at user input between an empty string and a `null` value. That is, an empty string value is automatically converted to `null` at user input.

Distinction between empty strings and 'null' value

There are certain cases where the distinction is made between an empty string and the `null` value, such as when:

- A primary key defines a pattern that allows empty strings.
- An element defines a foreign key constraint and the target table has a single primary key defining a pattern that allows empty strings.
- An element defines a static enumeration that contains an empty string.
- An element defines a dynamic enumeration to another element with one of the aforementioned cases.

If the distinction is made between an empty string and a `null` value, this implies the following behaviors:

- An empty string will not be converted to `null` at user input,
- Input fields for nodes of type `xs:string` will display an additional button for setting the value of the node to `null`,
- At validation time, an empty string will be considered to be a compliant value with regard to the `minOccurs="1"` property.

Validation message threshold

It is possible to specify at the data model level the maximum number of validation messages allowed per constraint when performing a validation.

Example:

```
<xs:schema ...>
 ...
 <xs:annotation>
  <xs:appinfo>
   <osd:validation>
    <validationMessageThreshold>250</validationMessageThreshold>
   </osd:validation>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:schema>
```

The threshold is considered for each constraint defined in a data model and in each dataset validation report. When the threshold is reached by a constraint, the validation of the constraint is stopped and an error message indicating that the threshold has been reached is added to the validation report.

The validation message threshold is set by default to `1000` if it is not defined in the data model. It is not allowed to set an unlimited number of validation messages. Also, the specified validation message threshold must be greater or equal than 100.

See also *`ValidationReport.hasConstraintsWithTooManyMessages`*[API]

CHAPTER 156

Triggers and functions

EBX® data model allows to define triggers and computed fields. It also provides auto-incremented fields

This chapter contains the following topics:

1. Computed values
2. Triggers
3. Auto-incremented values

156.1 Computed values

By default, data is read and persisted in the XML repository. Nevertheless, data may be the result of a computation and/or external database access, for example, an RDBMS or a central system.

EBX® allows taking into account other data in the current dataset context.

This is made possible by defining *computation rules*.

A computation rule is specified in the data model using the `osd:function` element (see example below).

- The value of the *class* attribute must be the qualified name of a Java class that implements the Java interface `ValueFunction`[API]
- Additional parameters may be specified at the data model level, in which case the JavaBean convention is applied.

Example:

```
<xs:element name="computedValue">
 <xs:annotation>
  <xs:appinfo>
   <osd:function class="com.foo.ComputeValue">
    <param1>...</param1>
    <param...n>...</param...n>
   </osd:function>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Disabling validation

In some cases, it can be useful to disable the validation of computed values if the execution of a function is time-consuming. Indeed, if the function is attached to a table with N records, then it will

be called N times when validating this table. The property `osd:disableValidation= "true"` specified in the data model allows to disable the validation of a computed value (see example below).

Example:

```
<xs:element name="computedValue" osd:disableValidation="true">
 <xs:annotation>
  <xs:appinfo>
   <osd:function class="com.foo.ComputeValue">
    ...
   </osd:function>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

156.2 Triggers

Datasets or table records can be associated with methods that are automatically executed when some operations are performed, such as creations, updates, or deletions.

In the data model, these triggers must be declared under the `annotation/appinfo` element using the `osd:trigger` element.

Trigger on dataset

For dataset triggers, a Java class that extends the abstract class `InstanceTrigger`[API] must be declared inside the element `osd:trigger`.

In the case of dataset triggers, it is advised to define `annotation/appinfo/osd:trigger` tags just under the `root` element of the data model.

Example:

```
<xs:element name="root" osd:access="--">
   ...
   <xs:annotation>
  <xs:appinfo>
   <osd:trigger class="com.foo.MyInstanceTrigger">
    <param1>...</param1>
    <param...n>...</param...n>
   </osd:trigger>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Trigger on table

For the definition of table record triggers, a Java class that extends the abstract class `TableTrigger`[API] must be defined inside the `osd:trigger` element. It is advised to define the `annotation/appinfo/osd:trigger` elements just under the element describing the associated table or table type.

Examples:

On a table element:

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:appinfo>
   <osd:table>
    <primaryKeys>/key</primaryKeys>
   </osd:table>
   <osd:trigger class="com.foo.MyTableTrigger" />
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

On a table type element:

```
<xs:complexType name="MyTableType">
   ...
   <xs:annotation>
    <xs:appinfo>
    <osd:trigger class="com.foo.MyTableTrigger">
    <param1>...</param1>
    <param...n>...</param...n>
    </osd:trigger>
    </xs:appinfo>
   </xs:annotation>
   ...
</xs:complexType>
```

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol. In the example above, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

156.3 Auto-incremented values

It is possible to define auto-incremented values. Auto-incremented values are only allowed inside tables, and they must be of the type `xs:int` or `xs:integer`.

An auto-increment is specified in the data model using the element `osd:autoIncrement` under the element `annotation/appinfo`.

Example:

```
<xs:element name="autoIncrementedValue" type="xs:int">
 <xs:annotation>
  <xs:appinfo>
   <osd:autoIncrement />
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Also, there are two optional elements, `start` and `step`:

- The `start` attribute specifies the first value for this auto-increment. If this attribute is not specified, then the value `1` is set by default.
- The `step` attribute specifies the step for the next value to be generated by the auto-increment. If this attribute is not specified, then the value *1* is set by default.

Example:

```
<xs:element name="autoIncrementedValue" type="xs:int">
 <xs:annotation>
  <xs:appinfo>
   <osd:autoIncrement>
    <start>100</start>
    <step>5</step>
   </osd:autoIncrement>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

A field specifying an `osd:autoIncrement` has the following behavior:

- The computation and allocation of the field value are performed whenever a new record is inserted and the field value is undefined.
- No allocation is performed if a programmatic insertion already specifies a non-`null` value. For example, if an archive import or an XML import specifies the value, that value is preserved.

  Consequently, the allocation is not performed for a record insertion in occulting or overwriting modes.

- A newly allocated value is, whenever possible, unique in the scope of the repository. More precisely, the uniqueness of the allocation spans over all the datasets of the data model, and it also spans over all the dataspaces. The latter case allows the merge of a dataspace into its parent with a reasonable guarantee that there will be no conflict if the `osd:autoIncrement` is part of the records' primary key.

  This principle has a very specific limitation: when a mass update transaction that specifies values is performed at the same time as a transaction that allocates a value on the same field, it is possible that the latter transaction will allocate a value that will be set by the first transaction (there is no locking between different dataspaces).

Internally, the auto-increment value is stored in the 'Auto-increments' table of the repository. In the user interface, it can be accessed by administrators in the 'Administration' area. This field is automatically updated so that it defines the greatest value ever set on the associated `osd:autoIncrement` field, in any instance or dataspace in the repository. This value is computed, taking into account the max value found in the table being updated.

In certain cases, for example when multiple environments have to be managed (development, test, production), each with different auto-increment ranges, it may be required to avoid this "max value" check. This particular behavior can be achieved using the `disableMaxTableCheck` property. It is generally not recommended to enable this property unless it is absolutely necessary, as this could generate conflicts in the auto-increment values. However, this property can be set in the following ways:

- Locally, by setting a parameter element in the auto-increment declaration: `<disableMaxTableCheck>true</disableMaxTableCheck>`,
- For the whole data model, by setting `<osd:autoIncrement disableMaxTableCheck="true"/>` in the element `xs:appinfo` of the data model declaration, or
- Globally, by setting the property `ebx.autoIncrement.disableMaxTableCheck=true` in the EBX® main configuration file.

  See TIBCO EBX® main configuration file [p 571].

> **Note**
>
> When this option is enabled globally, it becomes possible to create records in the table of auto-increments, for example by importing from XML or CSV. If this option is not selected, creating records in the table of auto-increments is prohibited to ensure the integrity of the repository.

CHAPTER 157

Labels and messages

TIBCO EBX® allows to have custom labels and error messages for data models to be displayed in the interface.

This chapter contains the following topics:

1. Label and description
2. Enumeration labels
3. Mandatory error message (osd:mandatoryErrorMessage)
4. Conversion error message
5. Facet validation message with severity

157.1 Label and description

A label and a description can be added to each node in a data model. In EBX®, each node is displayed with its label. If no label is defined, the name of the element is used.

Two different notations can be used:

Full	The label and description are defined by the elements `<osd:label>` and `<osd:description>` respectively.
Simple	The label is extracted from the text content, ending at the first period ('.'), with a maximum of 60 characters. The description uses the remainder of the text.

The description may also have a hyperlink, either a standard HTML `href` to an external document, or a link to another node of the adaptation within EBX®.

- When using the `href` notation or any other HTML, it must be properly escaped.
- EBX® link notation is not escaped and must specify the path of the target, for example:

  ```
  <osd:link path="../misc1">Link to another node in the adaptation</osd:link>
  ```

Example:

```
<xs:element name="misc1" type="xs:string">
 <xs:annotation>
  <xs:documentation>
   Miscellaneous 1. This is the description of miscellaneous element #1.
   Click <a href="https://www.tibco.com" target="_blank">here</a>
   to learn more.
```

```
  </xs:documentation>
 </xs:annotation>
</xs:element>
<xs:element name="misc2" type="xs:string">
 <xs:annotation>
  <xs:documentation>
   <osd:label>
    Miscellaneous 2
   </osd:label>
   <osd:description>
    This is the miscellaneous element #2 and here is a
    <osd:link path="../misc1"> link to another node in the
     adaptation</osd:link>.
   </osd:description>
  </xs:documentation>
 </xs:annotation>
</xs:element>
```

If a node points to a named type, then the label of the node replaces the label of the named type. The same mechanism applies to the description of the node (element `osd:description`).

> **Note**
>
> Regarding whitespace management, the label of a node is always *collapsed* when displayed. That is, contiguous sequences of blanks are collapsed to a single blank, and leading and trailing blanks are removed. In descriptions, however, whitespaces are always *preserved*.

Dynamic labels and descriptions

As an alternative to statically defining the localized labels and descriptions for each node, it is possible to specify a Java class that programmatically determines the labels and descriptions for the nodes of the data model. To define the class, include the element `osd:documentation`, with the attribute `class` in the data model. It is possible to pass JavaBean properties using nested parameter elements.

Example:

```
<xs:schema ...>
 <xs:annotation>
  <xs:appinfo>
   <osd:documentation class="com.foo.MySchemaDocumentation">
    <param1>...</param1>
    <param2>...</param2>
   </osd:documentation>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:schema ...>
```

The labels and descriptions that are provided programmatically take precedence over the ones defined locally on individual nodes.

See also *`SchemaDocumentation`*[API]

157.2 Enumeration labels

In an enumeration, a simple, non-localized label can be added to each enumeration element, using the attribute `osd:label`.

> **Attention**
>
> Labels defined for an enumeration element are always `collapsed` when displayed.

Example:

```
<xs:element name="Service" maxOccurs="unbounded">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="1" osd:label="Blue" />
   <xs:enumeration value="2" osd:label="Red" />
   <xs:enumeration value="3" osd:label="White" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

It is also possible to fully localize the labels using the standard `xs:documentation` element. If both non-localized and localized labels are added to an enumeration element, the non-localized label will be displayed in any locale that does not have a label defined.

Example:

```
<xs:element name="access" minOccurs="0">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="readOnly">
    <xs:annotation>
     <xs:documentation xml:lang="en-US">
      read only
     </xs:documentation>
     <xs:documentation xml:lang="fr-FR">
      lecture seule
     </xs:documentation>
    </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value="readWrite">
    <xs:annotation>
     <xs:documentation xml:lang="en-US">
      read/write
     </xs:documentation>
     <xs:documentation xml:lang="fr-FR">
      lecture écriture
     </xs:documentation>
    </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value="hidden">
    <xs:annotation>
     <xs:documentation xml:lang="en-US">
      hidden
     </xs:documentation>
     <xs:documentation xml:lang="fr-FR">
      masqué
     </xs:documentation>
    </xs:annotation>
   </xs:enumeration>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

157.3 Mandatory error message (osd:mandatoryErrorMessage)

If the node specifies the attribute `minOccurs="1"` (default behavior), then an error message, which must be provided, is displayed if the user does not complete the field. This error message can be defined specifically for each node using the element `osd:mandatoryErrorMessage`.

Example:

```
<xs:element name="birthDate" type="xs:date">
 <xs:annotation>
  <xs:documentation>
   <osd:mandatoryErrorMessage>
    Please give your birth date.
   </osd:mandatoryErrorMessage>
  </xs:documentation>
 </xs:annotation>
</xs:element>
```

The mandatory error message can be localized:

```
<xs:documentation>
 <osd:mandatoryErrorMessage xml:lang="en-US">
  Name is mandatory
 </osd:mandatoryErrorMessage>
 <osd:mandatoryErrorMessage xml:lang="fr-FR">
  Nom est obligatoire
 </osd:mandatoryErrorMessage>
</xs:documentation>
```

> **Note**
>
> Regarding whitespace management, the enumeration labels are always *collapsed* when displayed.

157.4 Conversion error message

For each predefined XML Schema element, it is possible to define a specific error message if the user entry has an incorrect format.

Example:

```
<xs:element name="email" type="xs:string">
 <xs:annotation>
  <xs:documentation>
   <osd:ConversionErrorMessage xml:lang="en-US">
    Please enter a valid email address.
   </osd:ConversionErrorMessage>
   <osd:ConversionErrorMessage xml:lang="fr-FR">
    Saisissez un e-mail valide.
   </osd:ConversionErrorMessage>
  </xs:documentation>
 </xs:annotation>
</xs:element>
```

157.5 Facet validation message with severity

The validation message that is displayed when the value of a field does not comply with a constraint can define a custom severity, a default non-localized message, and localized message variants. If no severity is specified, the default level is `error`. Blocking constraints *must* have the severity `error`.

XML Schema facet (osd:validation)

The validation message is described by the element `osd:validation` in `annotation/appinfo` under the definition of the facet.

Example:

```
<xs:element name="zipCode">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <!--facet is not localized, but validation message is localized-->
   <xs:minInclusive value="01000">
    <xs:annotation>
     <xs:appinfo>
      <osd:validation>
       <severity>error</severity>
       <message>Non-localized message.</message>
       <message xml:lang="en-US">English error message.</message>
       <message xml:lang="fr-FR">Message d'erreur en français.</message>
      </osd:validation>
     </xs:appinfo>
    </xs:annotation>
   </xs:minInclusive>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

XML Schema enumeration facet (osd:enumerationValidation)

The validation message is described by the element `osd:enumerationValidation` in `annotation/appinfo` under the definition of the field.

Example:

```
<xs:element name="Gender">
 <xs:annotation>
  <xs:appinfo>
   <osd:enumerationValidation>
    <severity>error</severity>
    <message>Non-localized message.</message>
    <message xml:lang="en-US">English error message.</message>
    <message xml:lang="fr-FR">Message d'erreur en français.</message>
   </osd:enumerationValidation>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="0" osd:label="male" />
   <xs:enumeration value="1" osd:label="female" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

EBX® facet (osd:validation)

The validation message is described by the element `osd:validation` under the definition of the facet (which is defined in `annotation/appinfo/otherFacets`).

Example:

```
<xs:element name="price" type="xs:decimal">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:minInclusive path="../priceMin">
     <osd:validation>
      <severity>error</severity>
      <message>Non-localized message.</message>
      <message xml:lang="en-US">English error message.</message>
      <message xml:lang="fr-FR">Message d'erreur en français.</message>
     </osd:validation>
    </osd:minInclusive>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

CHAPTER 158

Additional properties

This chapter contains the following topics:

1. Default values
2. Access properties
3. Information
4. Default view
5. Comparison mode
6. Apply last modifications policy
7. Categories

158.1 Default values

In a data model, it is possible to specify a default value for a field using the attribute `default`. This property is used to assign a default value if no value is defined for a field.

The default value is displayed in the user input field at the creation time. That is, the default value will be displayed when creating a new record or adding a new occurrence to an aggregated list.

Example:

In this example, the element specifies a default string value.

```
<xs:element name="fieldWithDefaultValue" type="xs:string" default="aDefaultValue" />
```

158.2 Access properties

The attribute `osd:access` defines the access mode, that is, whether the data of a particular data model node can be read and/or written. This attribute must have one of the following values: `RW`, `R-`, `CC` or `--`.

For each XML Schema node, three types of adaptation are possible:

1. *Adaptation terminal node*

   This node is displayed with an associated value in TIBCO EBX®.

   When accessed using the method `Adaptation.get()`, it uses the adaptation search algorithm.
2. *Adaptation non-terminal node*

This node is a complex type that is only displayed in EBX® if it has one child node that is also an adaptation terminal node. It has no value of its own.

When accessed using the method `Adaptation.get()`, it returns `null`.

3. *Non-adaptable node*

   This node is not an adaptation terminal node and has no child adaptation terminal nodes. This node is never displayed in EBX®.

   When accessing using the method `Adaptation.get()`, it returns the node default value if one is defined, otherwise it returns `null`.

See also *`Adaptation`*[API]

Access mode	Behavior
`RW`	*Adaptation terminal node*: value can be read and written in EBX®.
`R-`	*Adaptation terminal node*: value can only be read in EBX®.
`CC`	*Cut*: This is not an adaptation terminal node and none of its children are adaptation terminal nodes. This "instruction" has priority over any child node regardless of the value of their *access* attribute.
`--`	If the node is a simple type, it is not adaptable. If the node is a complex type, it is not an adaptation terminal node and does not define any child nodes. The `root` node of a data model must specify this access mode.
Default	If the *access* attribute is not defined: • If the node is a computed value, it is considered to be `R-` • If the node is a simple type and its value is not computed, it is considered to be `RW` • If the node is an aggregated list, it is then a terminal value and is considered to be `RW` • Otherwise, it is not an adaptation terminal node and it does not define anything about its child nodes.

Example:

In this example, the element is adaptable because it is an adaptation terminal node.

```
<xs:element name="proxyIpAddress" type="xs:string" osd:access="RW"/>
```

158.3 Information

The element `osd:information` allows specifying additional information.

See also *`SchemaNode.getInformation`*[API]

Example:

```
<xs:element name="misc" type="xs:string">
  <xs:annotation>
```

```
    <xs:appinfo>
   <osd:information>
    This is the text information of miscellaneous element.
   </osd:information>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

158.4 Default view

Hiding a field or a table in the default view

It is possible for a table or field inside a table to be hidden by default in EBX® by using the element `osd:defaultView/hidden`. This property is used to hide elements from the default view of a dataset without defining specific access permissions. That is, elements hidden by default will not be visible in any default forms and views, whether tabular or hierarchical, generated from the structure of the associated data model.

Attention

- If an element is configured to be hidden in the default view of a dataset, then the access permissions associated with this field will not be evaluated.
- It is possible to display a field that is hidden in the default view of a dataset by defining a view. Only in this case will the access permissions associated with this field be evaluated to determine whether the field will be displayed or not.
- It is not possible to display a table that is hidden in the default view of a dataset (in the navigation pane).

Example:

In this example, the element is hidden in the default view of a dataset.

```
<xs:element name="hiddenField" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hidden>true</hidden>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

Hiding groups and fields in views

It is possible for a field or a group to be hidden in all views of a table by using the element `osd:defaultView/hiddenInViews`. This property is used to hide elements from the tabular (including the default tabular view) and hierarchical views of a dataset without defining specific access permissions. That is, hidden elements will not be visible in any views, whether tabular or hierarchical, created from the structure of the associated data model. However, hidden elements in views will be displayed in forms.

To specify whether or not to hide an element in all views, use the `osd:defaultView/hiddenInViews="true|false"` element.

If this property is set to `true`, then the element will not be selectable when creating a custom view. As a consequence, the element will not be displayed in all views of a table in a dataset.

If a group is configured as hidden in views, then all the fields nested under this group will not be displayed respectively in the views of the table.

Hiding a field in structured search tools

To specify whether or not to hide an element in structured search tools, use the element `osd:defaultView/hiddenInSearch="true|false|textSearchOnly"`.

If this property is set to `true`, then the field will not be selectable in the text and typed search tools of a dataset.

If this property is set to `textSearchOnly`, then the field will not be selectable only in the text search of a dataset but will be selectable in the typed search.

> **Note**
>
> If a group is configured as hidden in search tools or only in the text search, then all the fields nested under this group will not be displayed respectively in the search tools or only in the text search.
>
> In all cases, the field will remain searchable in the quick search tool. A field can be excluded from all search tools, including the quick search, by defining a specific search strategy.
>
> **See also** *Excluding a field from search ('Void' indexing)* *[p 515]*

Example:

```
<xs:element name="hiddenFieldInSearch" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hiddenInSearch>true</hiddenInSearch>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the text and typed search tools of a dataset.

```
<xs:element name="hiddenFieldOnlyInTextSearch" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hiddenInSearch>textSearchOnly</hiddenInSearch>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

In this example, the element is hidden only in the text search tool of a dataset.

Hiding a field in Data Services

To specify whether or not to hide an element in data services, use the element `osd:defaultView/hiddenInDataServices`. For more information, see Disabling fields from data model [p 1079].

> **Note**
>
> - If a group is configured as being hidden, then all the fields nested under this group will be considered as hidden by data services.

Example:

```
<xs:element name="hiddenFieldInDataService" type="xs:string" minOccurs="0"/>
```

```
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hiddenInDataServices>true</hiddenInDataServices>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the Data Service select operation.

Defining a view for the combo box selector of a foreign key

It is possible to specify a published view that will be used to display the target table or the hierarchical view of a foreign key for a smoother selection. If a view has been defined, the selector will be displayed in the user interface in the combo box of this foreign key. The definition of a view can be done by using the XML Schema element `osd:defaultView/widget/viewForAdvancedSelection`.

> **Note**
>
> - This property can only be defined on foreign key fields.
> - The published view must be associated with the target table of the foreign key.
> - If the published view does not exist, then the advanced selection is not available in the foreign key field.

Example:

In this example, the name of a published view is defined to display the target table of a foreign key in the advanced selection.

```
<xs:element name="catalog_ref" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:tableRef>
     <tablePath>/root/Catalogs</tablePath>
    </osd:tableRef>
     </osd:otherFacets>
    <osd:defaultView>
    <widget>
     <viewForAdvancedSelection>catalogView</viewForAdvancedSelection>
    </widget>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

See Combo-box selector [p 65] for more information.

Customizing a default widget

A widget can be defined using the data model assistant. See Default view > Widget [p 65] for more information.

Customizing REST data services

Default view configuration is managed in REST data services through the session channel [p 1116].

158.5 Comparison mode

The attribute `osd:comparison` can be included on a terminal node element in order to set its comparison mode. This mode controls how differences are detected for the element during comparisons. The possible values for the attribute are:

default	Element is visible during comparisons of its data.
ignored	No changes will be detected when comparing two versions of the content in records or datasets. During a merge, the data values of an ignored element are not merged when contents are updated, even if the values are different. For new content, the values of ignored elements are merged. During an archive import, values of ignored elements are not imported when contents are updated. For new content, the values of ignored elements are imported.

Note

- If a group is configured as being ignored during comparisons, then all the fields nested under this group will also be ignored.
- If a terminal field does not include the attribute `osd:comparison`, then it will be included by default during comparisons.

Restrictions:

- This property cannot be defined on non-terminal fields.
- Primary key fields cannot be ignored during comparison.

Example:

In this example, the first element is explicitly ignored during comparison, the second element is explicitly included.

```
<xs:element name="fieldExplicitlyIgnoredInComparison"
 type="xs:string" minOccurs="0" osd:comparison="ignored"/>
<xs:element name="fieldExplicitlyNotIgnoredInComparison"
 type="xs:string" minOccurs="0" osd:comparison="default"/>
```

158.6 Apply last modifications policy

The attribute `osd:applyLastModification` can be defined on a terminal node element in order to specify if this element must be included or not in the 'apply last modifications' service that can be executed in a table of a dataset.

The possible values for the attribute are:

default	Last modifications can be applied to this element.
ignored	This element is ignored from the apply last modifications service. That is, the last modification that has been performed on this element cannot be applied to other records.

Note

- If a group is configured as being ignored by the 'apply last modifications' service, then all fields nested under this group will also be ignored.
- If a terminal field does not include the attribute `osd:applyLastModification`, then it will be included by default in the apply last modifications service.

Restriction:

- This property cannot be defined on non-terminal fields.

Example:

In this example, the first element is explicitly ignored in the 'apply last modifications' service, the second element is explicitly included.

```
<xs:element name="fieldExplicitlyIgnoredInApplyLastModification"
 type="xs:string" minOccurs="0" osd:applyLastModification="ignored"/>
<xs:element name="fieldExplicitlyNotIgnoredApplyLastModification"
 type="xs:string" minOccurs="0" osd:applyLastModification="default"/>
```

158.7 Categories

Categories can be used for "filtering", by restricting the display of data model elements.

To create a category, add the attribute `osd:category` to a table node in the data model XSD.

Filters on data

In the example below, the attribute `osd:category` is added to the node in order to create a category named *mycategory*.

```
<xs:element name="rebate" osd:category="mycategory">
   <xs:complexType>
  <xs:sequence>
    <xs:element name="label" type="xs:string"/>
    <xs:element name="beginDate" type="xs:date"/>
    <xs:element name="endDate" type="xs:date"/>
    <xs:element name="rate" type="xs:decimal"/>
  </xs:sequence>
   </xs:complexType>
</xs:element>
```

To activate a defined category filter on a dataset in the user interface, select **Actions** > **Categories** > ***<category name>*** from the navigation pane.

Predefined categories

Two categories with localized labels are predefined:

- Hidden

  An instance node, including a table node itself, is hidden in the default view, but can be revealed by selecting **Actions > Categories >** ***[hidden nodes]*** from the navigation pane.

  A table record node is always hidden.

- Constraint (deprecated)

Restriction

Categories do not apply to table record nodes, except the category 'Hidden'.

CHAPTER 159

Data services

This chapter details how WSDL operations' names related to a table are defined and managed by TIBCO EBX®.

This chapter contains the following topics:

1. Definition
2. Configuration
3. Publication
4. WSDL and table operations
5. Limitations

159.1 Definition

EBX® generates a WSDL that complies with the W3C Web Services Description Language 1.1 standard. By default, WSDL operations refer to a table using the last element of the table path. A WSDL operation name is composed of the action name (prefix) and the table name (suffix). It is possible to refer to tables in WSDL operations using entity names. Entity names allow to use unique names instead of the last element of tables' paths. Entity names override the suffix operations' names.

See also *Entity mappings using the Data Model Assistant* *[p 92]*

159.2 Configuration

Embedded data model

Entity names used as WSDL suffix operations' names are embedded in EBX®'s repository and linked to a publication. That is, when publishing an embedded data model, the list of entity names can be defined in the data model definition, under the 'Configuration > 'Entity mappings' table and managed by EBX®.

Packaged data model

Entity names used as WSDL suffix operations' names are defined in a dedicated `XML` document file and must be named as the data model and end with the keyword `_entities`. For instance, if a data model is named `catalog.xsd`, then the XML document containing the configuration of the WSDL operations' names overrided will be named `catalog_entities.xml`. This `XML` document must also be

located in the same location as the data model. The `XML` document is automatically loaded by EBX® if a file that matches this pattern is found when compiling a data model.

159.3 Publication

Entity names are validated at compilation time and contain a list of couples containing `Path` with a unique table name. Checked validation rules are:

- The `path` is not unique,
- The table name contains a syntax error,
- The table name is not unique in the `XML` document.

159.4 WSDL and table operations

WSDL Generator

An additional validation rule has been added: a unicity check is systematically applied to table names. The SOAP operation name is composed of the operation type as a prefix and, by default, of the table name (last step of the table path) as a suffix. A dataset can contain several identical table names but with different paths. It is possible to override table names that are not unique in order to guarantee the unicity.

SOAP operations

When an operation request on table has been invoked from the SOAP connector, the target table is retrieved by priority, the name corresponds to:

1. an overridden table name,
2. the last step of the table path.

See also *Data services* *[p 1060]*

159.5 Limitations

WSDL operations' names are not available with `external` data models.

CHAPTER 160

Toolbars

This chapter details how toolbars are defined and managed by TIBCO EBX®.

This chapter contains the following topics:

1. Definition
2. Using toolbars

160.1 Definition

Toolbars allow to customize the buttons and menus to display when accessing a table view, a hierarchical view, or a record form.

Toolbars can only be created and published using the *Data Model Assistant* and are available only on `embedded` and `packaged` data models.

For embedded data models, toolbars are embedded in EBX®'s repository and linked to a publication. That is, when publishing an embedded data model, the toolbars defined in the data model are embedded with the publication of the data model and managed by EBX®.

For packaged data models, toolbars are defined in a dedicated `XML` document and must be named as the data model and end with the keyword `_toolbars`. For instance, if a data model is named `catalog.xsd` then the XML document containing the definition of the toolbars must be named `catalog_toolbars.xml`. This `XML` document must also be placed in the same location as the data model. The toolbar document is automatically loaded by EBX® if a file complying with this pattern is found when compiling a data model.

See also

Configuring toolbars using the Data Model Assistant *[p 86]*

Using toolbars in data models *[p 917]*

Toolbar API `ToolbarFactory`[API]

160.2 Using toolbars

Toolbars can be used on `tables` and `associations`.

On tables, it is possible to specify the toolbar to display:

- On the top of a tabular view
- On each row of a tabular view

- On the top of a record form
- On the top of a hierarchical view.
- On the top of a tile view.
- On each card of a tile view.

On associations, it is possible to specify the toolbar to display:

- On top of the tabular view of the association
- On each row of the tabular view of the association

See also

Using toolbars *[p 917]*

Associations *[p 923]*

CHAPTER 161

Custom forms

This chapter details how custom forms are defined and managed by TIBCO EBX®.

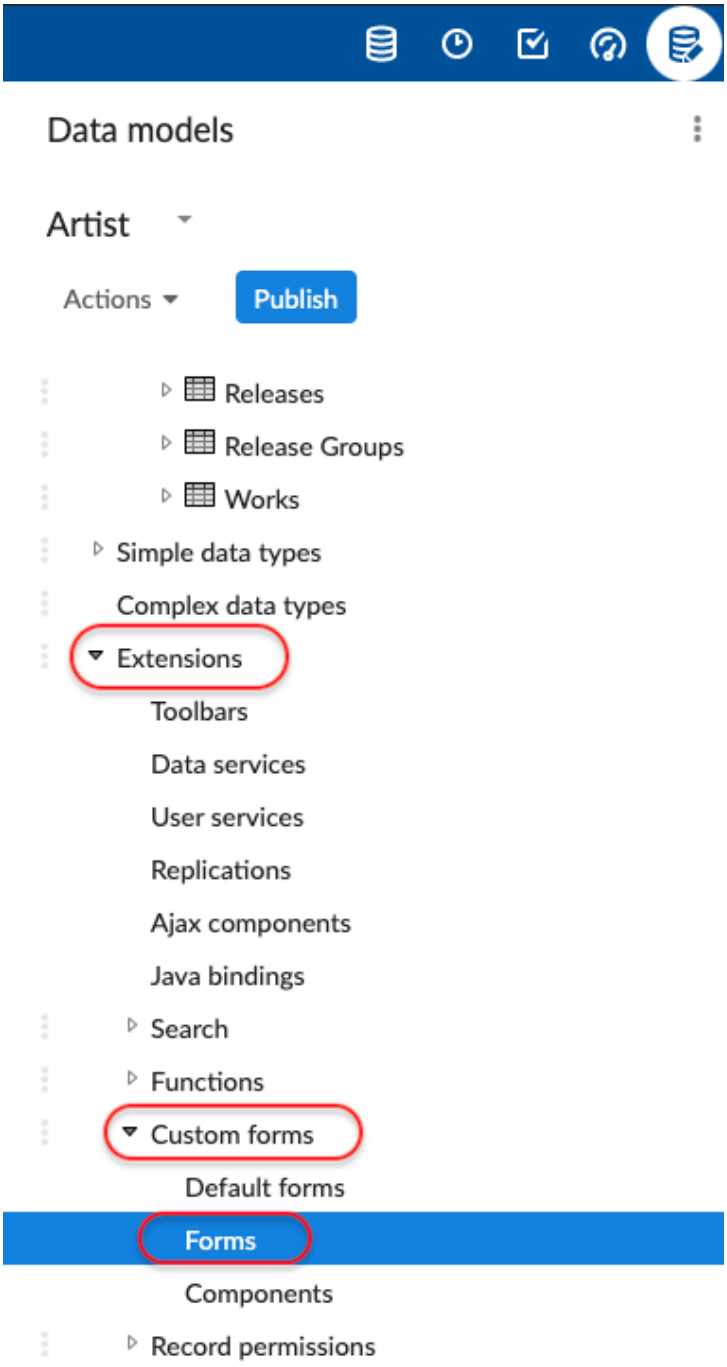
Related concepts *Interface customization* *[p 1024]*

This chapter contains the following topics:

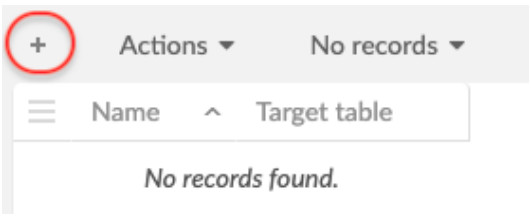
1. Access
2. Forms and components
3. The editor
4. Blocks

161.1 **Access**

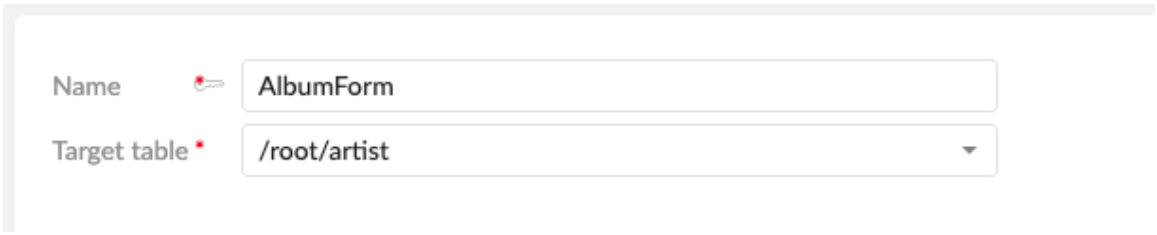
To access it, go to: Data Models > Extensions > Custom forms > Forms



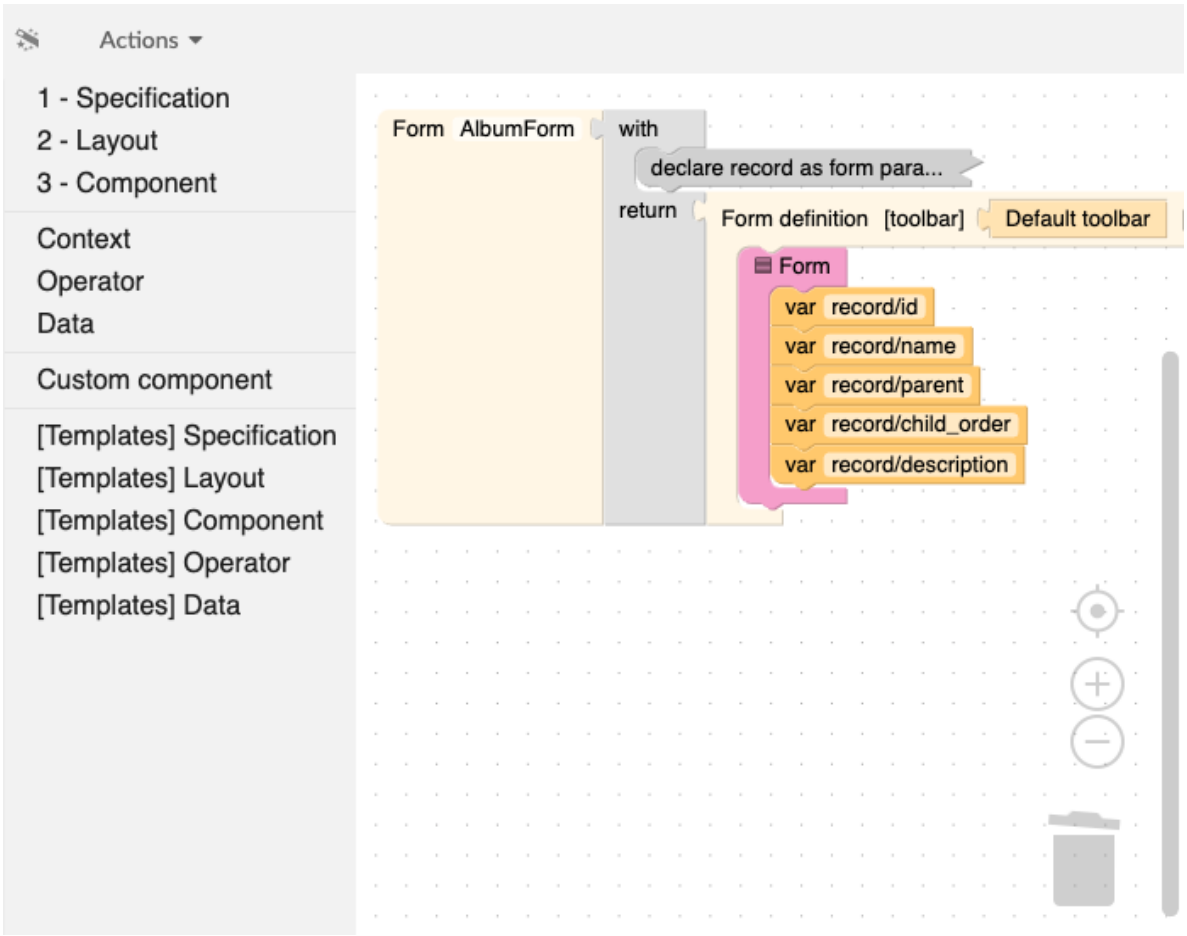
Click on the [+] button to create a form.



Name your new form and indicate the table on which it will be available.



Pressing the "Save" button will redirect you to the layout designer.



161.2 **Forms and components**

Both forms and components can be created in the 'Custom forms' data model extension.

- **A form describes the layout of a record.** It can access contextual information such as the record or the input parameters. When creating a form, the user is asked to provide its target table. Once created, **it can be declared as the default form of the table** either in the 'Default form' table of the extension, or in the 'Extensions' tab of the target table. Forms that are not the default for their table will not be used.
- **A component is a reusable fragment** that can be shared between forms and other components. Unlike forms, components don't have access to contextual information. If such information is needed, it must be provided explicitly by its caller.

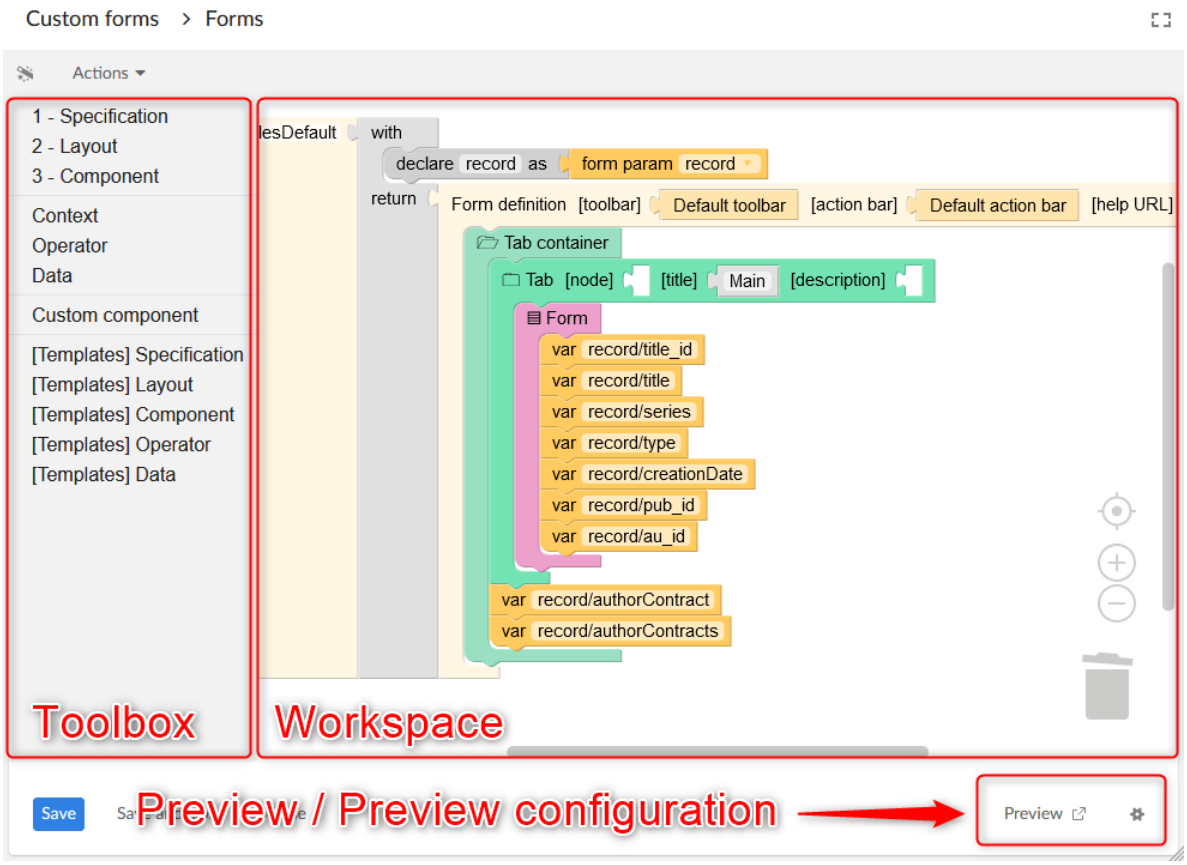
161.3 The editor

A form or component is just an imbrication of blocks. Blocks can represent a concrete graphical element or some piece of logic, allowing to have different layouts based on conditions like permissions, language, etc.

The workspace contains the description of the form, inside a predefined root block. Any block that is not connected to this root is grayed out, to mark it as inactive. **To move a block**, it must be dragged and dropped. Dragging a block also drags the blocks connected below it. If only one block has to be moved, hold the control key before clicking on it. **Right-clicking** on a block in the workspace also shows a list of options such as expand/collapse, comment, help, etc.

The toolbox on the left displays a list of categories. By clicking on a category, the blocks it contains are displayed. Some categories have a related 'Template' section. This section provides some combinations of the blocks of the section and can be considered as useful shortcuts or samples.

On the bottom right of the screen are the 'Preview' and 'Configure preview' buttons. These can be used to see what the form will look like.



161.4 Blocks

Here is the list of all the blocks that appear in the toolbox.

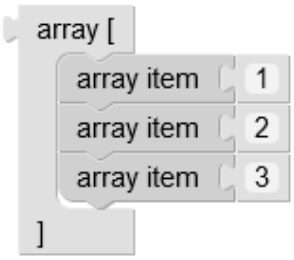
Arithmetic operator

Operations on two integers.



Array

A list of items.



See also *Array item* *[p 978]*

Array item

Adapter to make expression blocks into arrays.

See also *Array* *[p 977]*

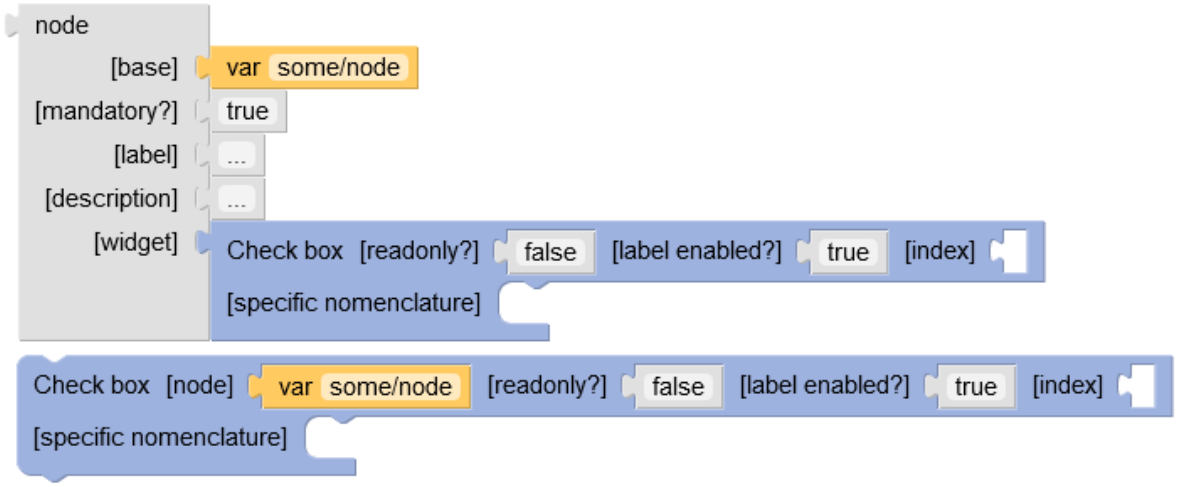
Boolean operator

Operations on two booleans.



Check box

Displays the checkbox widget.

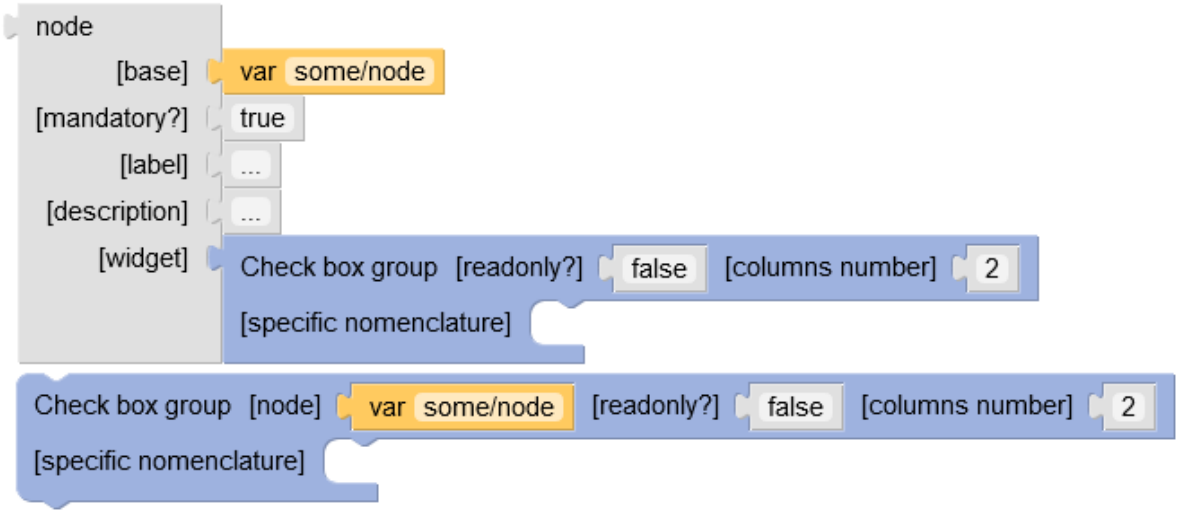


Parameters

Input	Description
`node`	The node for which to display the widget.
`readonly?`	Indicates if the widget should be in readonly mode.
`label enabled?`	If set, indicates if the item label is to be added next to the widget.
`index`	The index for this enumeration item.
`specific nomenclature`	If set, overrides the model-driven nomenclature.

Check box group

Displays the checkbox group widget.



Parameters

Input	Description
`node`	The node for which to display the widget.
`readonly?`	Indicates if the widget should be in readonly mode.
`columns number`	If set, defines the number of columns to use for the layout of the checkboxes.
`specific nomenclature`	If set, overrides the model-driven nomenclature.

Children

Returns the list of the children of the given node.

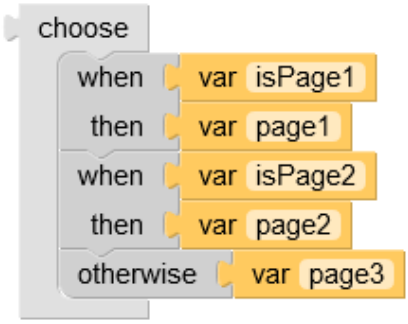


Parameters

Input	Description
`node`	The complex node from which to extract children.

Choose

A block returning the content of one of its inner 'when'/'otherwise' blocks.



Parameters

Input	Description
`content`	A list of 'when' blocks and optionally a final 'otherwise' block.

See also

When *[p 1009]*

Otherwise *[p 1000]*

Close button

Standard 'Close' button.

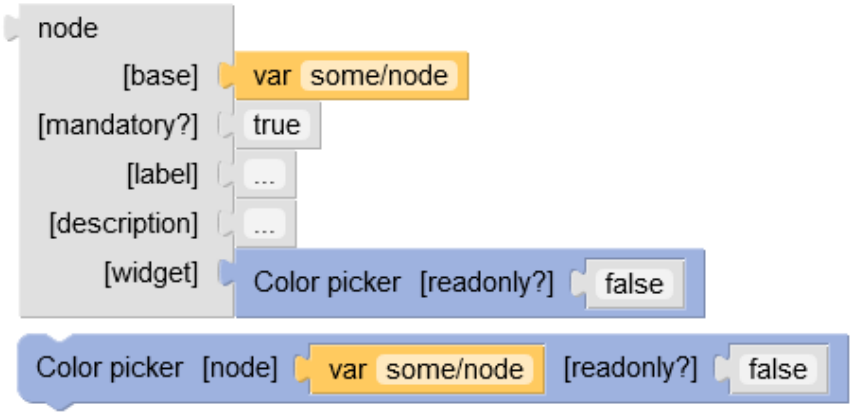


Parameters

Input	Description
`label`	If set, overrides the default label.
`default?`	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Color picker

Displays the color picker widget.

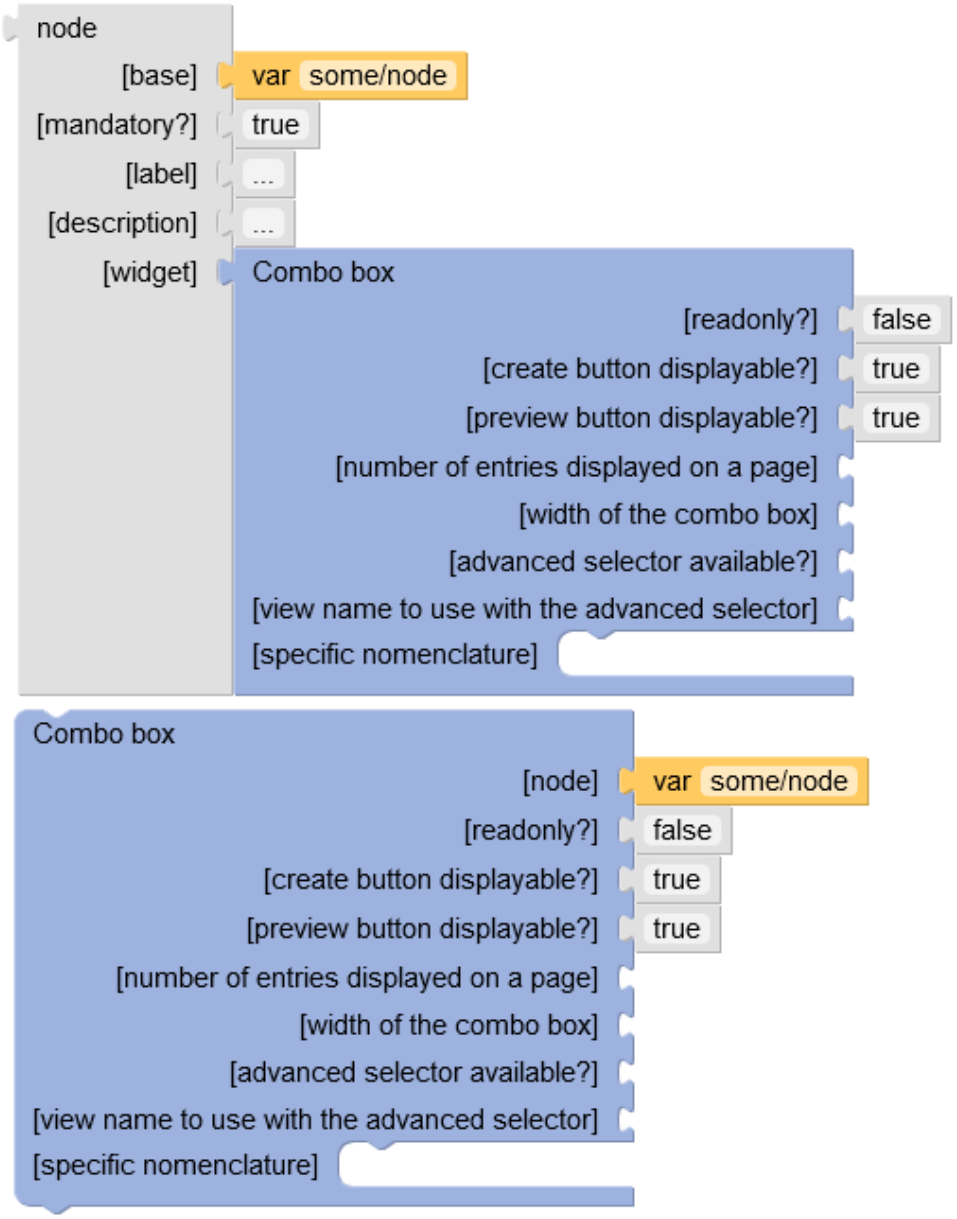


Parameters

Input	Description
`node`	The node for which to display the widget.
`readonly?`	Indicates if the widget should be in readonly mode.

Combo box

Displays the combo box widget.



Parameters

Input	Description
`node`	The node for which to display the widget.
`readonly?`	Indicates if the widget should be in readonly mode.
`create button displayable?`	If set, defines if the create button should be displayed when the underlying node is a foreign key.
`preview button displayable?`	If set, defines if the preview button should be displayed when the underlying node is a foreign key.
`number of entries displayed on a page`	If set, defines the number of entries on each page of the drop-down list.
`width of the combo box`	If set, defines the width of the combo box.
`advanced selector available?`	If set, defines if the advanced selector should be displayed when the underlying node is a foreign key.
`view name to use with the advanced selector`	If set, defines the name of the published view that will be used in the combo-box selection of the associated foreign key field.
`specific nomenclature`	If set, overrides the model-driven nomenclature.

Comparator

Compares two integers.



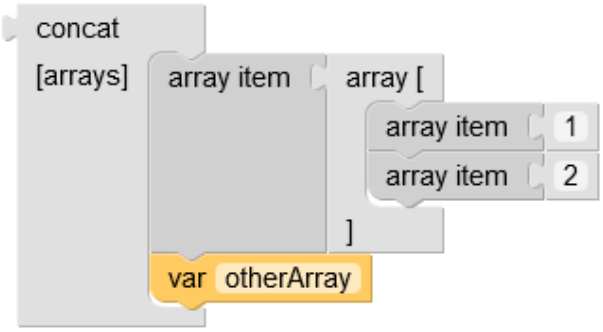
Comparison

Compares strings, numbers or booleans.



Concatenate arrays

Returns the concatenation of the given arrays.

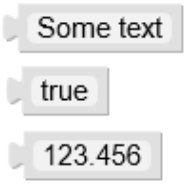


Parameters

Input	Description
`arrays`	A list of arrays.

Constant

A constant value. Depending on the context, this value may be interpreted as text, boolean or number.



Parameters

Input	Description
`value`	Text, boolean or number.

Content title

Display the workspace content title.

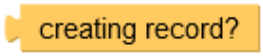


Parameters

Input	Description
`label`	The text to display.
`horizontal alignment`	The horizontal alignment of the element. If set, accepted values are: 'start', 'end', 'center'.

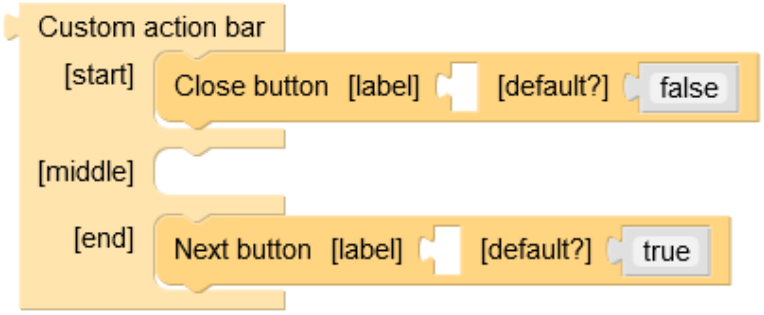
Creating record?

Returns a boolean indicating if the form is displayed in the context of a record creation or duplication.



Custom action bar

A custom action bar.

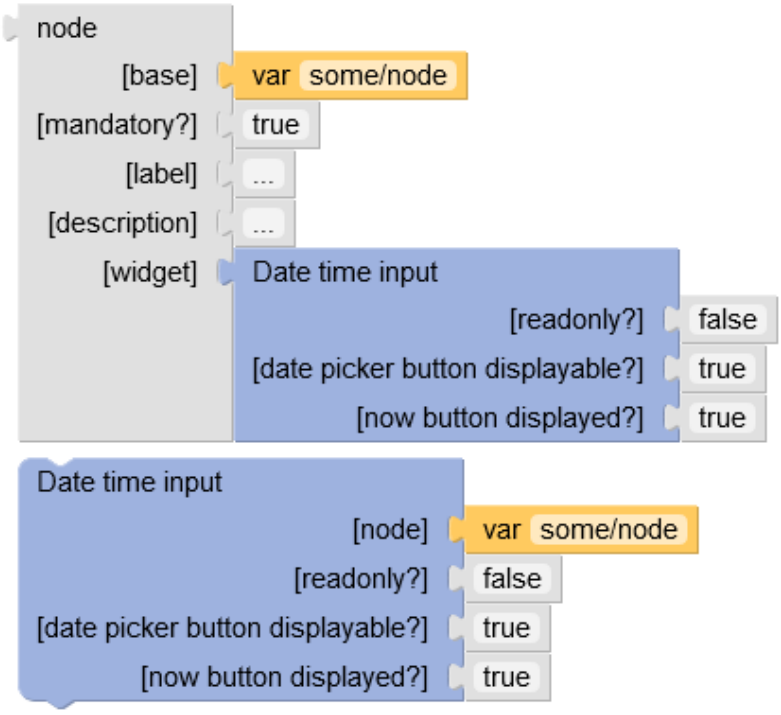


Parameters

Input	Description
`start`	A list of 'Button's to display on the left of the action bar.
`middle`	A list of 'Button's to display on the center of the action bar.
`end`	A list of 'Button's to display on the right of the action bar.

Date time input

Displays the date/time input.



Parameters

Input	Description
`node`	The node for which to display the widget.
`readonly?`	Indicates if the widget should be in readonly mode.
`date picker button displayable?`	Indicates if the button opening the date picker is displayed (read/write mode only).
`now button displayed?`	Indicates if the button setting the date to the current time is displayed (read/write mode only).

Declare

The declaration of a variable, in a 'with' block.



Parameters

Input	Description
`name`	The name of the variable. Must be unique for a given 'with' block.
`content`	The value of the variable, which will be returned by 'var' blocks referencing this.

See also *With* *[p 1010]*

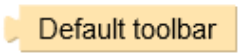
Default action bar

The default action bar.



Default toolbar

The model-driven toolbar.



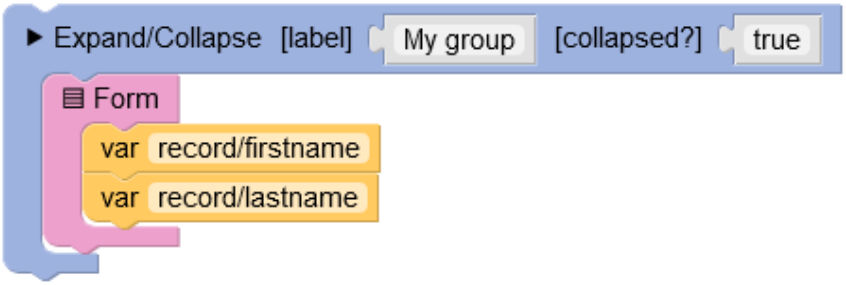
Duplicating record?

Returns a boolean indicating if the form is displayed in the context of a record duplication.



Expand/Collapse

Displays its content inside an expand/collapse block.

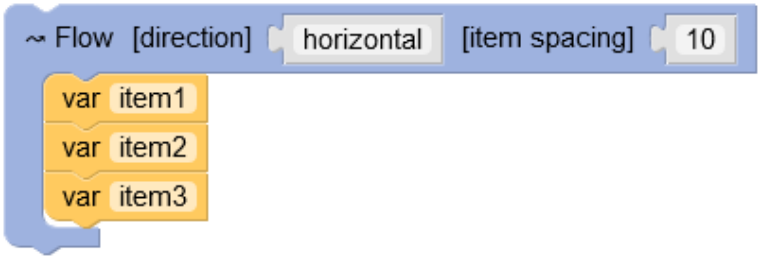


Parameters

Input	Description
`label`	Label displayed next to the arrow.
`collapsed?`	Indicates if the group is initially collapsed or not.
`content`	A list of components to be collapsible.

Flow

Displays its content in a fluid manner. If the elements don't fit in one row or column, they will wrap to start a new one.

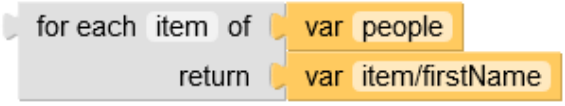


Parameters

Input	Description
`direction`	'horizontal' or 'vertical', indicates in which direction to queue its content.
`item spacing`	The space between elements, in pixels.
`content`	A list of elements to display either horizontally or vertically.

For each

Returns an array that is made of the result of the 'body' function, applied on each item of the 'of' array.

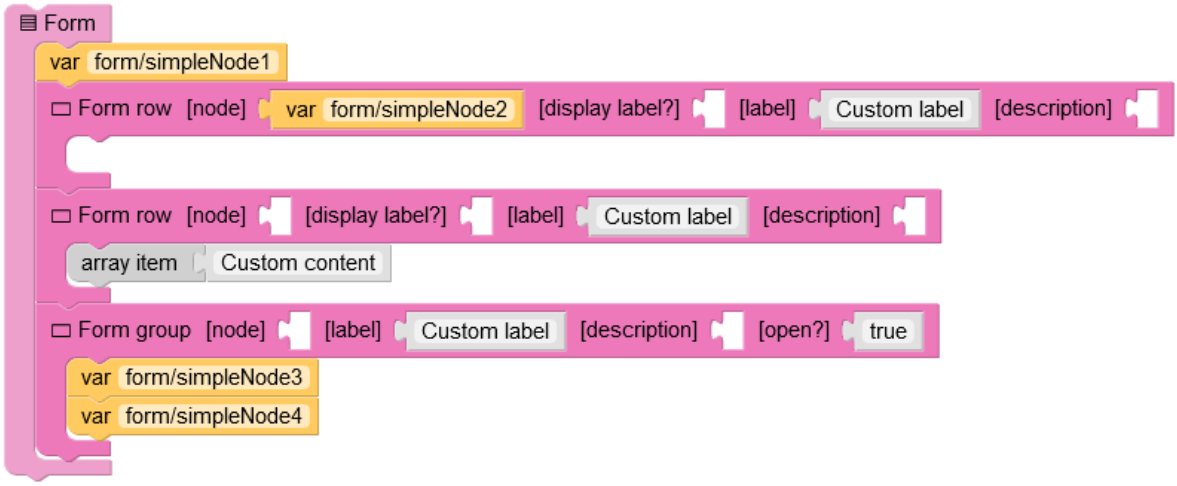


Parameters

Input	Description
`item`	The name of the variable containing the current item.
`array`	The array containing the items to transform.
`return`	What to return for the current item.

Form

Standard table-like layout with labels on the left side and values on the right side.



Parameters

Input	Description
`content`	A list of 'Form row', 'Form group' or nodes. Nodes will result in model driven display, 'Form row' and 'Form group' allow to display custom content.

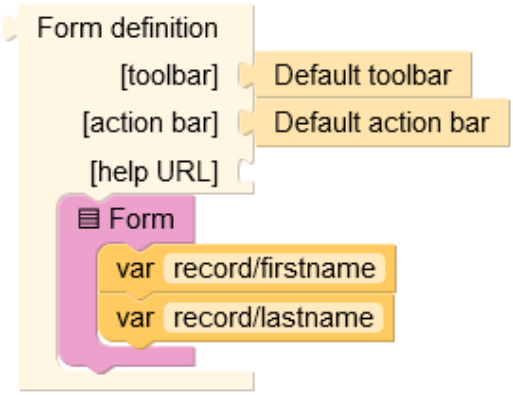
See also

Form row *[p 991]*

Form group *[p 990]*

Form definition

Actual definition of a form.

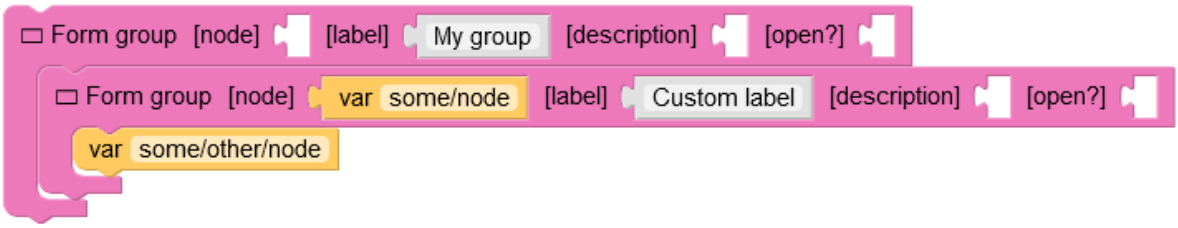


Parameters

Input	Description
`toolbar`	Definition of the toolbar displayed on top of the form.
`action bar`	Definition of the action bar displayed below the form.
`help URL`	URL of the help page.
`content`	The content of the layout.

Form group

Inside a 'Form', displays a collapsible group of items.



Parameters

Input	Description
`node`	If set, the group label, description and content will be model driven.
`label`	The label of the group. May override the model driven label if a node and a label are set.
`description`	The description of the group. May override the model driven description if a node and a description are set.
`open?`	Indicates if the group is initially open.
`content`	A list of 'Form row', 'Form group' or nodes. Nodes will result in model driven display, 'Form row' and 'Form group' allow to display custom content.

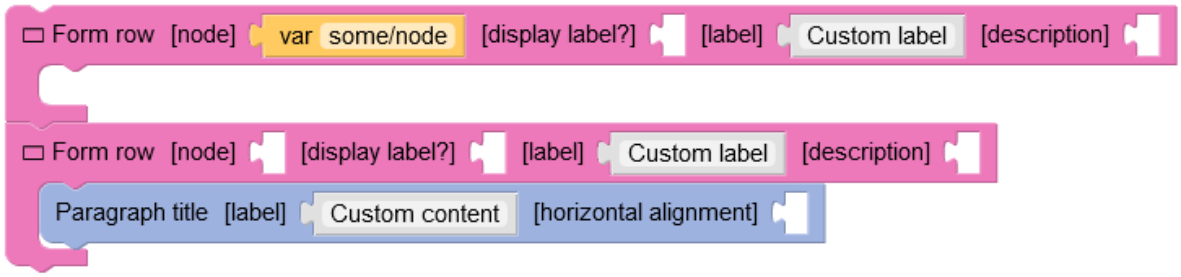
See also

Form *[p 989]*

Form row *[p 991]*

Form row

Inside a 'Form' or a 'Form group', displays a row, that is basically a label and a value.



Parameters

Input	Description
`node`	If set, the label, description and content will be model driven.
`display label?`	Indicates if the label should be displayed. If set to false in a 'Form' the content will span over both the label and content area.
`label`	The label of the row. May override the model driven label if a node and a label are set.
`description`	The description of the row. May override the model driven label if a node and a label are set.
`content`	The content to be displayed next to the label.

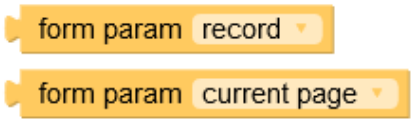
See also

Form *[p 989]*

Form group *[p 990]*

Form parameter

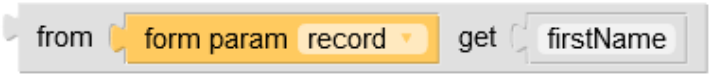
The 'record' is the node representing the displayed record. 'current page' is a number (1 by default) representing the displayed page. This value can be changed by the 'Previous' and 'Next' buttons.



From...get

Extract a value from an object or a node.

<from "node" get "path/to/value"> is equivalent to <var "node/path/to/value">

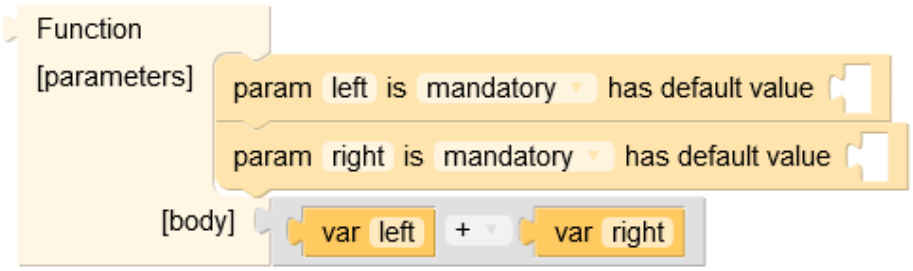


Parameters

Input	Description
`from`	Base object or node, from which to extract data.
`get`	Path of the data to extract, inside the object in 'from'.

Function

The definition of an anonymous function. A function makes use of its parameters in its body to return a result.



Parameters

Input	Description
`parameters`	List of 'param' blocks, defining the parameters accepted by this function.
`body`	Body describing the result of the function, using its parameters.

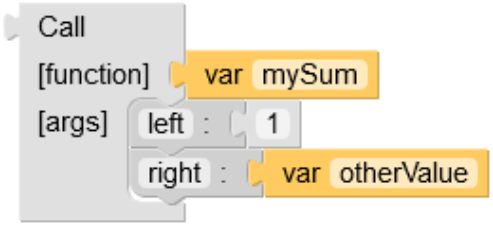
See also

Function parameter *[p 994]*

Function call *[p 993]*

Function call

Calls a function by setting actual values to its parameters.



Parameters

Input	Description
`function`	The function to call.
`args`	The arguments to pass to the function.

See also

Function call argument [p 994]

Function [p 993]

Function call argument

Actual value to pass to the parameter of the called function.



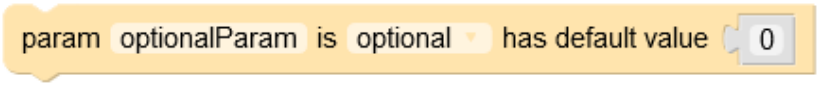
Parameters

Input	Description
`name`	The name of the parameter.
`value`	The actual value to pass the the parameter.

See also *Function call* [p 993]

Function parameter

This is the declaration of a parameter of a function.



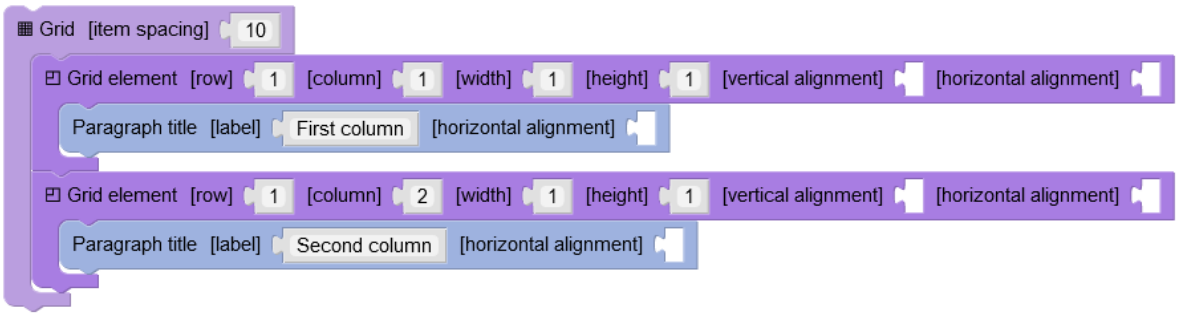
Parameters

Input	Description
`name`	The name of the parameter.
`mandatory?`	Indicates if this parameter must be provided by the caller.
`default value`	If set, this is the default value taken by this parameter if the caller does not provide one.

See also *Function* [p 993]

Grid

A layout allowing to place components in a grid with coordinates.



Parameters

Input	Description
`item spacing`	Space between elements, in pixels.
`content`	A list of 'Grid element's, which specify where to display their content.

See also *Grid element* *[p 995]*

Grid element

Inside a 'Grid', specifies the location of its content.

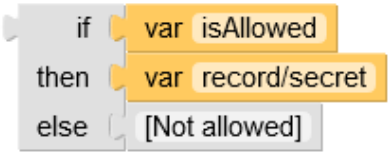
Parameters

Input	Description
`row`	The row where the element starts. Minimum value is 1.
`column`	The column where the element starts. Minimum value is 1.
`width`	The width of the element, in grid cells. Minimum value is 1.
`height`	The height of the element, in grid cells. Minimum value is 1.
`vertical alignment`	The vertical alignment of the element. If set, accepted values are: 'start', 'end', 'center'.
`horizontal alignment`	The horizontal alignment of the element. If set, accepted values are: 'start', 'end', 'center'.
`content`	The content inside the cell.

See also *Grid* *[p 995]*

If

A block returning either the content of 'then', or the content 'else', based on the result of the test.



Parameters

Input	Description
`test`	A test determining which value to return. This must be a boolean value.
`then`	The value returned if the test is true.
`else`	The value returned if the test is false.

Input parameter

Returns the raw value of an input parameter.



Parameters

Input	Description
`name`	The parameter name.

In role?

Returns a boolean indicating if the current user has the requested role.



Label

Displays the label of the given node.



Parameters

Input	Description
`node`	The node for which to display the label.

Localized

Block which result depends on the user locale.



Parameters

Input	Description
`[locale]`	Each parameter relates to a locale. The result of the block will be the one corresponding to the locale of the current user.

Message

Formats the given text by replacing java-like {0}, {1}, etc. placeholders by the argument at the given index.



Parameters

Input	Description
`pattern`	Text with placeholders ({0}, {1}, etc.) which will be replaced by the given arguments.
`arguments`	The list of values which will replace the placeholders in the pattern.

Next button

A button which will increment the 'current page' value by one.

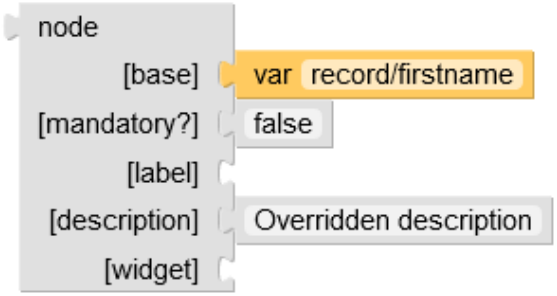


Parameters

Input	Description
`label`	If set, overrides the default label.
`default?`	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Node

Overrides the definition of an existing node.

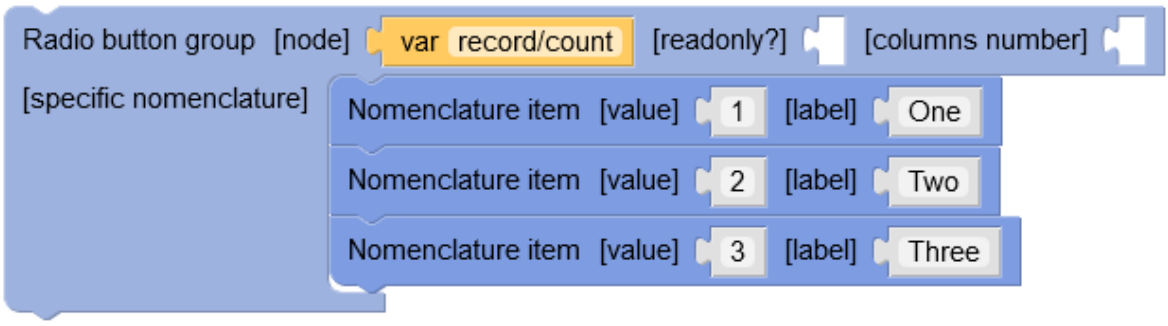


Parameters

Input	Description
`base`	The node to override.
`mandatory?`	If set, overrides the mandatory indicator of the base node.
`label`	If set, overrides the label of the node.
`description`	If set, overrides the label of the node.
`widget`	If set, overrides the default widget of the node.

Nomenclature item

A nomenclature item is a (key, label) pair.



Parameters

Input	Description
`value`	The value of the item.
`label`	The label of the item.

Not

Returns the inverse of the given boolean.



Object

An object composed of a list of (key, value) pairs.



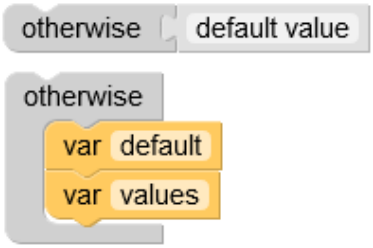
Parameters

Input	Description
`content`	A list of 'property' blocks.

See also *Property* *[p 1001]*

Otherwise

Final choice in a 'choose' block. Will be returned if no 'when' case matched.



Parameters

Input	Description
`content`	The value to return if no 'when' case matched.

See also *Choose* *[p 980]*

Paragraph title

Display a paragraph title.

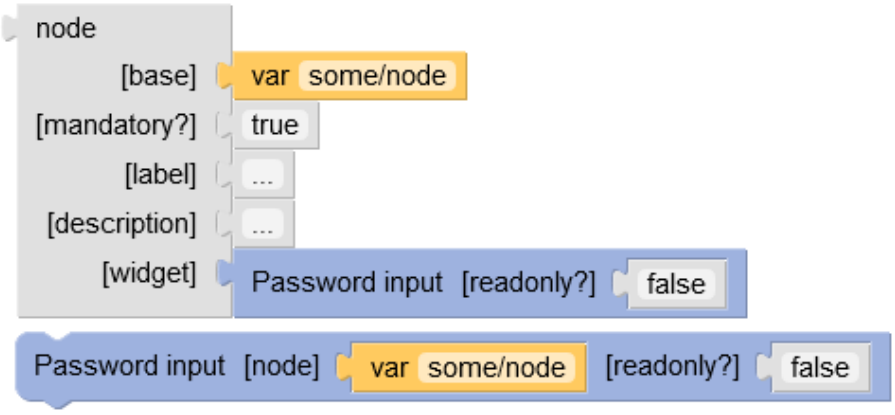


Parameters

Input	Description
`label`	The text to display.
`horizontal alignment`	The horizontal alignment of the element. If set, accepted values are: 'start', 'end', 'center'.

Password

Displays the input password widget.



Parameters

Input	Description
`node`	The node for which to display the widget.
`readonly?`	Indicates if the widget should be in readonly mode.

Previous button

A button which will decrement the 'current page' value by one.



Parameters

Input	Description
`label`	If set, overrides the default label.
`default?`	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Property

A property of an object.

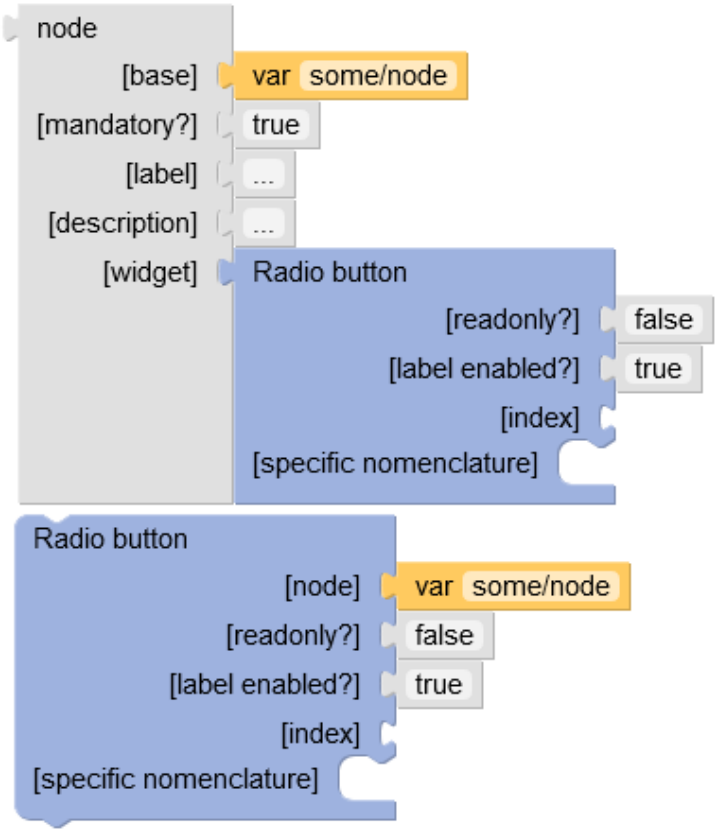


Parameters

Input	Description
`name`	Name of the property. Must be unique for a given object.
`value`	Value of the property.

Radio button

Displays a radio button widget.

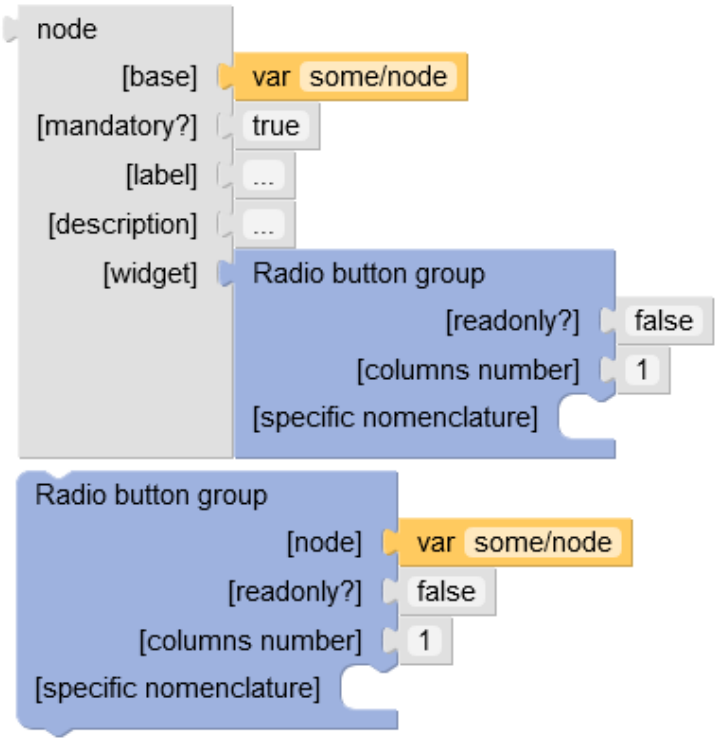


Parameters

Input	Description
`node`	The node for which to display the widget.
`readonly?`	Indicates if the widget should be in readonly mode.
`label enabled?`	If set, indicates if the item label is to be added next to the widget.
`index`	The index for this enumeration item.
`specific nomenclature`	If set, overrides the model-driven nomenclature.

Radio button group

Displays the radio button group widget.



Parameters

Input	Description
`node`	The node for which to display the widget.
`readonly?`	Indicates if the widget should be in readonly mode.
`columns number`	If set, defines the number of columns to use for the layout of the radio buttons.
`specific nomenclature`	If set, overrides the model-driven nomenclature.

Revert button

Standard 'Revert' button.



Parameters

Input	Description
`label`	If set, overrides the default label.
`default?`	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Save button

Standard 'Save' button.

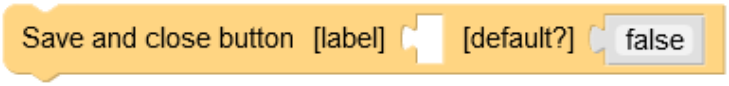


Parameters

Input	Description
`label`	If set, overrides the default label.
`default?`	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Save and close button

Standard 'Save and close' button.

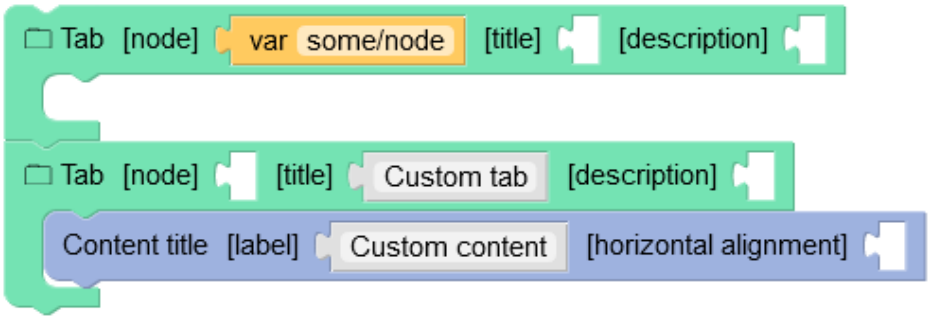


Parameters

Input	Description
`label`	If set, overrides the default label.
`default?`	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Tab

A tab, inside a 'Tab container'.



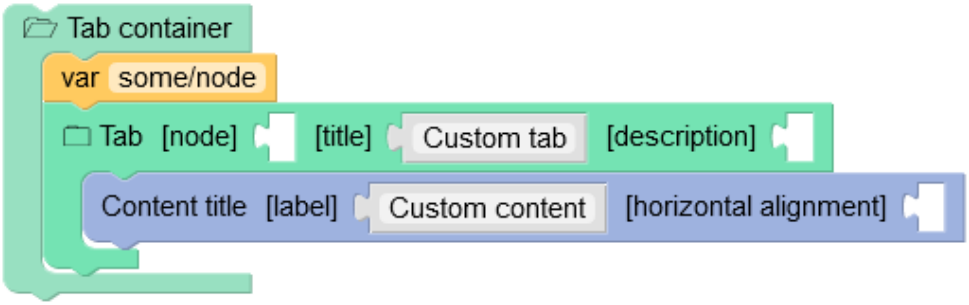
Parameters

Input	Description
`node`	If present, the tab title, description and content will be model driven.
`title`	The title of the tab. May override the model driven title if a node and a title are set.
`description`	The description of the tab. May override the model driven description if a node and a description are set.
`content`	The content of the tab. May override the model driven content if a node and a content are set.

See also *Tab container* *[p 1006]*

Tab container

A container of tabs, which displays a list of tab names and one active tab.



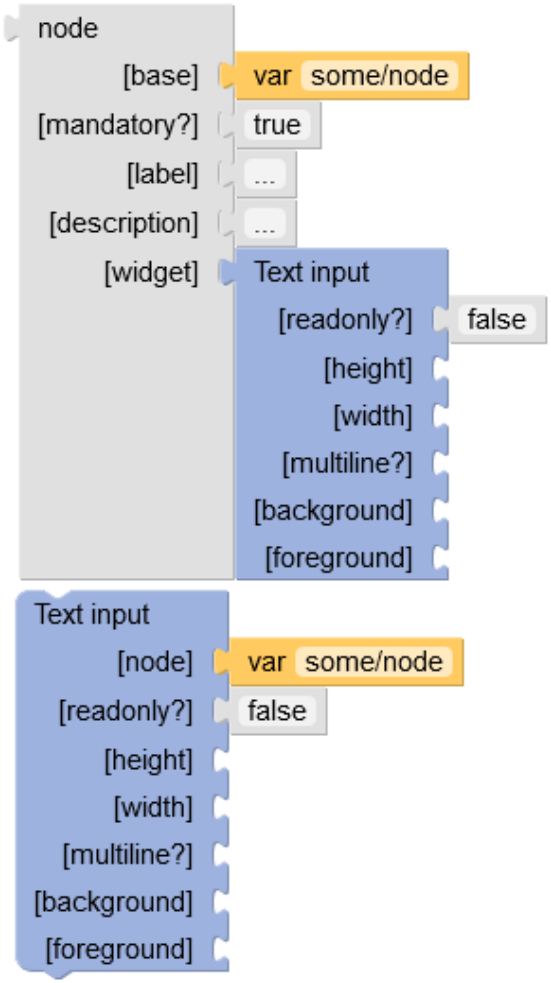
Parameters

Input	Description
`content`	A list of 'Tab's.

See also *Tab* *[p 1005]*

Text input

Displays the text input widget.



Parameters

Input	Description
`node`	The node for which to display the widget.
`readonly?`	Indicates if the widget should be in readonly mode.
`height`	If set, defines the height of the widget, in pixels.
`width`	If set, defines the width of the widget, in pixels.
`multiline?`	If set, defines if the widget spans over multiple lines.
`background`	If set, defines the background color, in hexadecimal format.
`foreground`	If set, defines the foreground color, in hexadecimal format.

Toolbar named

The toolbar having the specified name.

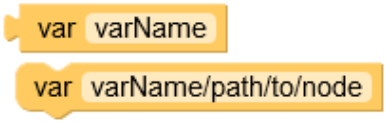


Parameters

Input	Description
`name`	The name of the toolbar to display.

Variable

A reference to a variable declared by a 'with...declare' block. A path is also accepted if the variable is an object or a node.

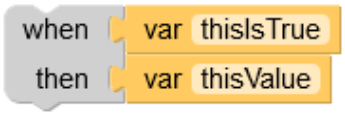


Parameters

Input	Description
`expression`	A reference to a variable declared by a 'with...declare' block. A path is also accepted if the variable is an object or a node.

When

A choice in a 'choose' block. If the test is true, the 'then' content will be returned, otherwise the next block will be tested.



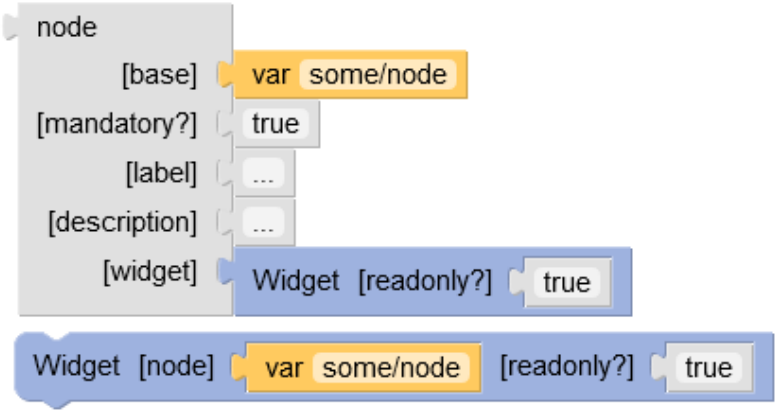
Parameters

Input	Description
`when`	A boolean test indicating if this case should be resolved.
`then`	If the test is true, this value will be returned.

See also *Choose* *[p 980]*

Widget

Displays the default widget.

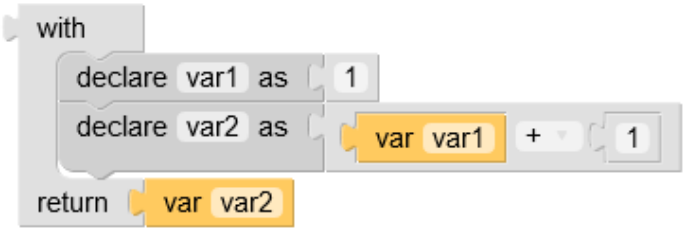


Parameters

Input	Description
`node`	The node for which to display the widget.
`readonly?`	Indicates if the widget should be in readonly mode.

With

Declares variables that can be used in its 'return' statement.



Parameters

Input	Description
`variables`	A list of 'declare' blocks.
`return`	The value to return. The variables declared in the 'variables' statement can be used here, with 'var' blocks.

See also *Declare* *[p 986]*

CHAPTER 162

Publication REST API

This chapter details the REST APIs used to publish a data model.

This chapter contains the following topics:

1. Publish DMA data model

162.1 Publish DMA data model

This service is designed to publish a data model defined in the Data Model Assistant (DMA). It allows you to:

- List all data models defined in the DMA.
- Retrieve the information of a data model.
- Publish an existing data model according to its configuration defined in the DMA.
- Publish an existing data model to a specific XML Schema Document overriding the configuration defined in the DMA.
- List all versions of a data model.
- Retrieve a specific version of a data model.
- Retrieve all the embedded publications of a data model.

The complete description of the services is available using the following URL to access the OpenAPI specification:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/datamodel/v1/dma/
```

CHAPTER 163

Workflow model

The workflow offers three types of steps: 'library' or 'specific' and 'script'.

'Library' is a bean defined in `module.xml` and is reusable. Using the 'library' bean improves the ergonomics: parameters are dynamically displayed in the definition screens.

A 'specific' object is a bean defined only by its class name. In this case, the display is not dynamic.

A 'script' is a procedural language based on DSL (Domain Specific Language) used to define custom tasks. The EBX script IDE is used to create and edit the script. The parameters of the workflow context are dynamically displayed in the definition screens.

This chapter contains the following topics:

1. Bean categories
2. Sample of ScriptTask
3. Sample of ScriptTaskBean
4. Samples of UserTask
5. Samples of Condition
6. Sample of ConditionBean
7. Sample of SubWorkflowsInvocationBean
8. Sample of WaitTaskBean
9. Sample of ActionPermissionsOnWorkflow
10. Sample of WorkflowTriggerBean
11. Sample of trigger starting a process instance

163.1 Bean categories

Step	**Library**	Specific
Scripts	ScriptTaskBean	ScriptTask
Conditions	ConditionBean	Condition
User task	UserTask	

163.2 Sample of ScriptTask

Java Code

A script task has to override the method `execute` as in the following example:

```
public class NppScriptTask_CreateWorkingBranch extends ScriptTask
{
    public void executeScript(ScriptTaskContext aContext) throws OperationException
    {
        Repository repository = aContext.getRepository();
        String initialBranchString = aContext.getVariableString("initialBranch");
        AdaptationHome initialBranch = repository.lookupHome(HomeKey.forBranchName(initialBranchString));
        if (initialBranch == null)
            throw OperationException.createError("Null value for initialBranch");

        HomeCreationSpec spec = new HomeCreationSpec();
        spec.setParent(initialBranch);
        spec.setKey(HomeKey.forBranchName("Name"));
        spec.setOwner(Profile.EVERYONE);
        spec.setHomeToCopyPermissionsFrom(initialBranch);
        AdaptationHome newHome = repository.createHome(spec, aContext.getSession());
                //feeds dataContext
        aContext.setVariableString("workingBranch", newHome.getKey().getName());
    }
}
```

See also ***com.orchestranetworks.workflow.ScriptTask*** *`ScriptTask`*[API]

163.3 Sample of ScriptTaskBean

Java Code

A script task bean has to override the method `executeScript` as in the following example:

```
public class ScriptTaskBean_CreateBranch extends ScriptTaskBean
{
    private String initialBranchName;

    private String newBranch;

    public String getInitialBranchName()
    {
        return this.initialBranchName;
    }

    public void setInitialBranchName(String initialBranchName)
    {
        this.initialBranchName = initialBranchName;
    }

    public String getNewBranch()
    {
        return this.newBranch;
    }

    public void setNewBranch(String newBranch)
    {
        this.newBranch = newBranch;
    }

    public void executeScript(ScriptTaskBeanContext aContext) throws OperationException
    {
        final Repository repository = aContext.getRepository();

        String initialBranchName = this.getInitialBranchName();
        final AdaptationHome initialBranch = repository.lookupHome(HomeKey.forBranchName(initialBranchName));
        final HomeCreationSpec spec = new HomeCreationSpec();
        spec.setParent(initialBranch);
        spec.setKey(HomeKey.forBranchName(XsFormats.SINGLETON.formatDateTime(new Date())));
        spec.setOwner(Profile.EVERYONE);
        spec.setHomeToCopyPermissionsFrom(initialBranch);
        final AdaptationHome branchCreate = repository.createHome(spec, aContext.getSession());
```

```
        this.setNewBranch(branchCreate.getKey().getName());
    }
}
```

See also ***com.orchestranetworks.workflow.ScriptTaskBean*** `ScriptTaskBean`[API]

Configuration through module.xml

A script task bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.genericScriptTask.ScriptTaskBean_CreateBranch">
            <documentation xml:lang="fr-FR">
                <label>Créer une branche</label>
                <description>
                    Ce script permet de créer une branche
                </description>
            </documentation>
            <documentation xml:lang="en-US">
                <label>Create a branch</label>
                <description>
                    This script creates a branch
                </description>
            </documentation>
            <properties>
                <property name="initialBranchName" input="true">
                    <documentation xml:lang="fr-FR">
                        <label>Branche initiale</label>
                        <description>
                            Nom de la branche initiale.
                        </description>
                    </documentation>
                    <documentation xml:lang="en-US">
                        <label>Initial branch</label>
                        <description>
                            Initial branch name.
                        </description>
                    </documentation>
                </property>
                <property name="newBranch" output="true">
                    <documentation xml:lang="fr-FR">
                        <label>Nouvelle branche</label>
                        <description>
                            Nom de la branche créée
                        </description>
                    </documentation>
                    <documentation xml:lang="en-US">
                        <label>New branch</label>
                        <description>
                            Created branch name.
                        </description>
                    </documentation>
                </property>
            </properties>
        </bean>
    </beans>
</module>
```

163.4 Samples of UserTask

Service declaration via module.xml

A built-in service can be declared in `module.xml` to be used in the user task definition.

```
 <services>
  <service name="ServiceModule">
     <resourcePath>/service.jsp</resourcePath>
     <type>branch</type>
        <documentation xml:lang="fr-FR">
            <label>Workflow service</label>
               <description>
                Ce service permet de ...
               </description>
  </documentation>
           <documentation xml:lang="en-US">
```

```
            <label>Service workflow</label>
               <description>
                The purpose of this service is ...
               </description>
  </documentation>
          <properties>
          <property name="param1" input="true">
               <documentation xml:lang="fr-FR">
                   <label>Param1</label>
                   <description>Param1 ...</description>
               </documentation>
          </property>
          <property name="param2" output="true">
          </property>
       </properties>
 </service>
    <serviceLink serviceName="adaptationService">
        <importFromSchema>
            /WEB-INF/ebx/schema/schema.xsd
        </importFromSchema>
    </serviceLink>
</services>
```

A more complex UserTask

The GUI is quite similar as the example above. The field 'Rule' must be filled to define the class extending the 'UserTask' to invoke.

```
public class NppUserTask_ValidateProduct extends UserTask
{
    public void handleWorkItemCompletion(UserTaskWorkItemCompletionContext context)
        throws OperationException
    {
        if (context.getCompletedWorkItem().isRejected())
        {
            context.setVariableString(NppConstants.VAR_VALIDATION, "KO");
            context.completeUserTask();
        }
        else if (context.checkAllWorkItemMatchStrategy())
        {
            context.setVariableString(NppConstants.VAR_VALIDATION, "OK");
            context.completeUserTask();
        }
    }

    public void handleCreate(UserTaskCreationContext context) throws OperationException
    {
        CreationWorkItemSpec spec = CreationWorkItemSpec.forOfferring(NppConstants.ROLE_PVALIDATOR);
        spec.setNotificationMail("1");
        context.createWorkItem(spec);
        context.setVariableString(NppConstants.VAR_VALIDATION, "validating");
    }
}
```

See also ***com.orchestranetworks.workflow.UserTask*** `UserTask`[API]

163.5 Samples of Condition

Java Code

The method `evaluate` has to be overridden:

```
public class NppCondition_IsValidationOK extends Condition
{
    public boolean evaluateCondition(ConditionContext context) throws OperationException
    {
        String validation = context.getVariableString("validationResult");
        boolean hasError = "KO".equals(validation);
        return !hasError;
    }
}
```

See also ***com.orchestranetworks.workflow.Condition*** `Condition`[API]

163.6 Sample of ConditionBean

Java Code

The method `evaluateCondition` has to be overridden as in the following sample:

```
public class ConditionBean_IsBranchValid extends ConditionBean
{
    private String branchName;

    public String getBranchName()
    {
        return this.branchName;
    }

    public void setBranchName(String branchName)
    {
        this.branchName = branchName;
    }

    public boolean evaluateCondition(ConditionBeanContext aContext) throws OperationException
    {
        final Repository repository = aContext.getRepository();
        Severity severityForValidation = Severity.ERROR;
        String branchToTestName = this.getBranchName();
        final AdaptationHome branchToTest = repository.lookupHome(HomeKey.forBranchName(branchToTestName));
        if (branchToTest.getValidationReportsMap(severityForValidation) != null
            && branchToTest.getValidationReportsMap(severityForValidation).size() > 0)
        {
            return false;
        }
        return true;
    }
}
```

See also ***com.orchestranetworks.workflow.ConditionBean*** `ConditionBean`[API]

Configuration through module.xml

The condition bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.genericScriptTask.ConditionBean_IsBranchValid">
            <documentation xml:lang="fr-FR">
                <label>Branche valide ?</label>
                <description>
                    Ce script permet de tester si une branche est valide.
                </description>
            </documentation>
            <documentation xml:lang="en-US">
                <label>Branch valid ?</label>
                <description>
                    This script allows to check if a branch is valid.
                </description>
            </documentation>
            <properties>
                <property name="branchName" input="true">
                    <documentation xml:lang="fr-FR">
                        <label>Branche à contrôler</label>
                        <description>
                            Nom de la branche à valider.
                        </description>
                    </documentation>
                    <documentation xml:lang="en-US">
                        <label>Branch to check</label>
                        <description>
                            Branch name to check.
                        </description>
                    </documentation>
                </property>
            </properties>
        </bean>
```

```
    </beans>
</module>
```

163.7 Sample of SubWorkflowsInvocationBean

Java Code

```
public class MySubWorkflowsInvocationBean extends SubWorkflowsInvocationBean
{
 @Override
 public void handleCreateSubWorkflows(SubWorkflowsCreationContext aContext)
  throws OperationException
 {
  final ProcessLauncher subWorkflow1 = aContext.registerSubWorkflow(
   AdaptationName.forName("validateProduct"),
   "validateProduct1");
  subWorkflow1.setLabel(UserMessage.createInfo("Validate the new product by marketing"));
  subWorkflow1.setInputParameter("workingBranch", aContext.getVariableString("workingBranch"));
  subWorkflow1.setInputParameter("code", aContext.getVariableString("code"));
  subWorkflow1.setInputParameter("service", aContext.getVariableString("marketing"));

  final ProcessLauncher subWorkflow2 = aContext.registerSubWorkflow(
   AdaptationName.forName("validateProduct"),
   "validateProduct2");
  subWorkflow2.setLabel(UserMessage.createInfo("Validate the new product by direction"));
  subWorkflow2.setInputParameter("workingBranch", aContext.getVariableString("workingBranch"));
  subWorkflow2.setInputParameter("code", aContext.getVariableString("code"));
  subWorkflow2.setInputParameter("service", aContext.getVariableString("direction"));

  // Conditional launching.
  if (aContext.getVariableString("productType").equals("book"))
  {
   final ProcessLauncher subWorkflow3 = aContext.registerSubWorkflow(
    AdaptationName.forName("generateISBN"),
    "generateISBN");
   subWorkflow3.setLabel(UserMessage.createInfo("Generate ISBN"));
   subWorkflow3.setInputParameter(
    "workingBranch",
    aContext.getVariableString("workingBranch"));
   subWorkflow3.setInputParameter("code", aContext.getVariableString("code"));
  }

  aContext.launchSubWorkflows();
 }
 @Override
 public void handleCompleteAllSubWorkflows(SubWorkflowsCompletionContext aContext)
  throws OperationException
 {
  aContext.getCompletedSubWorkflows();
  final ProcessInstance validateProductMarketing = aContext.getCompletedSubWorkflow("validateProduct1");
  final ProcessInstance validateProductDirection = aContext.getCompletedSubWorkflow("validateProduct2");
  if (aContext.getVariableString("productType").equals("book"))
  {
   final ProcessInstance generateISBN = aContext.getCompletedSubWorkflow("generateISBN");
   aContext.setVariableString("isbn", generateISBN.getDataContext().getVariableString(
    "newCode"));
  }

  if (validateProductMarketing.getDataContext().getVariableString("Accepted").equals("true")
   && validateProductDirection.getDataContext().getVariableString("Accepted").equals(
    "true"))
   aContext.setVariableString("validation", "ok");
 }
}
```

See also ***com.orchestranetworks.workflow.SubWorkflowsInvocationBean*** `SubWorkflowsInvocationBean`[API]

Configuration through module.xml

`SubWorkflowsInvocationBean` bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.test.MySubWorkflowsInvocationBean"/>
    </beans>
```

```
</module>
```

163.8 Sample of WaitTaskBean

Java Code

```
public class MyWaitTaskBean extends WaitTaskBean
{
 @Override
 public void onStart(WaitTaskOnStartContext aContext)
 {
  Map<String, String> params = new HashMap<String, String>();
  params.put("resumeId", aContext.getResumeId());
  myMethod.callWebService(params);
 }

 @Override
 public void onResume(WaitTaskOnResumeContext aContext) throws OperationException
 {
  // Defines a specific mapping.
  aContext.setVariableString("code", aContext.getOutputParameters().get("isbn"));
  aContext.setVariableString("comment", aContext.getOutputParameters().get("isbnComment"));
 }
}
```

See also ***com.orchestranetworks.workflow.WaitTaskBean*** *`WaitTaskBean`*[API]

Configuration through module.xml

`WaitTaskBean` bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.test.MyWaitTaskBean"/>
    </beans>
</module>
```

163.9 Sample of ActionPermissionsOnWorkflow

Java Code

```
package com.orchestranetworks.workflow.test;

import com.orchestranetworks.service.*;
import com.orchestranetworks.workflow.*;
import com.orchestranetworks.workflow.ProcessExecutionContext.*;

/**
 */
public class MyDynamicPermissions extends ActionPermissionsOnWorkflow
{

    public ActionPermission getActionPermission(
        WorkflowPermission aWorkflowAction,
        ActionPermissionsOnWorkflowContext aContext)
    {
        if (WorkflowPermission.VIEW.equals(aWorkflowAction)
            || WorkflowPermission.CREATE_PROCESS.equals(aWorkflowAction))
            return ActionPermission.getEnabled();
        return ActionPermission.getDisabled();
    }

}
```

See also ***com.orchestranetworks.workflow.ActionPermissionsOnWorkflow*** *`ActionPermissionsOnWorkflow`*[API]

Configuration through module.xml

`ActionPermissionsOnWorkflow` bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.test.MyDynamicPermissions"/>
    </beans>
</module>
```

163.10 **Sample of WorkflowTriggerBean**

Java Code

```
public class MyWorkflowTriggerBean extends WorkflowTriggerBean
{
 @Override
 public void handleAfterProcessInstanceStart(
  WorkflowTriggerAfterProcessInstanceStartContext aContext) throws OperationException
 {
  final DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());
  final MailSpec spec = aContext.createMailSpec();
  spec.notify(NotificationType.TO, "supervisor@mail.com");

  spec.setSubject("[TRIGGER] After process instance start");
  spec.setBody("The workflow '"
   + policy.formatUserMessage(aContext.getProcessInstance().getLabel())
   + "' has been created.");

  spec.sendMail(Locale.US);
 }

 @Override
 public void handleBeforeProcessInstanceTermination(
  WorkflowTriggerBeforeProcessInstanceTerminationContext aContext) throws OperationException
 {
  final DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

  final MailSpec spec = aContext.createMailSpec();
  spec.notify(NotificationType.TO, "supervisor@mail.com");

  spec.setSubject("[TRIGGER] Before process instance termination");
  spec.setBody("The workflow '"
   + policy.formatUserMessage(aContext.getProcessInstance().getLabel())
   + "' has been completed. The created product is: '"
   + aContext.getVariableString(NppConstants.VAR_CODE) + "'.");

  spec.sendMail(Locale.US);
 }

 @Override
 public void handleAfterWorkItemCreation(WorkflowTriggerAfterWorkItemCreationContext aContext)
  throws OperationException
 {
  DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

  MailSpec spec = aContext.createMailSpec();
  spec.notify(NotificationType.TO, "supervisor@mail.com");

  spec.setSubject("[TRIGGER] After work item creation");
  WorkItem workItem = aContext.getWorkItem();
  State state = workItem.getState();
  String body = "The work item '" + policy.formatUserMessage(workItem.getLabel())
   + "' has been created. \n The step id is : " + aContext.getCurrentStepId()
   + ". \n The work item is in state : " + policy.formatUserMessage(state.getLabel());

  if (workItem.getOfferedTo() != null)
   body += "\n The role is :" + workItem.getOfferedTo().format();
  if (workItem.getUserReference() != null)
   body += "\n The user is :" + workItem.getUserReference().format();

  spec.setBody(body);

  spec.sendMail(Locale.US);
 }

 @Override
```

```
public void handleBeforeWorkItemStart(WorkflowTriggerBeforeWorkItemStartContext aContext)
 throws OperationException
{
 DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

 MailSpec spec = aContext.createMailSpec();
 spec.notify(NotificationType.TO, "supervisor@mail.com");

 spec.setSubject("[TRIGGER] Before work item start");
 spec.setBody("The work item '"
  + policy.formatUserMessage(aContext.getWorkItem().getLabel())
  + "' has been started. \n  The current step id is : "
  + aContext.getCurrentStepId()
  + ". \n The work item user is: '"
  + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
   aContext.getWorkItem().getUserReference(),
   aContext.getSession().getLocale()) + "'.");

 spec.sendMail(Locale.US);
}

@Override
public void handleBeforeWorkItemAllocation(
 WorkflowTriggerBeforeWorkItemAllocationContext aContext) throws OperationException
{
 DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

 MailSpec spec = aContext.createMailSpec();
 spec.notify(NotificationType.TO, "supervisor@mail.com");

 spec.setSubject("[TRIGGER] Before work item allocation");
 spec.setBody("The work item '"
  + policy.formatUserMessage(aContext.getWorkItem().getLabel())
  + "' has been allocated. \n  The current step id is: "
  + aContext.getCurrentStepId()
  + ". \n  The work item user is: '"
  + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
   aContext.getUserReference(),
   aContext.getSession().getLocale()) + "'.");

 spec.sendMail(Locale.US);
}

@Override
public void handleBeforeWorkItemDeallocation(
 WorkflowTriggerBeforeWorkItemDeallocationContext aContext) throws OperationException
{
 DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

 MailSpec spec = aContext.createMailSpec();
 spec.notify(NotificationType.TO, "supervisor@mail.com");

 spec.setSubject("[TRIGGER] Before work item deallocation");
 spec.setBody("The work item '"
  + policy.formatUserMessage(aContext.getWorkItem().getLabel())
  + "' has been deallocated. \n  The current step id is: "
  + aContext.getCurrentStepId()
  + ". \n  The old work item user is: '"
  + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
   aContext.getWorkItem().getUserReference(),
   aContext.getSession().getLocale()) + ".");

 spec.sendMail(Locale.US);

}

@Override
public void handleBeforeWorkItemReallocation(
 WorkflowTriggerBeforeWorkItemReallocationContext aContext) throws OperationException
{
 DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

 MailSpec spec = aContext.createMailSpec();
 spec.notify(NotificationType.TO, "supervisor@mail.com");

 spec.setSubject("[TRIGGER] Before work item reallocation");
 spec.setBody("The work item '"
  + policy.formatUserMessage(aContext.getWorkItem().getLabel())
  + "' has been reallocated. \n  The current step id is: "
  + aContext.getCurrentStepId()
  + ". \n  The work item user is: '"
  + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
   aContext.getUserReference(),
   aContext.getSession().getLocale())
  + "'. The old work item user is: '"
```

```
   + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
    aContext.getWorkItem().getUserReference(),
    aContext.getSession().getLocale()) + "'.");

  spec.sendMail(Locale.US);

 }
 @Override
 public void handleBeforeWorkItemTermination(
  WorkflowTriggerBeforeWorkItemTerminationContext aContext) throws OperationException
 {
  DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

  MailSpec spec = aContext.createMailSpec();
  spec.notify(NotificationType.TO, "supervisor@mail.com");

  spec.setSubject("[TRIGGER] Before work item termination");
  spec.setBody("The work item '"
   + policy.formatUserMessage(aContext.getWorkItem().getLabel())
   + "' has been terminated. \n  The current step id is: " + aContext.getCurrentStepId()
   + ". \n  The work item has been accepted ? " + aContext.isAccepted());

  spec.sendMail(Locale.US);
 }
}
```

See also ***com.orchestranetworks.workflow.WorkflowTriggerBean*** `WorkflowTriggerBean`[API]

Configuration through module.xml

`WorkflowTriggerBean` bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.test.MyWorkflowTriggerBean"/>
    </beans>
</module>
```

163.11 Sample of trigger starting a process instance

Sample

```
public class TriggerWorkflow extends TableTrigger
{
    public void handleAfterModify(AfterModifyOccurrenceContext aContext) throws OperationException
    {
        ValueContext currentRecord = aContext.getOccurrenceContext();
        String code = (String) currentRecord.getValue(Path.parse("/code"));

        //Get published process
        PublishedProcessKey processPublishedKey = PublishedProcessKey.forName("productProcess");

        //Defines process instance
        WorkflowEngine engine = WorkflowEngine.getFromProcedureContext(aContext.getProcedureContext());
        ProcessLauncher launcher = engine.getProcessLauncher(processPublishedKey);

        //initialize Data Context
        launcher.setInputParameter("code", "/root/Client[./code=\"" + code + "\"]");
        launcher.setInputParameter("workingBranch", aContext.getAdaptationHome().getKey().getName());

        //Starts process
        launcher.launchProcess();

    }
      //...
}
```

User interface

CHAPTER 164

Interface customization

The TIBCO EBX® graphical interface can be customized through various EBX® APIs.

This chapter contains the following topics:

1. Using EBX® as a Web Component
2. Adding user services
3. Customizing forms
4. Customizing widgets
5. Customizing table filter
6. Customizing record label
7. Including CSS and JavaScript

164.1 Using EBX® as a Web Component

EBX® can be integrated into any application that is accessible through a supported web browser [p 524], thanks to the Web Component API.

A typical use is to integrate EBX® views into an organization's intranet framework. Web Components can also be invoked from the EBX® user interface using User services [p 1024].

To embed all or part of EBX® in a web page, the HTML tag `<iframe>` should be used by indicating the URL to EBX®. This URL can be specified either manually or by using the `UIHttpManagerComponent` API. A single web page may include several iframes that integrate EBX®. It is then possible to create a portal made of tables, forms, hierarchical views, etc., from EBX®.

See also *Using TIBCO EBX® as a Web Component [p 419]*

164.2 Adding user services

A user service is an extension of EBX® that provides a graphical user interface (GUI) that allows users to access specific or advanced functions.

Powerful custom user services can be developed using the same visual components and data validation mechanisms as standard EBX® user interfaces.

See also *User service overview [p 1027]*

164.3 Customizing forms

It is possible to override the default layout and behavior of forms in the user interface by using various tools and API.

Custom forms editor

It is possible to override the default layout of forms in the user interface by using the 'Custom forms' extension in the DMA. This extension provides a graphical editor, which gives the possibility to customize the layout of a form, while having the same components and standard behavior as record forms using the default layout.

See also *Custom forms [p 973]*

Programmatic form layout

It is also possible to use the `UIForm` API to override the default layout of forms. This API provides the standard input components from EBX®, which give the possibility to customize the layout of a form, while having the same components and standard behavior as record forms using the default layout.

See also

`UIForm`[API]

`UIFormPaneWriter`[API]

`UIWidget`[API]

`UIFormHeader`[API]

`UIFormBottomBar`[API]

User service as form layout

It is also possible to use user services to override the default layout of forms. This API gives the possibility to customize the layout of a form, while having the same components as record forms using the default layout, with a customizable behavior.

See also

`UserServiceRecordFormFactory`[API]

Overview [p 1027]

164.4 Customizing widgets

Custom widgets are included in the Java API to allow the development of user interface components for fields or groups of fields. A custom widget (`UICustomWidget`) allows, for a given node, to control the area where the input or display component is located. This allows having an input and display component that is fully customizable in HTML. The standard components (`UIWidgets`) are available and can be used. The custom widget can implement several display aspects: input component in the

form, display in the form, display in a table cell. If a custom widget writes its own HTML components, it has the possibility to save the value in the database when submitting the form.

See also *UICustomWidget*[API]

164.5 Customizing table filter

A table filter allows, for a given table, to create a criteria input form in order to apply a filter to the table view. The `UITableFilter` API is used to implement a table filter with a custom UI. It provides methods to create a UI that automatically adapts to the underlying data format (for example, by displaying a combo box when applicable).

See also

UITableFilter[API]

Properties of data model elements [p 61]

UILabelRendererForHierarchy[API]

164.6 Customizing record label

EBX® uses a label to display a reference to a given record (for example a foreign key). Labels are also used in the title of a record form and in hierarchical views. This label can be customized in the model using expressions. It is also possible to customize labels using the `UILabelRenderer` API.

See also *UILabelRenderer*[API]

164.7 Including CSS and JavaScript

It is possible to integrate CSS and JavaScript files in each EBX® page by declaring them in the registration module. The inclusion of JavaScript files can be subject to conditions through development depending on the context.

See also

Module registration [p 880]

Development recommendations [p 1055]

UIDependencyRegisterer[API]

CHAPTER 165

Overview

A user service is an extension to TIBCO EBX® that provides a graphical user interface (GUI) allowing users to access specific or advanced functionalities.

An API is available allowing the development of powerful custom user services using the same visual components and data validation mechanisms as standard EBX® user interfaces.

This chapter contains the following topics:

1. Nature
2. Declaration
3. Display
4. Legacy user services

165.1 **Nature**

User services exist in different types called *natures*. The nature defines the minimal elements (dataspace, dataset, table, record...) that need to be selected to execute the service. The following table lists the available natures.

Nature	Description
`Dataspace`	The nature of a user service that can be launched from the actions menu of a dataspace (branch or snapshot) or from any context where the current selection implies selecting a dataspace.
`Dataset`	The nature of a user service that can be launched from the actions menu of a dataset or from any context where the current selection implies selecting a dataset.
`TableView`	The nature of a user service that can be launched from the toolbar of a table, regardless of the selected view, or from any context where the current selection implies selecting a table.
`Record`	The nature of a user service that can be launched from the toolbar of a record form or from any context where the current selection implies selecting a *single* record.
`Hierarchy`	The nature of a user service that can be launched from the toolbar of a table when a hierarchy view is selected.
`HierarchyNode`	The nature of a user service that can be launched from the menu of a table hierarchy view node. Currently, only *record* hierarchy nodes are supported.
`Association`	The nature of a user service that can be launched from the target table view of an association or for any context where the current selection implies selecting the target table view of an association.
`AssociationRecord`	The nature of a user service that can be launched from the form of a target record of an association node or from any context where the current selection implies selecting a *single* association target record.

165.2 **Declaration**

A user service can be declared at two levels:

- Module,
- Data model.

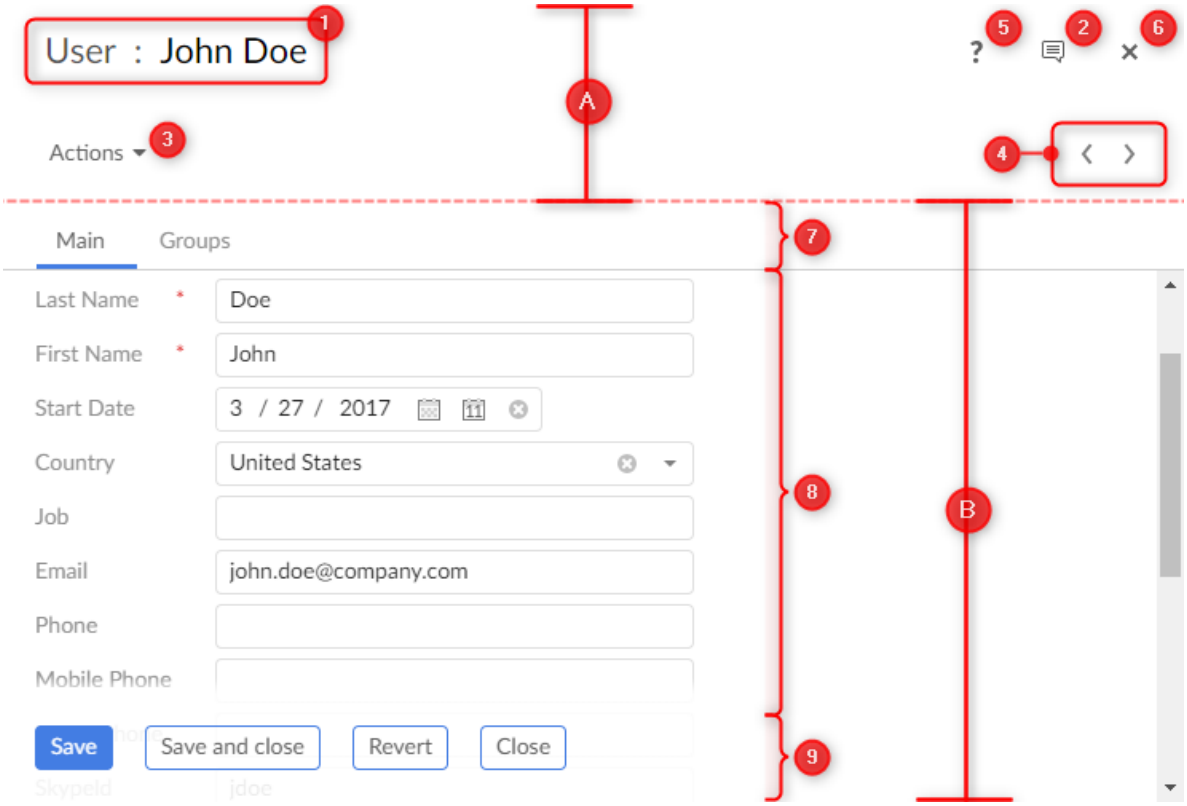
A service declared by a data model can only be launched when the current selection includes a dataset of this model. The user service cannot be of the `Dataspace` nature.

A service declared by a module may be launched for any dataspace or dataset.

The declaration can add restrictions on selections that are valid for the user service.

165.3 Display

On the following figure are displayed the functional areas of a user service.



A. Header B. Form	1. Breadcrumb 2. Message box button 3. Top toolbar 4. Navigation buttons 5. Help button 6. Close button (pop-ups only) 7. Tabs 8. Form pane (one per tab) 9. Bottom buttons

Most areas are optional and customizable. Refer to Quick start [p 1031], Implementing a user service [p 1035] and Declaring a user service [p 1049] for more details.

165.4 Legacy user services

Before the 5.8.0 version, user services were declared in XML and based on Servlet/JSP. Although this type of declaration should no longer be used, the *legacy documentation* is still available.

CHAPTER **166**

Quick start

This chapter contains the following topics:

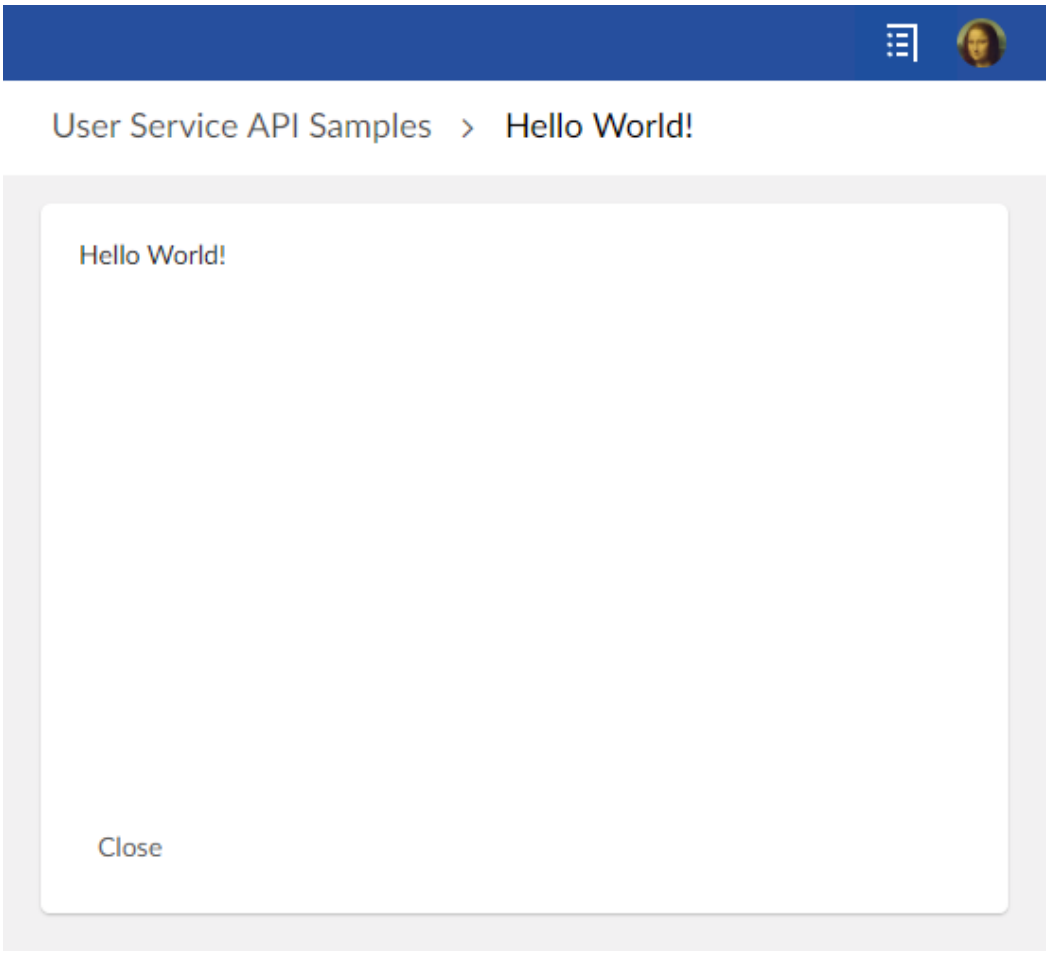
1. Main classes
2. Hello world

166.1 Main classes

The minimum requirement is to implement two classes, one for the service declaration and one for the implementation itself.

166.2 **Hello world**

The sample is a dataset user service that simply displays a "hello" message, it can be launched from the action menu of a dataset:



The service implementation class must implement the interface `UserService<DatasetEntitySelection>`:

```
/**
 * This service displays hello world!
 */
public class HelloWordService implements UserService<DatasetEntitySelection>
{
 public HelloWordService()
 {
 }

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DatasetEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  // Set bottom bar
  UIButtonSpecNavigation closeButton = aConfigurator.newCloseButton();
  closeButton.setDefaultButton(true);
  aConfigurator.setLeftButtons(closeButton);

  // Set content callback
  aConfigurator.setContent(this::writeHelloWorld);
 }

 private void writeHelloWorld(
  UserServicePaneContext aPaneContext,
```

```
  UserServicePaneWriter aWriter)
 {
  // Display Hello World!

  aWriter.add("<div ");
  aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
  aWriter.add(">");
  aWriter.add("Hello World!");
  aWriter.add("</div>");
 }

 @Override
 public void setupObjectContext(
  UserServiceSetupObjectContext<DatasetEntitySelection> aContext,
  UserServiceObjectContextBuilder aBuilder)
 {
  // No context yet.
 }

 @Override
 public void validate(UserServiceValidateContext<DatasetEntitySelection> aContext)
 {
  // No custom validation is necessary.
 }

 @Override
 public UserServiceEventOutcome processEventOutcome(
  UserServiceProcessEventOutcomeContext<DatasetEntitySelection> aContext,
  UserServiceEventOutcome anEventOutcome)
 {
  // By default do not modify the outcome.
  return anEventOutcome;
 }
}
```

The declaration class must implement the interface `UserServiceDeclaration.OnDataset`:

```
/**
 * Declaration for service hello world!
 */
public class HelloWorldServiceDeclaration implements UserServiceDeclaration.OnDataset
{
 // The service key identifies the user service.
 private static final ServiceKey serviceKey = ServiceKey.forName("HelloWorld");

 public HelloWorldServiceDeclaration()
 {
 }

 @Override
 public ServiceKey getServiceKey()
 {
  return serviceKey;
 }

 @Override
 public UserService<DatasetEntitySelection> createUserService()
 {
  // Creates an instance of the user service.
  return new HelloWordService();
 }

 @Override
 public void defineActivation(ActivationContextOnDataset aContext)
 {
  // The service is activated for all datasets instanciated with
  // the associated data model (see next example).
 }

 @Override
 public void defineProperties(UserServicePropertiesDefinitionContext aContext)
 {
  // This label is displayed in menus that can execute the user service.
  aContext.setLabel("Hello World Service");
 }

 @Override
 public void declareWebComponent(WebComponentDeclarationContext aContext)
 {
 }
}
```

In this sample, the user service is registered by a data model. The data model needs to define a schema extension that implements the following code:

```
public class CustomSchemaExtensions implements SchemaExtensions
{
 @Override
 public void defineExtensions(SchemaExtensionsContext aContext)
 {
  // Register the service.
  aContext.registerUserService(new HelloWorldServiceDeclaration());
 }
}
```

For details on the declaration of schema extensions, see `SchemaExtensions`[API].

CHAPTER 167

Implementing a user service

This chapter contains the following topics:

1. Implementation interface
2. Life cycle and threading model
3. Object Context
4. Display setup
5. Database updates
6. Ajax
7. REST data services
8. File upload
9. File download
10. User service without display

167.1 Implementation interface

The following table lists, per nature, the interface to implement:

Nature	Interface
`Dataspace`	`UserService<DataspaceEntitySelection>`
`Dataset`	`UserService<DatasetEntitySelection>`
`TableView`	`UserService<TableViewEntitySelection>`
`Record`	`UserService<RecordEntitySelection>`
`Hierarchy`	`UserService<HierarchyEntitySelection>`
`HierarchyNode`	`UserService<HierarchyNodeEntitySelection>`
`Association`	`UserService<AssociationEntitySelection>`
`AssociationRecord`	`UserService<AssociationRecordEntitySelection>`

167.2 Life cycle and threading model

The user service implementation class is:

- Instantiated at the first HTTP request by a call to its declaration **createUserService()** `UserServiceDeclaration.createUserService`[API] method.
- Discarded when the current page goes out of scope or when the session times out.

Access to this class is synchronized by TIBCO EBX® to make sure that only one HTTP request is processed at a time. Therefore, the class does not need to be thread-safe.

The user service may have attributes. The state of these attributes will be preserved between HTTP requests. However, developers must be aware that these attributes should have moderate use of resources, such as memory, not to overload the EBX® server.

167.3 Object Context

The object context is a container for objects managed by the user service. This context is initialized and modified by the user service's implementation of the method `UserService.setupObjectContext`[API].

An object of the object context is identified by an object key:

```
ObjectKey customerKey = ObjectKey.forName("customer");
```

An object can be:

- A record,
- A dataset,
- A new record not yet persisted,
- A dynamic object.

The object context is maintained between HTTP requests and usually only needs to be set up upon the first request.

Once persisted, a *new record* object is automatically changed to a *plain record* object.

As with **adaptations** `Adaptation`[API], **path** `Path`[API] expressions are used to reference a sub-element of an object.

In the following sample, a pane writer adds a form input mapped to the attribute of an object:

```
// Add an input field for customer's last name.
aWriter.setCurrentObject(customerKey);
aWriter.addFormRow(Path.parse("lastName"));
```

In the following sample, an event callback gets the value of the attribute of an object:

```
// Get value of customer's last name.
ValueContext customerValueContext = aValueContext.getValueContext(customerKey);
String lastName = customerValueContext.getValue(Path.parse("lastName"));
```

A *dynamic object* is an object whose schema is defined by the user service itself. An API is provided to define the schema programmatically. This API allows defining only instance elements (instance nodes). Defining tables is not supported. It supports most other features available with standard EBX® data models, such as types, labels, custom widgets, enumerations and constraints, including programmatic ones.

The following sample defines two objects having the same schema:

```
public class SampleService implements UserService<TableViewEntitySelection>
{
 // Define an object key per object:
 private static final ObjectKey _PersonObjectKey = ObjectKey.forName("person");
 private static final ObjectKey _PartnerObjectKey = ObjectKey.forName("partner");

 // Define a path for each property:
 private static final Path _FirstName = Path.parse("firstName");
 private static final Path _LastName = Path.parse("lastName");
 private static final Path _BirthDate = Path.parse("birthDate");

 ...

 // Define and register objects:
 @Override
 public void setupObjectContext(
  UserServiceSetupObjectContext<DataspaceEntitySelection> aContext,
  UserServiceObjectContextBuilder aBuilder)
 {
  if (aContext.isInitialDisplay())
  {
   BeanDefinition def = aBuilder.createBeanDefinition();

   BeanElement firstName = def.createElement(_FirstName, SchemaTypeName.XS_STRING);
   firstName.setLabel("First name");
   firstName.setDescription("This is the given name");
   firstName.setMinOccurs(1);

   BeanElement lastName = def.createElement(_LastName, SchemaTypeName.XS_STRING);
   lastName.setLabel("Last name");
   lastName.setDescription("This is the familly name");
   lastName.setMinOccurs(1);

   BeanElement birthDate = def.createElement(_BirthDate, SchemaTypeName.XS_DATE);
   birthDate.setLabel("Birth date");
   birthDate.addFacetMax(new Date(), false);

   aBuilder.registerBean(_PersonObjectKey, def);
   aBuilder.registerBean(_PartnerObjectKey, def);
  }

  ...
 }
```

167.4 **Display setup**

The display is set up by the user service's implementation of the method `UserService.setupDisplay`[API].

This method is called at each request and can set the following:

- The title (the default is the label specified by the user service declaration),
- The contextual help URL,
- The breadcrumbs,
- The toolbar,
- The bottom buttons.

If necessary, the header and the bottom buttons can be hidden.

The display setup is not persisted and, at each HTTP request, is reset to default before calling the method `UserService.setupDisplay`[API].

Bottom buttons

Buttons may be of two types: *action* and *submit*.

An *action* button triggers an *action* event without submitting the form. By default, the user needs to acknowledge that, by leaving the page, the last changes will be lost. This behavior can be customized.

A *submit* button triggers a *submit* event that always submits the form.

More information on events can be found in the following sections.

Content callback

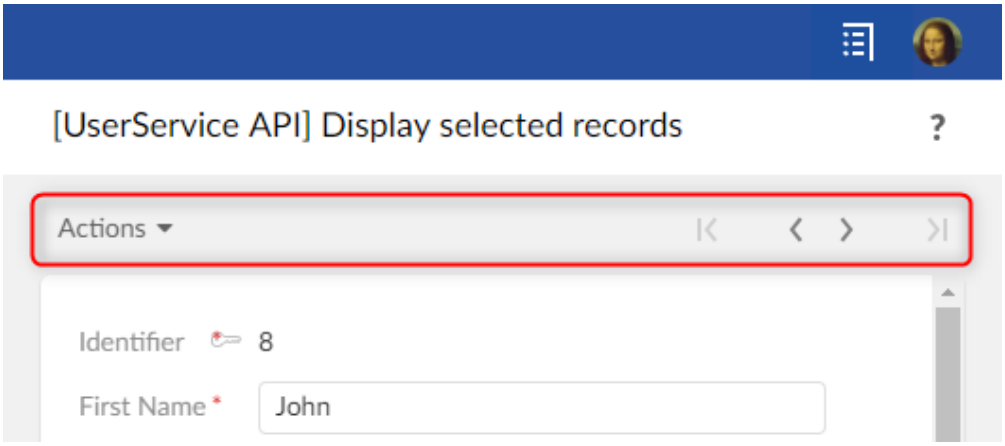
This callback usually implements the interface `UserServicePane`[API] to render a plain EBX® form. The callback can also be an instance of `UserServiceTabbedPane`[API] to render an EBX® form with tabs.

For specific cases, the callback can implement `UserServiceRawPane`[API]. This interface has restrictions but is useful when one wants to implement an HTML form that is not managed by EBX®.

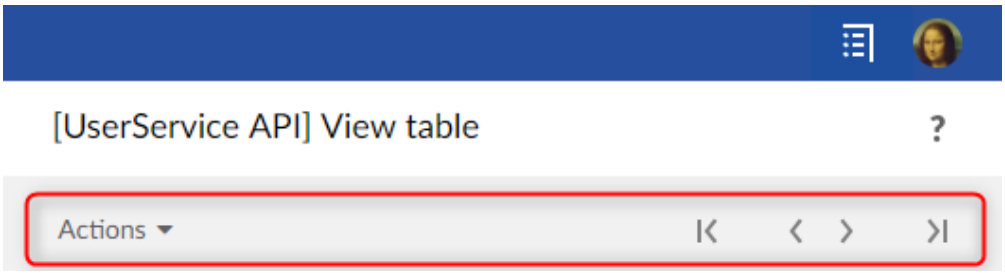
Toolbars

Toolbars are optional and come in two flavors.

The *form* style:



The *table view* style:



The style is automatically selected: toolbars defined for a *record* are of the form style and toolbars defined for a *table* are of the table view style.

Samples

The following sample implements a button that closes the current user service and redirects the user back to the current selection, only if saving the data was successful:

```
public class SampleService implements UserService<...>
{
 private static final ObjectKey _RecordObjectKey = ObjectKey.forName("record");

 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<RecordEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  ...
  // Define a "save and close" button with callback onSave().
  aConfigurator.setLeftButtons(aConfigurator.newSaveCloseButton(this::onSave));
 }

 private UserServiceEventOutcome onSave(UserServiceEventContext anEventContext)
```

```
 {
  ProcedureResult result = anEventContext.save(_RecordObjectKey);
  if (result.hasFailed())
  {
   // Save has failed. Redisplay the user message.
   return null;
  }

  // Save has succeded.Close the service.
  return UserServiceNext.nextClose();
 }
}
```

The following sample is compatible with the Java 6 syntax. Only differences with the previous code are shown:

```
public class SampleService implements UserService<...>
{
 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<RecordEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  ...
  // Define a "save and close" button with callback onSave().
  aConfigurator.setLeftButtons(aConfigurator.newSaveCloseButton(new UserServiceEvent() {
   @Override
   public UserServiceEventOutcome processEvent(UserServiceEventContext anEventContext)
   {
    return onSave(anEventContext);
   }
  }));
 }
}
```

The following sample implements a URL that closes the service and redirects the current user to another user service:

```
public class SampleService implements UserService<...>
{
 ...
 private void writePane(UserServicePaneContext aPaneContext, UserServicePaneWriter aWriter)
 {
  // Displays an ULR that redirect current user.
  String url = aWriter.getURLForAction(this::goElsewhere);
  aWriter.add("<a ");
  aWriter.addSafeAttribute("href", url);
  aWriter.add(">Go elsewhere</a");
 }

 private UserServiceEventOutcome goElsewhere(UserServiceEventContext anEventContext)
 {
  // Redirects current user to another user service.
  ServiceKey serviceKey = ServiceKey.forModuleServiceName("CustomerModule", "CustomService");
   return UserServiceNext.nextService(serviceKey);
 }
}
```

The following code is an implementation of the method `UserService.processEventOutcome`[API], sufficient for simple user services:

```
public class HelloWordService implements UserService<...>
{
 @Override
 public UserServiceEventOutcome processEventOutcome(
  UserServiceProcessEventOutcomeContext<DatasetEntitySelection> aContext,
  UserServiceEventOutcome anEventOutcome)
 {
  // By default do not modify the outcome.
  return anEventOutcome;
 }
}
```

The following sample is a more complex "wizard" service that includes three steps, each having its own `UserService.setupDisplay`[API] method:

```
// Custom outcome values.
```

```
public enum CustomOutcome implements UserServiceEventOutcome {
 displayStep1, displayStep2, displayStep3
};

// All steps of the  wizard service implement this interface.
public interface WizardStep
{
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator);
}

// The user service implementation.
public class WizardService implements UserService<...>
{
 // Attribute for current step.
 private WizardStep step = new WizardStep1();

 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  ...

  // Display current step.
  this.step.setupDisplay(aContext, aConfigurator);
 }

 @Override
 public UserServiceEventOutcome processEventOutcome(
  UserServiceProcessEventOutcomeContext<DataspaceEntitySelection> aContext,
  UserServiceEventOutcome anEventOutcome)
 {
  // Custom outcome value processing.

  if (anEventOutcome instanceof CustomOutcome)
  {
   CustomOutcome action = (CustomOutcome) anEventOutcome;
   switch (action)
   {
   case displayStep1:
    this.step = new WizardStep1();
    break;

   case displayStep2:
    this.step = new WizardStep2();
    break;

   case displayStep3:
    this.step = new WizardStep3();
    break;
   }

   // Redisplay the user service.
   return null;
  }

  // Let EBX® process the event outcome.
  return anEventOutcome;
 }
}
```

167.5 **Database updates**

An event callback may update the database.

The following sample saves two objects using a single transaction:

```
public class MultipleObjectsSampleService implements UserService<...>
{
 // This service defines a two objects having same schema.
 private static final ObjectKey _Person1_ObjectKey = ObjectKey.forName("person1");
 private static final ObjectKey _Person2_ObjectKey = ObjectKey.forName("person2");

 ...

 // Save button callback.
```

```
 private UserServiceEventOutcome onSave(UserServiceEventContext aContext)
 {
  ProcedureResult result = aContext.save(_Person1_ObjectKey, _Person2_ObjectKey);
  if (result.hasFailed())
  {
   //Save failed. Redisplay the service.
   //The user interface will automatically report error messages.
   return null;
  }

  // Save succeeded. Close the service.
  return UserServiceNext.nextClose();
 }
}
```

The following sample updates the database using a **procedure** `Procedure`[API]:

```
import com.orchestranetworks.service.*;
import com.orchestranetworks.userservice.*;

public class MultipleObjectsSampleService implements UserService<...>
{
 ...

 // Event callback.
 private UserServiceEventOutcome onUpdateSomething(UserServiceEventContext aContext)
 {
  Procedure procedure = new Procedure()
  {
   public void execute(ProcedureContext aContext) throws Exception
   {
    // Code that updates database should be here.
    ...
   }
  };

  UserServiceTransaction transaction = aContext.createTransaction();
  transaction.add(procedure);

  ProcedureResult result = transaction.execute();
  if (result.hasFailed())
  {
   aContext.addError("Procedure failed");
  }
  else
  {
   aContext.addInfo("Procedure succeeded");
  }

  return null;
}
```

167.6 **Ajax**

A user service can implement Ajax callbacks. An Ajax callback must implement the interface `UserServiceAjaxRequest`[API].

The client calls an Ajax callback using the URL generated by: `UserServiceResourceLocator.getURLForAjaxRequest`[API].

To facilitate the use of Ajax components, EBX® provides the JavaScript prototype `EBX_AJAXResponseHandler` for sending the request and handling the response. For more information on `EBX_AJAXResponseHandler` see `UserServiceAjaxRequest`[API].

The following sample implements an Ajax callback that returns partial HTML:

```
public class AjaxSampleService implements UserService<DataspaceEntitySelection>
{
 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  aConfigurator.setLeftButtons(aConfigurator.newCloseButton());
  aConfigurator.setContent(this::writePane);
```

```
 }

 /**
  * Displays an URL that will execute the callback
  * and display the returned partial HTML inside a <div> tag.
  */
 private void writePane(UserServicePaneContext aPaneContext, UserServicePaneWriter aWriter)
 {
  // Generate the URL of the Ajax callback.
  String url = aWriter.getURLForAjaxRequest(this::ajaxCallback);

  // The id of the <div> that will display the partial HTML returned by the Ajax callback.
  String divId = "sampleId";

  aWriter.add("<div ");
  aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
  aWriter.add(">");

  // Display the URL that will execute the callback.
  aWriter.add("<a ");
  aWriter.addSafeAttribute("href", "javascript:sample_sendAjaxRequest('" + url + "', '"
   + divId + "')");
  aWriter.add(">");
  aWriter.add("Click to call a user service Ajax callback");
  aWriter.add("</a>");

  // Output the <div> tag that will display the partial HTML returned by the callback.
  aWriter.add("<div ");
  aWriter.addSafeAttribute("id", divId);
  aWriter.add("></div>");

  aWriter.add("</div>");

  // JavaScript method that will send the Java request.
  aWriter.addJS_cr();
  aWriter.addJS_cr("function sample_sendAjaxRequest(url, targetDivId) {");
  aWriter.addJS_cr("  var ajaxHandler = new EBX_AJAXResponseHandler();");

  aWriter.addJS_cr("  ajaxHandler.handleAjaxResponseSuccess = function(responseContent) {");
  aWriter.addJS_cr("    var element = document.getElementById(targetDivId);");
  aWriter.addJS_cr("    element.innerHTML = responseContent;");
  aWriter.addJS_cr("  };");

  aWriter.addJS_cr("  ajaxHandler.handleAjaxResponseFailed = function(responseContent) {");
  aWriter.addJS_cr("    var element = document.getElementById(targetDivId);");
  aWriter.addJS_cr("    element.innerHTML = \"<span class='" + UICSSClasses.TEXT.ERROR
   + "'>Ajax call failed</span>\";");
  aWriter.addJS_cr("  }");

  aWriter.addJS_cr("  ajaxHandler.sendRequest(url);");
  aWriter.addJS_cr("}");
 }

 /**
  * The Ajax callback that returns partial HTML.
  */
 private void ajaxCallback(
  UserServiceAjaxContext anAjaxContext,
  UserServiceAjaxResponse anAjaxResponse)
 {
  UserServiceWriter writer = anAjaxResponse.getWriter();
  writer.add("<p style=\"color:green\">Ajax callback succeeded!</p>");
  writer.add("<p>Current data and time is: ");

  DateFormat format = DateFormat.getDateTimeInstance(
   DateFormat.FULL,
   DateFormat.FULL,
   Locale.US);
  writer.addSafeInnerHTML(format.format(new Date()));

  writer.add("</p>");
 }
}
```

167.7 **REST data services**

A user service can access REST data services through HTTP requests.

The client should use the URL generated by: `UIResourceLocator.getURLForRest`[API]. This URL includes required information for the user authentication.

For more information on REST data services see the Built-in RESTful services [p 1125].

The following sample implements a REST data service call whose response is printed in a `textarea`:

```
public class RestCallSampleService implements UserService<DataspaceEntitySelection>
{
 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  aConfigurator.setLeftButtons(aConfigurator.newCloseButton());
  aConfigurator.setContent(this::writePane);
 }

 private void writePane(UserServicePaneContext aPaneContext, UserServicePaneWriter aWriter)
 {
  // Generates the URL for REST data service call without additional parameters
  final String url = aWriter.getURLForRest("/ebx-dataservices/rest/{specificPath}", null);

  final String resultAreaId = "restResult";

  // Displays a link for REST data service call
  aWriter.add("<div ");
  aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
  aWriter.add(">");
  aWriter.add("<p>This link will display the response after making a REST call</p>");
  aWriter.add("<a ");
  aWriter.addSafeAttribute("href",
   "javascript:sendRestRequest('" + url + "', '" + resultAreaId + "')");
  aWriter.add(">");
  aWriter.add("Make the call.");
  aWriter.add("</a>");
  aWriter.add("<textarea ");
  aWriter.addSafeAttribute("id", resultAreaId);
  aWriter.add(" readonly=\"readonly\" style=\"width: 100%;\" ></textarea>");
  aWriter.add("</div>");

  // JavaScript method that will send the HTTP REST request
  aWriter.addJS_cr("function sendRestRequest(url, targetId) {");
  aWriter.addJS_cr("  var xhttp = new XMLHttpRequest();");
  aWriter.addJS_cr("  xhttp.open('GET', url, true);");
  aWriter.addJS_cr("  xhttp.setRequestHeader('Content-type', 'application/json');");
  aWriter.addJS_cr("  xhttp.send();");
  aWriter.addJS_cr("  var element = document.getElementById(targetId);");
  aWriter.addJS_cr("  xhttp.onreadystatechange = function() {");
  aWriter.addJS_cr("   if (xhttp.readyState == 4)");
  aWriter.addJS_cr("      element.innerHTML = xhttp.responseText;");
  aWriter.addJS_cr("  }");
  aWriter.addJS_cr("}");
 }
}
```

167.8 **File upload**

A user service can display forms with file input fields.

The following sample displays a form with two input fields, a title and a file:

```
public class FileUploadService implements UserService<...>
{
 // This service defines a single object named "file".
 private static final ObjectKey _File_ObjectKey = ObjectKey.forName("file");

 // Paths for the "file" object.
 public static final Path _Title = Path.parse("title");
 public static final Path _File = Path.parse("file");

 ...

 @Override
 public void setupObjectContext(
  UserServiceSetupObjectContext<DataspaceEntitySelection> aContext,
  UserServiceObjectContextBuilder aBuilder)
 {
  if (aContext.isInitialDisplay())
  {
   // Create a definition for the "model" object.
```

```
   BeanDefinition def = aBuilder.createBeanDefinition();
   aBuilder.registerBean(_File_ObjectKey, def);

   BeanElement element;

   element = def.createElement(_Title, SchemaTypeName.XS_STRING);
   element.setLabel("Title");
   element.setMinOccurs(1);

   // Type for a file must be BeanDefinition.OSD_FILE_UPLOAD.
   element = def.createElement(_File, BeanDefinition.OSD_FILE_UPLOAD);
   element.setLabel("File");
   element.setMinOccurs(1);
  }
 }

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  aConfigurator.setTitle("File upload service");
  aConfigurator.setLeftButtons(aConfigurator.newSubmitButton("Upload", this::onUpload),
 aConfigurator.newCancelButton());

  // IMPORTANT: Following method must be called to enable file upload.
  // This will set form  encryption type to  "multipart/form-data".
  aConfigurator.setFileUploadEnabled(true);

  aConfigurator.setContent(this::writePane);
 }


 private void writePane(UserServicePaneContext aContext, UserServicePaneWriter aWriter)
 {
  final UIWidgetFileUploadFactory fileUploadFactory = new UIWidgetFileUploadFactory();

  aWriter.setCurrentObject(_File_ObjectKey);

  aWriter.startTableFormRow();

  // Title input.
  aWriter.addFormRow(_Title);

  // File upload input.
  UIWidgetFileUpload widget = aWriter.newCustomWidget(_File, fileUploadFactory);
  // Default filter for file names.
  widget.setAccept(".txt");
  aWriter.addFormRow(widget);

  aWriter.endTableFormRow();
 }

 private UserServiceEventOutcome onUpload(UserServiceEventContext anEventContext)
 {
  ValueContextForInputValidation valueContext = anEventContext.getValueContext(_File_ObjectKey);

  String title = (String) valueContext.getValue(_Title);
  UploadedFile file = (UploadedFile) valueContext.getValue(_File);

  InputStream in;
  try
  {
   in = file.getInputStream();
  }
  catch (IOException e)
  {
   // Should not happen.
   anEventContext.addError("Cannot read file.");
   return null;
  }

  // Do something with title and the input stream.
  return UserServiceNext.nextClose();
 }
}
```

For more information, see `UIWidgetFileUpload`[API].

167.9 File download

A user service can display URLs or buttons to download files. The actual downloading of a file is under the control of the user service.

The following sample displays a URL to download a file:

```
public class FileDownloadService implements UserService<DataspaceEntitySelection>
{
 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  aConfigurator.setLeftButtons(aConfigurator.newCloseButton());
  aConfigurator.setContent(this::writePane);
 }

 private void writePane(UserServicePaneContext aContext, UserServicePaneWriter aWriter)
 {
  aWriter.add("<div ");
  aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
  aWriter.add(">");

  // Generate and display the URL for the download.
  String downloadURL = aWriter.getURLForGetRequest(this::processDownloadRequest);

  aWriter.add("<a ");
  aWriter.addSafeAttribute("href", downloadURL);
  aWriter.add(">Click here to download a sample file</a>");

  aWriter.add("</div>");
 }

 private void processDownloadRequest(
  UserServiceGetContext aContext,
  UserServiceGetResponse aResponse)
 {
  // The file is plain text.
  aResponse.setContentType("text/plain;charset=UTF-8");
  // Remove the following statement to display the file directly in the browser.
  aResponse.setHeader("Content-Disposition", "attachment; filename=\"sample.txt\"");

  // Write a text file using UTF-8 encoding.
  PrintWriter out;
  try
  {
   out = new PrintWriter(new OutputStreamWriter(aResponse.getOutputStream(), "UTF-8"));
  }
  catch (IOException ex)
  {
   throw new RuntimeException(ex);
  }

  DateFormat format = DateFormat.getDateTimeInstance(
   DateFormat.FULL,
   DateFormat.MEDIUM,
   Locale.US);
  Date now = new Date();

  out.println("Hello !");
  out.println("This is a sample text file downloaded on " + format.format(now)
   + ", from EBX®.");

  out.close();
 }
}
```

167.10 User service without display

A user service may be designed to execute a task without display and return to the previous screen or redirect the user to another screen.

This type of service must implement the interface **UserServiceExtended** `UserServiceExtended`[API] and method `UserServiceExtended.initialize`[API].

The following sample deletes selected records in the current table view:

```
public class DeleteRecordsService implements UserServiceExtended<TableViewEntitySelection>
{
 ...

 @Override
 public UserServiceEventOutcome initialize(
  UserServiceInitializeContext<TableViewEntitySelection> aContext)
 {
  final List<AdaptationName> records = new ArrayList<>();

  // Deletes all selected rows in a single transaction.
  RequestResult requestResult = aContext.getEntitySelection().getSelectedRecords().execute();
  try
  {
   for (Adaptation record = requestResult.nextAdaptation(); record != null; record =
 requestResult.nextAdaptation())
   {
    records.add(record.getAdaptationName());
   }
  }
  finally
  {
   requestResult.close();
  }

  Procedure deleteProcedure = new Procedure()
  {
   @Override
   public void execute(ProcedureContext aContext) throws Exception
   {
    for (AdaptationName record : records)
    {
     aContext.doDelete(record, false);
    }
   }
  };

  UserServiceTransaction transaction = aContext.createTransaction();
  transaction.add(deleteProcedure);

  // Adds an information messages for current user.
  ProcedureResult procedureResult = transaction.execute(true);
  if (!procedureResult.hasFailed())
  {
   if (records.size() <= 1)
   {
    aContext.addInfo(records.size() + " record was deleted.");
   }
   else
   {
    aContext.addInfo(records.size() + " records were deleted.");
   }
  }

  // Do not display the user service and return to current view.
  return UserServiceNext.nextClose();
 }

 @Override
 public void setupObjectContext(
  UserServiceSetupObjectContext<TableViewEntitySelection> aContext,
  UserServiceObjectContextBuilder aBuilder)
 {
  //Do nothing.
 }

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<TableViewEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  //Do nothing.
 }

 @Override
 public void validate(UserServiceValidateContext<TableViewEntitySelection> aContext)
 {
  //Do nothing.
```

```
 }

 @Override
 public UserServiceEventOutcome processEventOutcome(
  UserServiceProcessEventOutcomeContext<TableViewEntitySelection> aContext,
  UserServiceEventOutcome anEventOutcome)
 {
  return anEventOutcome;
 }
}
```

Known limitation

If such service is called in the context of a Web component, an association, a perspective action or a hierarchy node, The service will be launched, initialized and closed, but the service's target entity will still be displayed.

CHAPTER 168

Declaring a user service

This chapter contains the following topics:

1. Declaration interface
2. Life cycle and threading model
3. Registration
4. Service properties
5. Service activation scope
6. Web component declaration
7. User service groups

168.1 Declaration interface

The following table lists, per nature, the interface that the declaration class of a user service must implement:

Nature	Declaration **Interface**
`Dataspace`	`UserServiceDeclaration.OnDataspace`
`Dataset`	`UserServiceDeclaration.OnDataset`
`TableView`	`UserServiceDeclaration.OnTableView`
`Record`	`UserServiceDeclaration.OnRecord`
`Hierarchy`	`UserServiceDeclaration.OnHierarchy`
`HierarchyNode`	`UserServiceDeclaration.OnHierarchyNode`
`Association`	`UserServiceDeclaration.OnAssociation`
`AssociationRecord`	`UserServiceDeclaration.OnAssociationRecord`

168.2 Life cycle and threading model

The user service declaration class is instantiated at the TIBCO EBX® startup and must be coded to be thread-safe. This is usually not an issue as most implementations should be immutable classes.

168.3 Registration

A user service declaration must be registered by a module or a data model.

Registration by a module is achieved by the module registration servlet by a code similar to:

```
public class CustomRegistrationServlet extends ModuleRegistrationServlet
{
 @Override
 public void handleServiceRegistration(ModuleServiceRegistrationContext aContext)
 {
  // Register custom user service declaration.
  aContext.registerUserService(new CustomServiceDeclaration());
 }
}
```

For more information on the module registration servlet, see module registration [p 880] and `ModuleRegistrationServlet`[API].

Registration by a data model is achieved by a code similar to:

```
public class CustomSchemaExtensions implements SchemaExtensions
{
 @Override
 public void defineExtensions(SchemaExtensionsContext aContext)
 {
  // Register custom user service declaration.
  aContext.registerUserService(new CustomServiceDeclaration());
 }
}
```

For more information on data model extensions, see `SchemaExtensions`[API].

168.4 Service properties

The properties of a user service include its *label*, *description*, *confirmation message* and the *group* that owns the service. All are optional but it is a good practice to at least define the label.

For more information, see `UserServiceDeclaration.defineProperties`[API].

168.5 Service activation scope

The activation scope defines on which selection the service is available.

Example of a service activation definition:

```
public class CustomServiceDeclaration implements UserServiceDeclaration.OnTableView
{
 ...

 @Override
 public void defineActivation(ActivationContextOnTableView aContext)
 {
  // activates the service in all dataspaces except the "Reference" branch.
  aContext.includeAllDataspaces(DataspaceType.BRANCH);
  aContext.excludeDataspacesMatching(Repository.REFERENCE, DataspaceChildrenPolicy.NONE);

  // activates the service only on tables "table01" and "table03".
  aContext.includeSchemaNodesMatching(
   CustomDataModelPath._Root_Table01.getPathInSchema(),
   CustomDataModelPath._Root_Table03.getPathInSchema());

  // service will be enabled only when at least one record is selected.
```

```
  aContext.forbidEmptyRecordSelection();

  // service will not be displayed in hierarchical views (neither in the
  // top toolbar, nor in the hierarchy nodes' menu).
  aContext.setDisplayForLocations(
   ActionDisplaySpec.HIDDEN,
   ToolbarLocation.HIERARCHICAL_VIEW_TOP,
   ToolbarLocation.HIERARCHICAL_VIEW_NODE);

  // service will be considered as disabled if not explicitly enabled
  // via the UI.
  aContext.setDefaultPermission(UserServicePermission.getDisabled());
 }
}
```

For more information about declaring the activation scope, see `UserServiceDeclaration.defineActivation`[API].

For more information about the resolution of the user service availability, see Resolving permissions on services [p 504].

168.6 Web component declaration

Parameters declaration and availability in workflows and perspectives

User services are automatically available as web components with a set of built-in parameters depending on the service's nature. To define custom parameters and/or set the service web component as available when configuring a workflow user task, a perspective menu action or a toolbar web component action, `UserServiceDeclaration.declareWebComponent`[API] must be used.

Example of a web component declaration:

```
public class CustomServiceDeclaration implements UserServiceDeclaration.OnDataset
{
 ...

 @Override
 public void declareWebComponent(WebComponentDeclarationContext aContext)
 {
  // makes this web component available when configuring a workflow user task.
  aContext.setAvailableAsWorkflowUserTask(true);

  // adds a custom input parameter.
  aContext.addInputParameter(
   "source",
   UserMessage.createInfo("Source"),
   UserMessage.createInfo("Source of the imported data."));

  // modifies the built-in "instance" parameter to be "input/output" instead of "input".
  aContext.getBuiltInParameterForOverride("instance").setOutput(true);
 }
}
```

See Using TIBCO EBX® as a Web Component [p 419] for more information.

User service extension

It is possible to extend existing user services (built-in or custom) in order to add input/output parameters when using these services as web components.

In order to do so, a user service extension must first be registered by a module or a data model.

Registration by a module is achieved by the module registration servlet by code similar to:

```
public class CustomRegistrationServlet extends ModuleRegistrationServlet
{
 ...

 @Override
 public void handleServiceRegistration(ModuleServiceRegistrationContext aContext)
```

```
 {
  // Register user service extension declaration.
  aContext.registerUserServiceExtension(new ServiceExtensionDeclaration());
 }
}
```

For more information on the module registration servlet, see module registration [p 880] and `ModuleRegistrationServlet`[API].

Registration by a data model is achieved by a code similar to:

```
public class CustomSchemaExtensions implements SchemaExtensions
{
 ...

 @Override
 public void defineExtensions(SchemaExtensionsContext aContext)
 {
  // Register user service extension declaration.
  aContext.registerUserServiceExtension(new ServiceExtensionDeclaration());
 }
}
```

For more information on the data model extension, see `SchemaExtensions`[API].

168.7 User service groups

User service groups are used to organize the display of user services in menus and permission management screens.

The following types of service groups are available:

- Built-in User Service Groups [p 1052] provided by EBX®,
- Custom User Service Groups [p 1053] declared in a module.

The link between groups and services is made upon service declaration. See Associating a service to a group [p 1053].

Built-in User Service Groups

Available built-in service groups:

Service Group Key	Description
`@ebx-importExport`	Group containing all built-in import and export services provided by EBX®. In the default menus, these services are displayed in an "Import / Export" sub-menu.
`@ebx-views`	Group containing services to display in the 'Views' menu. Unlike other service groups, services associated with this group are not displayed in the default menus, but only in the 'Views' menu displayed in the non-customizable part of the table top toolbar. These services can still be added manually to a custom toolbar.

Declaring a User Service Group

User Service Groups must be declared while registering the module, using the method `ModuleServiceRegistrationContext.registerServiceGroup`[API]:

```
public class CustomRegistrationServlet extends ModuleRegistrationServlet
{
 ...

 @Override
 public void handleServiceRegistration(ModuleServiceRegistrationContext aContext)
 {
  // In CustomModuleConstants,
  // CUSTOM_SERVICE_GROUP_KEY = ServiceGroupKey.forServiceGroupInModule("customModule", "customGroup")

  // registers CUSTOM_SERVICE_GROUP_KEY service group
  aContext.registerServiceGroup(
   CustomModuleConstants.CUSTOM_SERVICE_GROUP_KEY,
   UserMessage.createInfo("Custom group"),
   UserMessage.createInfo("This group contains services related to..."));
 }
}
```

Associating a service to a group

The association of a service with a group is made at its **declaration** `UserServiceDeclaration`[API], using the method `UserServicePropertiesDefinitionContext.setGroup`[API]:

```
public class CustomServiceDeclaration implements UserServiceDeclaration.OnDataset
{
 ...

 @Override
 public void defineProperties(UserServicePropertiesDefinitionContext aContext)
 {
  // associates the current service to the CUSTOM_SERVICE_GROUP_KEY group
  aContext.setGroup(CustomModuleConstants.CUSTOM_SERVICE_GROUP_KEY);
 }
}
```

A service can be associated with either a built-in or a custom service group. In the latter case, this service will be displayed in this built-in group, just like other built-in services belonging to this group.

CHAPTER 169

Development recommendations

This chapter contains the following topics:

1. HTML
2. CSS
3. JavaScript

169.1 HTML

It is recommended to minimize the inclusion of specific HTML styles and tags to allow the default styles of TIBCO EBX® to apply to custom interfaces. The approach of the API is to automatically apply a standardized style to all elements on HTML pages, while simplifying the implementation process for the developer.

XHTML

EBX® is a Rich Internet Application developed in XHTML 1.0 Transitional. It means that the structure of the HTML is strict XML file and that all tags must be closed, including "br" tags. This structure allows for greater control over CSS rules, with fewer differences in browser rendering.

iFrames

Using iFrame is allowed in EBX®, especially in collaboration with a URL of a `UIHttpManagerComponent`[API]. For technical reasons, it is advised to set the `src` attribute of an iFrame using JavaScript only. In this way, the iFrame will be loaded once the page is fully rendered and when all the built-in HTML components are ready.

Example

The following example, developed from any `UIComponentWriter`[API], uses a `UIHttpManagerComponent`[API] to build the URL of an iFrame, and set it in the right way:

```
// using iFrame in the current page requires a sub session component
UIHttpManagerComponent managerComponent = writer.createWebComponentForSubSession();

// [...] managerComponent configuration

String iFrameURL = managerComponent.getURIWithParameters();

String iFrameId = "mySweetIFrame";

// place the iFrame in the page, with an empty src attribute
writer.add("<iframe id=\"").add(iFrameId).add("\" src=\"\" >").add("</iframe>");

// launch the iFrame from JavaScript
```

```
writer.addJS("document.getElementById(\"").addJS(iFrameId).addJS("\").src = \"").addJS(iFrameURL).addJS("\";");
```

169.2 CSS

Public CSS classes

The constant catalog `UICSSClasses`[API] offers the main CSS classes used in the software to style the components. These CSS classes ensure a proper long-term integration into the software, because they follow the background colors, borders, customizable text in the administration; the floating margins and paddings fluctuate according to the variable density; to the style of the icons, etc.

See also *`UICSSUtils`*[API]

Advanced CSS

EBX® allows to integrate to all its pages one or more external Cascading Style Sheet. These external CSS, considered as resources, need to be declared in the Module registration [p 880].

In order to ensure the proper functioning of your CSS rules and properties without altering the software, the following recommendations should be respected. Failure to respect these rules could lead to:

- Improper functioning of the software, both aesthetically and functionally: risk of losing the display of some of the data and some input components may stop working.
- Improper functioning of your CSS rules and properties, since the native CSS rules will impact the CSS implementation.

Reserved prefixes for CSS identifiers and class names

The following prefixes should not be used to create CSS #ids and .classes.

ebx_	Internal built-in
yui	Yahoo User Interface global
ygtv	Yahoo User Interface tree view
layout-doc	Yahoo User Interface layout
cke_	CK editor (used by HTML editor widget)
fa	Font Awesome (icons used by perspectives and toolbars)

CSS classes used internally by EBX®

The following CSS classes should never be included in a ruleset that has no contextual selector.

If you do not prefix your CSS selector using one of the CSS classes below, it will cause conflicts and corrupt the UI of EBX®.

selected	YUI selected tree node
hd	YUI floating pane header
bd	YUI floating pane body
ft	YUI floating pane footer
container-close	YUI inner popup close button
underlay	YUI inner popup shadow
hastitle	YUI menu group with title
topscrollbar	YUI menu top scroll zone
bottomscrollbar	YUI menu bottom scroll zone
withtitle	YUI calendar
link-close	YUI calendar close button
collapse	YUI layout closed pane indicator
pull-right	Font Awesome parameter
pull-left	Font Awesome parameter

Examples to avoid conflicts

Don't

```
.selected {
 background-color: red;
}
```

Do

```
#myCustomComponent li.selected {
 background-color: red;
}
```

169.3 JavaScript

Public JS functions

The catalog of JavaScript functions `JavaScriptCatalog`[API] offers a list of functions to use directly (through copy-paste) in the JS files.

JavaScript call during page generation in Java

When generating the HTML of a Java component, it is possible to add specific JavaScript code with the API `UIJavaScriptWriter`[API].

This JavaScript is executed once the whole page is loaded. It is possible to instantly manage the HTML elements written with `UIBodyWriter.add`[API]. Setting on-load functions (such as `window.onload = myFunctionToCallOnload;`) is not supported because the execution context comes after the on-load event.

Advanced JavaScript

EBX® allows to include one or more external JavaScript files. These external JavaScript files, considered as resources, need to be declared in the Module registration [p 880]. For performance reasons, it is recommended to include the JavaScript resource only when necessary (in a User service or a specific form, for example). The API `UIDependencyRegisterer`[API] allows a developer to specify the conditions for which the JavaScript resources will be integrated into a given page according to its context.

In order to ensure the proper functioning of your JavaScript resources without altering the software, the following recommendations should be respected. Failure to respect them could lead to:

- Improper functioning of the software: if functions or global variables of the software were to be erased, some input or display components (including the whole screen) may stop working.
- Improper functioning of your JavaScript instructions, since global variables or function names could be erased.

Reserved JS prefixes

The following prefixes are reserved and should not be used to create variables, functions, methods, classes, etc.

ebx_	Internal built-in API
EBX_	Internal built-in API
YAHOO	Yahoo User Interface API

SOAP data services

CHAPTER 170

Introduction

This chapter contains the following topics:

1. Overview
2. Activation and configuration
3. Interactions
4. Security
5. Monitoring
6. SOAP and REST comparative
7. Limitations

170.1 Overview

Data services allow external systems to interact with the data governed in the TIBCO EBX® repository using the SOAP/Web Services Description Language (WSDL) standards.

In order to invoke SOAP operations [p 1079], for an integration use case, a WSDL [p 1071] must be generated from a data model. It will be possible to perform operations such as:

- Selecting, inserting, updating, deleting, or counting records
- Selecting dataset values
- Getting the differences on a table between dataspaces or snapshots, or between two datasets based on the same data model
- Getting the credentials of records

Other generic WSDLs can be generated and allow performing operations such as:

- Creating, merging, or closing a dataspace
- Creating or closing a snapshot
- Validating a dataset, dataspace, or a snapshot
- Starting, resuming or ending a data workflow

- Administrative operations to manage access to the UI or to system information

> **Note**
>
> See SOAP and REST comparative [p 1068].

170.2 Activation and configuration

Data services are enabled by deploying the `ebx-dataservices` web application along with the other EBX® modules. See Jakarta EE deployment [p 533] for more information.

In case of specific deployment, for example using reverse-proxy mode, see URLs computing [p 590] for more information.

The default method for accessing data services is over HTTP, although it is also possible to use JMS for the SOAP operations. See JMS configuration [p 588] and Using JMS [p 1062] for more information.

170.3 Interactions

Input and output message encoding

All input messages must be *exclusively* in UTF-8. All output messages are in UTF-8.

Tracking information

Depending on the data services operation being called, it may be possible to specify session tracking information.

- Example for a SOAP operation, the request header contains:

```
<SOAP-ENV:Header>
 <!-- optional security header here -->
 <m:session xmlns:m="urn:ebx-schemas:dataservices_1.0">
  <trackingInformation>String</trackingInformation>
 </m:session>
</SOAP-ENV:Header>
```

For more information, see `Session.getTrackingInfo`[API] in the Java API.

Session parameters

Depending on the data services operation being called, it is possible to specify session input parameters. They are defined in the request body.

Input parameters are available on custom Java components with a session object, such as: triggers, access rules, `custom` web services. They are also available on data workflow operations.

- Example for a SOAP operation, the optional request header contains:

```
<SOAP-ENV:Header>
 <!-- optional security header here -->
 <m:session xmlns:m="urn:ebx-schemas:dataservices_1.0">
  <!-- optional trackingInformation header here -->
  <inputParameters>
   <parameter>
    <name>String</name>
    <value>String</value>
   </parameter>
   <!-- for some other parameters, copy complex
        element 'parameter' -->
  </inputParameters>
 </m:session>
</SOAP-ENV:Header>
```

For more information, see `Session.getInputParameterValue`[API] in the Java API.

Exception handling

In case of unexpected server error upon execution of:

- A SOAP operation, a SOAP exception response is returned to the caller via the `soap:Fault` element. For example:

```
<soapenv:Fault>
 <faultcode>soapenv:java.lang.IllegalArgumentException</faultcode>
 <faultstring />
 <faultactor>admin</faultactor>
 <detail>
  <m:StandardException xmlns:m="urn:ebx-schemas:dataservices_1.0">
   <code>java.lang.IllegalArgumentException</code>
   <label/>
   <description>java.lang.IllegalArgumentException:
    Parent home not found at
    com.orchestranetworks.XX.YY.ZZ.AA.BB(AA.java:44) at
    com.orchestranetworks.XX.YY.ZZ.CC.DD(CC.java:40) ...
   </description>
  </m:StandardException>
 </detail>
</soapenv:Fault>
```

Using JMS

It is possible to access SOAP operations using JMS instead of HTTP. The JMS architecture relies on one JMS request queue (mandatory), on one JMS failure queue (optional), and on JMS response queues, see configuration JMS [p 588]. The mandatory queue is the input queue. Request messages must be put in the input queue, and response messages are put by EBX® in the `replyTo` queue of the JMS request. The optional queue is the failure queue which allows you to replay an input message if necessary. If the queue is set and activated in the configuration file and an exception occurs while handling a request message, this input message will be copied in the failure queue.

The relationship between a request and a response is made by copying the `messageId` message identifier field of the JMS request into the `correlId` correlation identifier field of the response.

JMS location points must be defined in the Lineage administration in order to specialize the generated WSDL. If no specific location point is given, the default value will be `jms:queue:jms/EBX_QueueIn`.

170.4 Security

Authentication

Authentication is mandatory to access to the content. Several authentication methods are available and described below. The descriptions are ordered by priority (EBX® applies the highest priority authentication method first).

- 'Basic Authentication Scheme' method is based on the HTTP-Header `Authorization` in base 64 encoding, as described in RFC 2617 (Basic Authentication Scheme).

```
If the user agent wishes to send the userid "Alibaba" and password "open sesame",
it will use the following header field:
> Authorization: Basic QWxpYmFiYTpvcGVuIHNlc2FtZQ==
```

- 'Bearer Authentication Scheme' method works with bearer tokens that may be issued from a third party provider.

```
> Authorization: bearer <accessToken>
```

For more information on this authentication scheme, see Bearer authentication configuration. [p 580]

- 'Standard Authentication Scheme' is based on the HTTP `Request`. User and password are extracted from request parameters. For more information on request parameters, see Parameters [p 1074] section.

  For more information on this authentication scheme, see `Directory.authenticateUserFromLoginPassword`[API].

- The 'SOAP Security Header Authentication Scheme' method is based on the Web Services Security UsernameToken Profile 1.0 specification.

  By default, the type `PasswordText` is supported. This is done with the following SOAP-Header defined in the WSDL:

```
<SOAP-ENV:Header>
 <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
  <wsse:UsernameToken>
   <wsse:Username>String</wsse:Username>
   <wsse:Password Type="wsse:PasswordText">String</wsse:Password>
  </wsse:UsernameToken>
 </wsse:Security>
</SOAP-ENV:Header>
```

> **Note**
>
> Only available for SOAP operations [p 1079].

- 'Specific authentication Scheme' is based on the HTTP `Request`. An implementation of this method can, for example, extract a password-digest or a ticket from the HTTP request. See `Directory.authenticateUserFromHttpRequest`[API] for more information.
- The 'SOAP Specific Header Authentication Scheme'.

  For more information, see Overriding the SOAP security header [p 1063].

Global permissions

Global access permissions can be independently defined for the SOAP and WSDL connector accesses.

See also *Global permissions [p 671]*

Overriding the SOAP security header

It is possible to override the default WSS header in order to define another security authentication mechanism. Such an override is taken into account for both HTTP and JMS. To define and override,

use the 'SOAP Header Security declaration' configuration settings under Administration > Lineage, which includes the following fields:

Schema location	The URI of the Security XML Schema to import into the WSDL.
Target namespace	The target namespace of elements in the schema.
Namespace prefix	The prefix for the target namespace.
Message name	The message name to use in the WSDL.
Root element name	The root element name of the security header. The name must be the same as the one declared in the schema.
wsdl:part element name	The name of the `wsdl:part` of the message.

The purpose of overriding the default security header is to change the declaration of the WSDL message matching the security header so that it contains the following:

```
<wsdl:definitions ... xmlns:MyPrefix="MyTargetNameSpace" ...
  ...
  <xs:schema ...>
    <xs:import namespace="MyTargetNameSpace" schemaLocation="MySchemaURI"/>
    ...
  </xs:schema>
  ...
  <wsdl:message name="MySecurityMessage">
    <wsdl:part name="MyPartElementName" element="MyPrefix:MySecurityRootElement"/>
  </wsdl:message>
  ...
  <wsdl:operation name="...">
    <soap:operation soapAction="..." style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
      <soap:header message="impl:MySecurityMessage" part="MyPartElementName" use="literal"/>
  ...
  </wsdl:operation>
</wsdl:definitions>
```

The corresponding XML Schema header declaration would be as follows:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="MyNameSpace"
  xmlns:MyPrefix="MyNameSpace">
  <element name="MySecurityRootElement" type="MyPrefix:SpecificSecurity"/>
  <complexType name="SpecificSecurity">
    <sequence>
      <element name="AuthToken" type="string"/>
    </sequence>
  </complexType>
</schema>
```

A SOAP message using the XML schema and configuration above would have the following header:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Header>
    <m:MySecurityRootElement xmlns:m="MyNameSpace">
      <AuthToken>String</AuthToken>
    </m:MySecurityRootElement>
    ...
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    ...
  </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

To handle this non-default header, you must implement the method: `Directory.authenticateUserFromSOAPHeader`[API].

> **Note**
>
> Only available for SOAP operations [p 1079].

Lookup mechanism

Because EBX® offers several authentication methods, a lookup mechanism based on conditions was set to know which method should be applied for a given request. The method application conditions are evaluated according to the authentication scheme priority. If the conditions are not satisfied, the server evaluates the next method. The following table presents the available authentication methods

for each supported protocol and their application conditions. They are ordered from the highest priority to the lowest.

Operation / Protocol	Authentication methods and application conditions
SOAP / JMS	SOAP Security Header [p 1063] • The SOAP request is received over the JMS protocol. • The SOAP header content must contains a `Security` element. SOAP Specific Header [p 1063] • The SOAP request is received over the JMS protocol. • The SOAP header content must not contain a `Security` element.
SOAP / HTTP	Basic [p 1062] • The HTTP request must hold an `Authorization` header. • `Authorization` header value must start with the word `Basic`. • No `login` is provided in the URL parameters. Standard [p 1063] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters. SOAP Security Header [p 1063] • The SOAP header content must contain a `Security` element. • The HTTP request must not hold an `Authorization` header. • No `login` is provided in the URL parameters. Specific [p 1063] • The HTTP request must not satisfy the conditions of the previous authentication methods. SOAP Specific Header [p 1063] • The SOAP header content must not contain a `Security` element. • The HTTP request must not hold an `Authorization` header. • No `login` is provided in the URL parameters.
WSDL / HTTP	Basic [p 1062] • The HTTP request must not hold an `Authorization` header. • `Authorization` header value must start with the word `Basic`. • No `login` is provided in the URL parameters. Standard [p 1063] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters. Specific [p 1063] • The HTTP request must not satisfy the conditions of the previous authentication methods.

In case of multiple authentication methods present in the same request, EBX® will return an HTTP code `401 Unauthorized`.

170.5 Monitoring

Data service events can be monitored through the log category `ebx.dataServices`, as declared in the EBX® main configuration file. For example, `ebx.log4j.category.log.dataServices= INFO, ebxFile:dataservices`.

See also

Configuring the EBX® logs *[p 582]*

TIBCO EBX® main configuration file *[p 571]*

170.6 **SOAP and REST comparative**

Operations	**SOAP**	**REST**
Data		
Select metamodel		X
Select or count records (with filter and/or view publication)	X	X
Select records with association values (first level)	X	
Select business objects		X
Update or delete records (with filter and/or view publication)		X
Selector for possible enumeration values (with filter)		X
Prepare for creation or duplication		X
Insert, update or delete records	X	X
Select or count history records (with filter and/or view publication)		X
Select node values from dataset	X	X
Update node value from dataset		X
Get table or dataset changes between dataspaces or snapshots	X	
Refresh a replication unit	X	
Get credentials for records	X	
Generate service contract	WSDL	OpenAPI
Views		
Look up published table views		X
Datasets		
Select dataset information		X
Select root or child datasets		X

Operations	SOAP	REST
Dataspaces		
Select dataspace or snapshot information		X
Select root or children dataspaces, or select snapshots		X
Create, close, merge a dataspace	X	X
Create, close a snapshot	X	X
Validate a dataspace or a snapshot	X	
Validate a dataset	X	
Locking, unlocking a dataspace	X	X
Workflow		
Start, resume or end a workflow	X	
Administration		
Manage the default directory content 'Users', 'Roles'... tables.	X	X
Open, close the user interface	X	X
Select, insert, update, delete operations for administration dataset		X
Select the system information	X	X
Staging API		X
Health operations		X
Other		
Develop custom services from the Java API (*)		X
Generate custom service contracts (*)		OpenAPI

(*) See REST Toolkit [p 1285] for more information.

170.7 **Limitations**

Date, time & dateTime format

Data services only support the following date and time formats:

Type	Format	Example
xs:date	yyyy-MM-dd	2007-12-31
xs:time	HH:mm:ss or HH:mm:ss.SSS	11:55:00
xs:dateTime	yyyy-MM-ddTHH:mm:ss or yyyy-MM-ddTHH:mm:ss.SSS	2007-12-31T11:55:00

SOAP naming convention

Due to the naming convention of the data service operations, each table defined within a data model must have a unique name for the WSDL generation.

CHAPTER 171

WSDL generation

This chapter contains the following topics:

1. Supported standard
2. Operation types
3. WSDL download methods
4. HTTP examples

171.1 Supported standard

TIBCO EBX® generates a WSDL that complies with the W3C Web Services Description Language 1.1 standard.

171.2 Operation types

A WSDL can be generated for different types of operations:

Operation type	WSDL description
`custom`	WSDL for EBX® add-ons.
`dataset`	WSDL for dataset and replication operations.
`directory`	WSDL for default EBX® directory operations. It is also possible to filter data using the `tablePaths` [p 1075] or `operations` [p 1075] parameters.
`repository`	WSDL for dataspace or snapshot management operations.
`tables`	WSDL for operations on the tables of a specific dataset.
`userInterface`	Deprecated since version 5.8.1. This operation type has been replaced by `administration`. While the user interface management operations are still available for backward compatibility reasons, it is recommended to no longer use this type. WSDL for user interface management operations (these operations can only be accessed by administrators).
`administration`	WSDL for administration operations like: • user interface management • system information retrieval These operations can only be accessed by administrators.
`workflow`	WSDL for EBX® workflow management operations.

171.3 WSDL download methods

EBX® supports the following methods:

- from the user interface
- from HTTP protocol

A WSDL can only be downloaded by authorized profiles:

Operation type	Access right permissions
`custom`	All profiles, if at least one web service is registered.
`dataset`	All profiles.
`directory`	All profiles, if the following conditions are valid: • No specific directory implementation is used. (The built-in Administrator role is only subject to this condition). • Global access permissions are defined for the administration. • 'Directory' dataset permissions have writing access for the current profile.
`repository`	All profiles.
`tables`	All profiles.
`userInterface`	Deprecated since version 5.8.1. This operation type has been replaced by `administration`. While the user interface management operations are still available for backward compatibility reasons, it is recommended to no longer use this type. Built-in administrator role or delegated administrator profiles, if all conditions are valid: • Global access permissions are defined for the administration. • 'User interface' dataset permissions have writing access for the current profile.
`administration`	Built-in administrator role or delegated administrator profiles, if all conditions are valid: • Global access permissions are defined for the administration. • 'Administration' dataset permissions have write access for the current profile.
`workflow`	All profiles.

WSDL download from the user interface

An authorized user can download an EBX® WSDL from the data services administration area.

> **Note**
>
> See Generating a WSDL for dataspace operations [p 248] in the user guide for more information.

WSDL download from HTTP protocol

An application can download an EBX® WSDL using `GET` or `POST` HTTP method. The application has to be authenticated using a profile with appropriate rights.

URL format

```
http[s]://<host>[:<port>]/<ebx-dataservices>/{type}[/{dataspace}[/{dataset}]]?
{queryParameters}
```

Where:

- `<ebx-dataservices>` corresponds to the 'ebx-dataservices.war' web application's path. The path is composed by multiple, or none, URI segments followed by the web application's name.
- `{type}` corresponds to the operation type [p 1072].
- `{dataspace}` corresponds to the dataspace or snapshot identifier.
- `{dataset}` corresponds to the dataset name.
- `{queryParameters}` corresponds to common or dedicated operation parameters passed through the URL.

Parameters

A request parameter can be specified by one of the following methods:

- a `PathParam` which corresponds to a path segment from the URL (recommended)

- a `QueryParam` which corresponds to a standard HTTP parameter with value.

Parameter name	`PathParam`	`QueryParam`	Required	Description
`WSDL`	no	yes	yes	Specifies the WSDL download operation. Empty value.
`login`	no	yes	no	Specifies the user identifier. Required when the standard authentication method is used. `String` type value.
`password`	no	yes	no	Specifies the user password. Required when the standard authentication method is used. `String` type value.
`type`	yes	no	yes	Specifies the operation type [p 1072]. Possible values are: `custom`, `dataset`, `directory`, `administration`, `userInterface`, `repository`, `tables` or `workflow`. `String` type value.
`branch` `version`	yes	yes	(*)	Specifies the dataspace or the snapshot identifier. (*) required for `tables` and `dataset` types, ignored otherwise. `String` type value.
`instance`	yes	yes	(*)	Specifies the dataset name. `String` type value.
`tablePaths`	no	yes	no	Specifies the list of table paths. Optional for `tables` or `directory` types, ignored otherwise. If not defined, all tables are selected. Each table path is separated by a comma character. `String` type value.
`operations`	no	yes	no	Allows generating a WSDL for an operations subset. Optional for `tables` or `directory` operation types, ignored otherwise. If not defined, all operations for the given type are generated. This parameter's value is a concatenation of one or more of the following characters: • C = Count record(s) • D = Delete record(s) • E = Get credentials • G = Get changes • I = Insert record(s) • U = Update record(s)

Parameter name	`PathParam`	`QueryParam`	Required	Description
				• R = Read operations (equivalent to CEGS) • S = Select record(s) • W = Write operations (equivalent to DIU) `String` type value.
`namespaceURI`	yes	yes	(**)	Specifies the unique name space URI of the custom web service. (**)Is required when `type` parameter is set to `custom`, ignored otherwise. `URI` type value.
`attachmentFilename`	no	yes	(***)	Specifies the attachment file name. (***) optional if `isContentInAttachment` parameter is defined and set to `true`, ignored otherwise. `String` type value.
`isContentInAttachment`	no	yes	no	Specifies if the WSDL is downloaded as an attachment. `Boolean` type value. Default value is `false`.
`targetNamespace`	no	yes	no	Overrides the target namespace URI of the WSDL. `URI` type value. Default value is `urn:ebx:ebx-dataservices`.

Message body

No message body is required.

HTTP codes

An HTTP code is always returned. Errors are indicated by a code above 400.

Status code	Information
`200` *(OK)*	The WSDL content was successfully generated and is returned by the request (optionally in an attachment [p 1076]).
`400` *(Bad request)*	The request is incorrect. This occurs when: • A request element is incorrect. • The unicity check on table names contains at least one error. **Note** See WSDL and table operations [p 970] for more information.
`401` *(Unauthorized)*	Request requires an authenticated user.
`403` *(Forbidden)*	Request is not allowed for the authenticated user.
`405` *(Method not allowed)*	Request is not allowed in this configuration.
`500` *(Internal error)*	Request generates an error (a stack trace and a detailed error message are returned).

Response body

The response body depends on the returned status code and on the requested WSDL content.

- `200` *(OK)*: the HTTP header `Content-Type` is set to `text/xml;charset=UTF-8`.

  If the content is in attachment, the HTTP header `Content-Disposition` is set to `attachment; filename*=UTF-8''<filename.wsdl>`.
- `4xx`: A detailed message is returned in the body. The HTTP header `Content-Type` is set to `text/html;charset=utf-8`.

171.4 HTTP examples

Some of the following examples are displayed in two methods: `PathParam` and `QueryParam`.

- The WSDL will contain all repository operations, using standard authentication method.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/repository?
  WSDL&login=<login>&password=<password>
  ```
- The WSDL will contain all workflow operations.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/workflow?WSDL
  ```
- The WSDL will contain all tables operations for the `'dataset1'` dataset in `'dataspace1'` dataspace.

  `PathParam`

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables/dataspace1/dataset1?WSDL
```

```
QueryParam
```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables?
WSDL&branch=<dataspace1>&instance=<dataset1>
```

- The WSDL will contain all tables with only read operations for the `'dataset1'` dataset in `'dataspace1'` dataspace.

```
PathParam
```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables/dataspace1/dataset1?
WSDL&operations=R
```

```
QueryParam
```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables?
WSDL&branch=dataspace1&instance=dataset1&operations=R
```

- The WSDL will contain the two selected tables operations for the `'dataset1'` dataset in `'dataspace1'` dataspace.

```
PathParam
```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables/dataspace1/dataset1?
WSDL&tablePaths=/root/table1,/root/table2
```

```
QueryParam
```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables?
WSDL&branch=dataspace1&instance=dataset1&tablePaths=/root/table1,/root/table2
```

- The WSDL will contain custom web service operations for the dedicated URI.

```
PathParam
```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/custom/urn:ebx-
test:com.orchestranetworks.dataservices.WSDemo?WSDL
```

```
QueryParam
```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/custom?WSDL&namespaceURI=urn:ebx-
test:com.orchestranetworks.dataservices.WSDemo
```

CHAPTER 172

SOAP operations

This chapter contains the following topics:

1. Operations generated from a data model
2. Operations on datasets and dataspaces
3. Operations on data workflows
4. Administrative services

172.1 Operations generated from a data model

For a data model used in an TIBCO EBX® repository, it is possible to dynamically generate a corresponding WSDL, that defines its operations. When using this WSDL, it will be possible to read and/or write in the EBX® repository. For example, for a table located at the path `/root/XX/exampleTable`, the generated requests would follow the structure of its underlying data model and include the name of the table `<m:{operation}_exampleTable xmlns:m="urn:ebx-schemas:dataservices_1.0">`.

Attention

Since the WSDL and the SOAP operations tightly depend on the data model structure, it is important to redistribute the up-to-date WSDL after any data model change.

Content policy

Access to the content of records, the presence or absence of XML elements, depend on the resolved permissions [p 487] of the authenticated user session. Additional aspects, detailed below, can impact the content.

Disabling fields from data model

The `hiddenInDataServices` property, defined in the data model, allows always hiding fields in data services, regardless of the user profile. This parameter has an impact on the generated WSDL: any hidden field or group will be absent from the request and response structure.

Modifying the `hiddenInDataServices` parameter value has the following impact on a client which would still use the former WSDL:

- On request, if the data model property has been changed to `true`, and if the concerned field is present in the WSDL request, an exception will be thrown.

- On response, if the schema property has been changed to `false`, WSDL validation will return an error if it is activated.

This setting of "Default view" is defined inside data model.

See also

Hiding a field in Data Services *[p 964]*

Permissions *[p 487]*

Association field

Read-access on table records can export the association fields as displayed in UI Manager. This feature can be coupled with the 'hiddenInDataServices' model parameter.

> **Note**
>
> Limitations: change and update operations do not manage association fields. Also, the select operation only exports the first level of association elements (the content of associated objects cannot contain association elements).

Common request parameters

Several parameters are common to several operations and are detailed below.

Element	Description	Required
branch	The identifier of the dataspace to which the dataset belongs.	Either this parameter or the 'version' parameter must be defined. Required for the 'insert', 'update' and 'delete' operations.
version	The identifier of the snapshot to which the dataset belongs.	Either this parameter or the 'branch' parameter must be defined
instance	The unique name of the dataset which contains the table to query.	Yes
predicate	XPath predicate [p 441] defines the records on which the request is applied. If empty, all records will be retrieved, except for the 'delete' operation where this field is mandatory. **See also** *`includesMetadata` parameter [p 1083]*	Only required for the 'delete' operation
data	Contains the records to be modified, represented within the structure of their data model. The whole operation is equivalent to an XML import. The details of the operations performed on data content are specified in the section Import [p 153].	Only required for the insert and update operations
viewPublication	This parameter can be combined with the predicate [p 1081] parameter as a logical `AND` operation. The behavior of this parameter is described in the section EBX® as a Web Component [p 422]. It cannot be used if the 'viewId' parameter is used, and cannot be used on hierarchical nor tile views.	No
viewId	*Deprecated since version 5.2.3*. This parameter has been replaced by the parameter 'viewPublication'. While it remains available for backward compatibility, it will eventually be removed in a future version. This parameter cannot be used if the 'viewPublication' parameter is used.	No
blockingConstraintsDisabled	This property is available for all table updates data service operations. If `true`, the validation process disables blocking constraints defined in the data model. If this parameter is not present, the default is `false`. See Blocking and non-blocking constraints [p 945] for more information.	No
details	The `details` element specifies the following option:	No

Element	Description	Required
	The optional attribute `locale` (default 'en-US') defines the language of the blockingConstraintsDisabled [p 1081] parameter in which the validation messages must be returned.	
disableRedirectionToLastBroadcast	This property is available for all data service operations. If `true`, access to a delivery dataspace on a D3 primary node is not redirected to the last broadcast snapshot. Otherwise, access to such a dataspace is always redirected to the last snapshot broadcast. If this parameter is not present, the default is `false` (redirection on a D3 master enabled), unless the configuration property ebx.dataservices.disableRedirectionToLastBroadcast.default [p 587] has been set. If the specified dataspace is not a delivery dataspace on a D3 primary node, this parameter is ignored.	No

Select operations

Select request on table

```
<m:select_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <includesMetadata>String</includesMetadata>
 <viewPublication>String</viewPublication>
 <exportCredentials>boolean</exportCredentials>
 <pagination>
  <previousPageLastRecordPredicate>String</previousPageLastRecordPredicate>
  <pageSize>Integer</pageSize>
 </pagination>
 <disableRedirectionToLastBroadcast>boolean</disableRedirectionToLastBroadcast>
</m:select_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 1081].	
version	See the description under Common parameters [p 1081].	
instance	See the description under Common parameters [p 1081].	
predicate	See the description under Common parameters [p 1081]. This parameter can be combined with the viewPublication [p 1081] parameter as a logical `AND` operation.	
viewPublication	See the description under Common parameters [p 1081].	

includesMetadata	The response contains the specified metadata. Each returned record contains additional elements called steps under the `ebx-metadata` root element.
	For instance, a 'system' step value will contain technical data. See also the optimistic locking [p 1098] section:
	`<system>`
	`  <uuid>7FE03810-6A67-11ED-A892-00FF20216100</uuid>`
	`  <creator>Uadmin</creator>`
	`  <creation_time>2022-11-12T00:07:40.858</creation_time>`
	`  <updater>Uadmin</updater>`
	`  <update_time>2022-11-12T00:07:40.831</update_time>`
	`</system>`
	String value steps (default value is empty) must be separated by a comma: `system, teamUp`. All steps can be returned using the `ebx-all` value.
		No

includesTechnicalData	*Deprecated since version 6.1.0 and replaced by `includesMetadata` request parameter.* The response will contain technical data if `true`. See also the optimistic locking [p 1098] section.
	Each returned record will contain additional attributes for this technical information, for instance:
	`...<table`
	`ebxd:lastTime="2010-06-28T10:10:31.046"`
	` ebxd:lastUser="Uadmin"`
	`ebxd:uuid="9E7D0530-828C-11DF-B733-0012D01B6E76">... .`
		No

exportCredentials	If `true` the select will also return the credentials for each record.	No
pagination	Specifies the pagination configuration, see child elements below.	No
pageSize (nested under the `pagination` element)	Specifies the maximum number of records per page. `Integer` type, default value is 10. **See also** *ebx.dataservices.pagination.pageSize.default* [p 587] The value 0 selects the maximum recommended page size, default is 10000. **See also** *ebx.dataservices.pagination.maxSize.default* [p 587]	No

Element	Description	Required
	Note If the specified or default page size value exceeds the maximum recommended page size, the maximum page size is used.	
previousPageLastRecordPredicate (nested under the `pagination` element)	When pagination is enabled, XPath predicate that defines the record after which the page must fetched, this value is provided by the previous response, as the element `lastRecordPredicate`. If the passed record is not found, the first page will be returned.	No
disableRedirectionToLastBroadcast	See the description under Common parameters [p 1082].	

Select response on table

```
<ns1:select_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <data>
  <XX>
   <TableName>
    <a>key1</a>
    <b>valueb</b>
    <c>1</c>
    <d>1</d>
   </TableName>
  </XX>
 </data>
 <credentials>
  <XX>
   <TableName predicate="./a='key1'">
    <a>W</a>
    <b>W</b>
    <c>W</c>
    <d>W</d>
   </TableName>
  </XX>
 </credentials>
 <lastRecordPredicate>./a='key1'</lastRecordPredicate>
</ns1:select_{TableName}Response>
```

with:

Element	Description
data	Content of records that are displayed following the table path.
credentials	Contains the access right for each node of each record.
lastRecordPredicate	Only returned if the pagination is enabled, this defines the last records in order to be used on the next request in the element `previousPageLastRecordPredicated`.

See also the optimistic locking [p 1098] section.

Select request on dataset

This operation returns dataset content without table.

```
<m:selectInstance xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
```

```
</m:selectInstance>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 1081].	
version	See the description under Common parameters [p 1081].	
instance	See the description under Common parameters [p 1081].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 1082].	

Select response on dataset

```
<ns1:selectInstanceResponse xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <data>
  <settings>
   <XX>
    <a>key1</a>
    <b>valueb</b>
    <c>1</c>
    <d>true</d>
   </XX>
  </settings>
 </data>
</ns1:selectInstanceResponse>
```

with:

Element	Description
data	Dataset content without table.

Delete operation

Deletes records or, for a child dataset, defines the record state as "occulting" or "inherited" according to the record context. Records are selected by the `predicate` parameter.

Delete request

```
<m:delete_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <instance>String</instance>
 <predicate>String</predicate>
 <includeOcculting>boolean</includeOcculting>
 <inheritIfInOccultingMode>boolean</inheritIfInOccultingMode>
 <checkNotChangedSinceLastTime>dateTime</checkNotChangedSinceLastTime>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
 <disableRedirectionToLastBroadcast>Boolean</disableRedirectionToLastBroadcast>
</m:delete_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 1081].	
instance	See the description under Common parameters [p 1081].	
predicate	See the description under Common parameters [p 1081].	
includeOcculting	Includes the records in occulting mode. Default value is `false`.	No
inheritIfInOccultingMode	Inherits the record from its parent if it is in occulting mode. Default value is `false`.	No
occultIfInherit	*Deprecated since version 5.7.0* Occults the record if it is in inherit mode. Default value is `false`.	No
checkNotChangedSinceLastTime	Timestamp used to ensure that the record has not been modified since the last read. Also see the optimistic locking [p 1098] section.	No
blockingConstraintsDisabled	See the description under Common parameters [p 1081].	
details	See the description under Common parameters [p 1081].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 1082].	

Delete response

If one of the provided parameters is illegal, if a required parameter is missing, if the action is not authorized or if no record is selected, an exception is returned. Otherwise, the specific response is returned:

```
<ns1:delete_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <blockingConstraintMessage>String</blockingConstraintMessage>
</ns1:delete_{TableName}Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. '95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted.
blockingConstraintMessage	This element is present if the status is equal to '95' with a localized message. The locale of the message is retrieved from the request parameter or from the user session.

Count operation

Count request

```
<m:count_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <disableRedirectionToLastBroadcast>boolean</disableRedirectionToLastBroadcast>
</m:count_{TableName}>
```

with:

Element	Description
branch	See the description under Common parameters [p 1081].
version	See the description under Common parameters [p 1081].
instance	See the description under Common parameters [p 1081].
predicate	See the description under Common parameters [p 1081].
disableRedirectionToLastBroadcast	See the description under Common parameters [p 1082].

Count response

```
<ns1:count_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <count>Integer</count>
</ns1:count_{TableName}Response>
```

with:

Element	Description
count	The number of records that correspond to the predicate in the request.

Update operation

Update request

```
<m:update_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <instance>String</instance>
 <updateOrInsert>boolean</updateOrInsert>
 <byDelta>boolean</byDelta>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
 <disableRedirectionToLastBroadcast>boolean</disableRedirectionToLastBroadcast>
 <data>
  <XX>
   <TableName>
    <a>String</a>
    <b>String</b>
    <c>String</c>
    <d>String</d>
    ...
   </TableName>
```

```
  </XX>
 </data>
</m:update_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 1081].	
instance	See the description under Common parameters [p 1081].	
updateOrInsert	If `true` and the record does not currently exist, the operation creates the record. `boolean` type, the default value is `false`.	No
byDelta	If `true` and an element does not currently exist in the incoming message, the target value is not changed. If `false` and node is declared `hiddenInDataServices`, the target value is not changed. The complete behavior is described in the sections Insert and update operations [p 154].	No
blockingConstraintsDisabled	See the description under Common parameters [p 1081].	
details	See the description under Common parameters [p 1081].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 1082].	
data	See the description under Common parameters [p 1081].	

See also *Optimistic locking [p 1098]*

Update response

```
<ns1:update_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <blockingConstraintMessage>String</blockingConstraintMessage>
</ns1:update_{TableName}Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. '95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted.
blockingConstraintMessage	This element is present if the status is equal to '95' with a localized message. The locale of the message is retrieved from the request parameter or from the user session.

Insert operation

Insert request

```
<m:insert_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <instance>String</instance>
 <byDelta>boolean</byDelta>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
 <disableRedirectionToLastBroadcast>boolean</disableRedirectionToLastBroadcast>
 <data>
  <XX>
   <TableName>
    <a>String</a>
    <b>String</b>
    <c>String</c>
    <d>String</d>
    ...
   </TableName>
  </XX>
 </data>
</m:insert_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 1081].	
instance	See the description under Common parameters [p 1081].	
byDelta	If `true` and an element does not currently exist in the incoming message, the target value is not changed. If `false` and node is declared `hiddenInDataServices`, the target value is not changed. The complete behavior is described in the sections Insert and update operations [p 154].	No
blockingConstraintsDisabled	See the description under Common parameters [p 1081].	
details	See the description under Common parameters [p 1081].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 1082].	
data	See the description under Common parameters [p 1081].	

Insert response

```
<ns1:insert_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <blockingConstraintMessage>String</blockingConstraintMessage>
 <inserted>
  <predicate>./a='String'</predicate>
 </inserted>
</ns1:insert_{TableName}Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. '95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted.
blockingConstraintMessage	This element is present if the status is equal to '95' with a localized message. The locale of the message is retrieved from the request parameter or from the user session.
predicate	A predicate matching the primary key of the inserted record. When several records are inserted, the predicates follow the declaration order of the records in the input message.

Get changes operations

Returns changes according to the Content policy [p 1079].

Get changes requests

Changes between two datasets:

```
<m:getChangesOnDataSet_{schemaName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <compareWithBranch>String</compareWithBranch>
 <compareWithVersion>String</compareWithVersion>
 <compareWithInstance>String</compareWithInstance>
 <resolvedMode>boolean</resolvedMode>
 <includeInstanceUpdates>boolean</includeInstanceUpdates>
</m:getChangesOnDataSet_{schemaName}>
```

Changes between two tables:

```
<m:getChanges_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <compareWithBranch>String</compareWithBranch>
 <compareWithVersion>String</compareWithVersion>
 <resolvedMode>boolean</resolvedMode>
 <pagination>
  <pageSize>Integer</pageSize>
 </pagination>
 <disableRedirectionToLastBroadcast>boolean</disableRedirectionToLastBroadcast>
</m:getChanges_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 1081].	
version	See the description under Common parameters [p 1081].	
instance	See the description under Common parameters [p 1081].	
compareWithBranch	The identifier of the dataspace with which to compare.	One of either this parameter or the 'compareWithVersion [p 1091]' parameter must be defined.
compareWithVersion	The identifier of the snapshot with which to compare.	One of either this parameter or the 'compareWithBranch [p 1091]' parameter must be defined.
compareWithInstance	The identifier of the dataset with which to compare. If it is undefined, instance [p 1091] parameter is used.	No
resolvedMode	Defines whether or not the difference is calculated in resolved mode. Default is `true`. See **Resolved mode** `DifferenceHelper.resolvedMode`[API] for more information.	No
includeInstanceUpdates	Defines if the content updates of the dataset are included. Default is `false`.	No
pagination	Enables pagination context for the operations `getChanges` and `getChangesOnDataSet`. Allows client to define pagination context size. Each page contains a collection of inserted, updated and/or deleted records of tables according to the maximum size. Get changes persisted context is built at first call according to the page size parameter in request. The pagination context is persisted on the server file system [p 669] and allows invoking the next page until last page or when a timeout is reached. For creation: Defines `pageSize` parameter. For next: Defines `context` element with `identifier` from previous response. Enables pagination, see child elements below.	No
pageSize (nested under `pagination` element)	Defines maximum number of records in each page. Minimal size is `50`.	No (Only for creation)
context (nested under `pagination` element)	Defines content of pagination context.	No (Only for next)

Element	Description	Required
identifier (nested under `context` element)	Pagination context identifier.	Yes
disableRedirectionToLastBroadcast	See the description under Common parameters [p 1082].	

> **Note**
>
> If none of the *compareWithBranch* or *compareWithVersion* parameters are specified, the comparison will be made with their parent:
>
> - if the current dataspace or snapshot is a dataspace, the comparison is made with its initial snapshot (includes all changes made in the dataspace);
> - if the current dataspace or snapshot is a snapshot, the comparison is made with its parent dataspace (includes all changes made in the parent dataspace since the current snapshot was created);
> - returns an exception if the current dataspace is the 'Reference' dataspace.

See also *`DifferenceHelper`*[API]

Get changes responses

Changes between two datasets:

```
<ns1:getChangesOnDataSet_{schemaName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <updated>
  <changes>
   <path>... Path of changed terminal value ...</path>
   <path>...</path>
  </changes>
  <data>
   ... see the whole content of dataset values (without table) ...
  </data>
 </updated>
 <getChanges_{TableName1}>
  ... see the getChanges between tables response example ...
 </getChanges_{TableName1}>
 <getChanges_{TableName2}>
  ... see the getChanges between tables response example ...
 </getChanges_{TableName2}>
 ...
</ns1:getChangesOnDataSet_{schemaName}Response>
```

Changes between two tables:

```
<ns1:getChanges_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <inserted>
  <XX>
   <TableName>
    <a>AVALUE3</a>
    <b>BVALUE3</b>
    <c>CVALUE3</c>
    <d>DVALUE3</d>
   </TableName>
  </XX>
 </inserted>
 <updated>
  <changes>
   <change predicate="./a='AVALUE2'">
    <path>/b</path>
    <path>/c</path>
   </change>
  </changes>
  <data>
   <XX>
    <TableName>
     <a>AVALUE2</a>
```

```
      <b>BVALUE2.1</b>
      <c>CVALUE2.1</c>
      <d>DVALUE2</d>
     </TableName>
    </XX>
   </data>
  </updated>
  <deleted>
   <predicate>./a='AVALUE1'</predicate>
  </deleted>
</ns1:getChanges_{TableName}Response>
```

with:

Element	Description	Required
inserted	Contains inserted record(s) from choice `compareWithBranch` or `compareWithVersion`. Content under this element corresponding to an XML export of inserted records.	No
updated	Contains updated record(s) or dataset content.	No
changes (nested under `updated` element)	Only the group of field have been updated.	Yes
change (nested under `changes` element)	Group of fields have been updated with own `XPath predicate` attribute of the record.	Yes
path (nested under `change` element)	Path in the record.	Yes
path (nested under `changes` element)	Path in the dataset.	Yes
data (nested under `updated` element)	Content under this element corresponding to an XML export of dataset or updated records.	No
deleted	Records have been deleted from context of request. Content corresponding to a list of `predicate` element who contains the XPath predicate of record.	No
pagination	When pagination is enabled on request. Get changes persisted context allows invoking the next page until last page or when the context timeout is reached. Contains a next page: Defines `context` element with `identifier`. Is the last page: Defines `context` element without `identifier`. Enables pagination, see child elements below.	No
context (nested under `pagination` element)	Defines content of pagination context.	Yes (Only for next and last)
identifier (nested under `context` element)	Pagination context identifier. Not defined at last returned page.	No
pageNumber (nested under `context` element)	Current page number in pagination context.	Yes
totalPages (nested under `context` element)	Total pages in pagination context.	Yes

Get changes operation with pagination enabled

Only `pagination` element and sub elements have been described.

For creation:

Extract of request:

```
...
 <pagination>
  <!-- on first request for creation -->
  <pageSize>Integer</pageSize>
 </pagination>
...
```

Extract of response:

```
...
 <pagination>
  <!-- on next request to continue -->
  <context>
   <identifier>String</identifier>
   <pageNumber>Integer</pageNumber>
   <totalPages>Integer</totalPages>
  </context>
 </pagination>
...
```

For next:

Extract of request:

```
...
 <pagination>
  <context>
   <identifier>String</identifier>
  </context>
 </pagination>
...
```

Extract of response:

```
...
 <pagination>
  <!-- on next request to continue -->
  <context>
   <identifier>String</identifier>
   <pageNumber>Integer</pageNumber>
   <totalPages>Integer</totalPages>
  </context>
 </pagination>
...
```

For last:

Extract of request:

```
...
 <pagination>
  <context>
   <identifier>String</identifier>
  </context>
 </pagination>
...
```

Extract of response:

```
...
 <pagination>
  <context>
   <pageNumber>Integer</pageNumber>
   <totalPages>Integer</totalPages>
  </context>
 </pagination>
...
```

Get credentials operation

Get credentials request

```
<m:getCredentials_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <viewPublication>String</viewPublication>
 <disableRedirectionToLastBroadcast>boolean</disableRedirectionToLastBroadcast>
</m:getCredentials_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 1081].	
version	See the description under Common parameters [p 1081].	
instance	See the description under Common parameters [p 1081].	
predicate	See the description under Common parameters [p 1081].	
viewPublication	See the description under Common parameters [p 1081].	

Get credentials response

```
<ns1:getCredentials_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <XX>
  <TableName>
   <a>R</a>
   <b>W</b>
   <c>H</c>
   <d>W</d>
   ...
  </TableName>
 </XX>
</ns1:getCredentials_{TableName}Response>
```

With the following possible values:

- R: for read-only
- W: for read-write
- H: for hidden

Multiple chained operations

Multiple operations request

It is possible to run multiple operations across tables in the dataset, while ensuring a consistent response. The operations are executed sequentially, according to the order defined on the client side.

All operations are executed in a single transaction with a `SERIALIZABLE` isolation level. If all requests in the multiple operation are read-only, they are allowed to run fully concurrently along with other read-only transactions, even in the same dataspace.

When an error occurs during one operation in the sequence, all updates are rolled back and the client receives a `StandardException` error message with details.

See Concurrency [p 886].

```
<m:multi_ xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <disableRedirectionToLastBroadcast>boolean</disableRedirectionToLastBroadcast>
 <details locale="Locale"/>
 <request id="id1">
  <{operation}_{TableName}>
  ...
  </{operation}_{TableName}>
 </request>
 <request id="id2">
  <{operation}_{TableName}>
  ...
  </{operation}_{TableName}>
 </request>
</m:multi_>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 1081].	
version	See the description under Common parameters [p 1081].	
instance	See the description under Common parameters [p 1081].	
blockingConstraintsDisabled	See the description under Common parameters [p 1081].	
details	See the description under Common parameters [p 1081].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 1082].	
request	This element contains one operation, like a single operation without branch, version and instance parameters. This element can be repeated multiple times for additional operations. Each request can be identified by an 'id' attribute. In a response, this 'id' attribute is returned for identification purposes. Operations such as `count`, `select`, `getChanges`, `getCredentials`, `insert`, `delete` or `update`.	Yes

Note:

- Does not accept a limit on the number of `request` elements.
- The request `id` attribute must be unique in multi-operation requests.
- If all operations are read only (`count`, `select`, `getChanges`, or `getCredentials`) then the whole transaction is set as read-only for performance considerations.

Limitation:

- The `multi` operation applies to one model and one dataset (parameter `instance`).

See also

Procedure[API]

Repository[API]

Multiple operations response

See each response operation for details.

```
<ns1:multi_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <response id="id1">
  <{operation}_{TableName}Response>
  ...
  </{operation}_{TableName}Response>
 </response>
 <response id="id2">
  <{operation}_{TableName}Response>
  ...
  </{operation}_{TableName}Response>
 </response>
</ns1:multi_Response>
```

with:

Element	Description
response	This element contains the response of one operation. It is be repeated multiple times for additional operations. Each response is identified by an 'id' attribute set in the request or automatically generated. The content of the element corresponds to the response of a single operation, such as `count`, `select`, `getChanges`, `getCredentials`, `insert`, `delete` or `update`.

Optimistic locking

To prevent an update or a delete operation on a record that was read earlier but may have changed in the meantime, an optimistic locking mechanism is provided.

A select request can include technical information by adding the element `includesMetadata` with `system` value:

```
<m:select_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <includesMetadata>system</includesMetadata>
</m:select_{TableName}>
```

Use the value of the `ebx-metadata/system/update_time` element in the following update request. If the record was changed since the specified time, the update cancels. To prevent update of a modified record, add the `ebx-metadata/system/update_time` element value.

```
<m:update_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
    <version>String</version>
 <instance>String</instance>
 <updateOrInsert>true</updateOrInsert>
 <data>
  <XX>
   <TableName>
    <a>String</a>
    <b>String</b>
    <c>String</c>
    <d>String</d>
    ...
    <ebx-metadata>
```

```
     <system>
      ...
      <update_time>2010-06-28T10:10:31.046</update_time>
     </system>
    </ebx-metadata>
   </TableName>
  </XX>
 </data>
</m:update_{TableName}>
```

The value of the `lastTime` attribute can also be used to prevent deletion on a modified record:

```
<m:delete_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <checkNotChangedSinceLastTime>2010-06-28T10:10:31.046</checkNotChangedSinceLastTime>
</m:delete_{TableName}>
```

> **Note**
>
> You can use the `checkNotChangedSinceLastTime` element more than once, but only for the same record. This implies that if the `predicate` element returns more than one record, the request will fail.

172.2 **Operations on datasets and dataspaces**

Parameters for operations on dataspaces and snapshots are as follows:

Element	*Description*	*Required*
branch	Identifier of the target dataspace on which the operation is applied. When not specified, the 'Reference' dataspace is used except for the merge dataspace operation where it is required.	One of either this parameter or the 'version' parameter must be defined. Required for the dataspace merge, locking, unlocking and replication refresh operations.
version	Identifier of the target snapshot on which the operation is applied.	One of either this parameter or the 'branch' parameter must be defined
versionName	Identifier of the snapshot to create. If empty, it will be defined on the server side.	No
childBranchName	Identifier of the dataspace child to create. If empty, it will be defined on the server side.	No
instance	The unique name of the dataset on which the operation is applied.	Required for the replication refresh operation.
ensureActivation	Defines if validation must also check whether this instance is activated.	Yes
details	Defines if validation returns details. The optional attribute `severityThreshold` defines the lowest severity level of message to return. The possible values are, in descending order of severity, 'fatal', 'error', 'warning', or 'info'. For example, setting the value to 'error' will return error and fatal validation messages. If this attribute is not defined, all levels of messages are returned by default. The optional attribute `locale` (default 'en-US') defines the language in which the validation messages are to be returned.	No. If not specified, no details are returned.
owner	Defines the owner. Must respect the inner format as returned by `Profile.format`[API].	No
branchToCopyPermissionFrom	Defines the identifier of the dataspace from which to copy the permissions.	No
documentation	Documentation for a dedicated language.	No

Element	*Description*	*Required*
	Multiple `documentation` elements may be used for several languages.	
locale (nested under the `documentation` element)	Locale of the documentation.	Only required when the `documentation` element is used
label (nested under the `documentation` element)	Label for the language.	No
description (nested under the `documentation` element)	Description for the language.	No

Validate a dataspace

Validate dataspace request

```
<m:validate xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
</m:validate>
```

Validate dataspace response

```
<ns1:validate_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <validationReport>
  <instanceName>String</instanceName>
  <fatals>boolean</fatals>
  <errors>boolean</errors>
  <infos>boolean</infos>
  <warnings>boolean</warnings>
 </validationReport>
</ns1:validate_Response>
```

Validate a dataset

Validate dataset request

```
<m:validateInstance xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <ensureActivation>boolean</ensureActivation>
 <details severityThreshold="fatal|error|warning|info" locale="Locale"/>
</m:validateInstance>
```

Validate dataset response

```
<ns1:validateInstance_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <validationReport>
  <instanceName>String</instanceName>
  <fatals>boolean</fatals>
  <errors>boolean</errors>
  <infos>boolean</infos>
  <warnings>boolean</warnings>
  <details>
   <reportItem>
    <severity>{fatal|error|warning|info}</severity>
    <message>
     <internalId />
     <text>String</text>
    </message>
    <subject>
     <table>Path</table>
     <predicate>String</predicate>
```

```
     <path>Path</path>
    </subject>
   </reportItem>
  </details>
 </validationReport>
</ns1:validateInstance_Response>
```

Create a dataspace

Create dataspace request

```
<m:createBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <owner>String</owner>
 <branchToCopyPermissionFrom>String</branchToCopyPermissionFrom>
 <documentation>
  <locale>Locale</locale>
  <label>String</label>
  <description>String</description>
 </documentation>
 <childBranchName>String</childBranchName>
</m:createBranch>
```

Create dataspace response

```
<ns1:createBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <childBranchName>String</childBranchName>
</ns1:createBranch_Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully.

Create a snapshot

Create snapshot request

```
<m:createVersion xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <versionName>String</versionName>
 <owner>String</owner>
 <documentation>
  <locale>Locale</locale>
  <label>String</label>
  <description>String</description>
 </documentation>
</m:createVersion>
```

Create snapshot response

```
<ns1:createVersion_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <versionName>String</versionName>
</ns1:createVersion_Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully.

Locking a dataspace

Lock dataspace request

```
<m:lockBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <durationToWaitForLock>Integer</durationToWaitForLock>
 <message>
  <locale>Locale</locale>
  <label>String</label>
 </message>
</m:lockBranch>
```

with:

Element	*Description*	*Required*
durationToWaitForLock	This parameter defines the maximum duration (in seconds) that the operation waits for a lock before aborting.	No, does not wait by default
message	User message of the lock. Multiple `message` elements may be used.	No
locale (nested under the `message` element)	Locale of the user message.	Only required when the `message` element is used
label (nested under the `message` element)	The user message.	No

Lock dataspace response

```
<ns1:lockBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:lockBranch_Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully. '94' indicates that the dataspace has been already locked by another user. Otherwise, a SOAP exception is thrown.

Unlocking a dataspace

Unlock dataspace request

```
<m:unlockBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
</m:unlockBranch>
```

Unlock dataspace response

```
<ns1:unlockBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:unlockBranch_Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. Otherwise, a SOAP exception is thrown.

Merge a dataspace

Merge dataspace request

```
<m:mergeBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <deleteDataOnMerge>boolean</deleteDataOnMerge>
 <deleteHistoryOnMerge>boolean</deleteHistoryOnMerge>
</m:mergeBranch>
```

with:

Element	*Description*	*Required*
deleteDataOnMerge	This parameter is available for the merge dataspace operation. Sets whether the specified dataspace and its associated snapshots will be deleted upon merge. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.dataDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 587]. See Deleting data and history [p 667] for more information.	No
deleteHistoryOnMerge	This parameter is available for the merge dataspace operation. Sets whether the history associated with the specified dataspace will be deleted upon merge. Default value is `false`. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.historyDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 587]. See Deleting data and history [p 667] for more information.	No

> **Note**
>
> The merge decision step is bypassed during merges performed through data services. In such cases, the data in the child dataspace automatically overrides the data in the parent dataspace.

Merge dataspace response

```
<ns1:mergeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:mergeBranch_Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully.

Close a dataspace or snapshot

Close dataspace or snapshot request

Close dataspace request:

```
<m:closeBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <deleteDataOnClose>boolean</deleteDataOnClose>
 <deleteHistoryOnClose>boolean</deleteHistoryOnClose>
</m:closeBranch>
```

Close snapshot request:

```
<m:closeVersion xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <version>String</version>
 <deleteDataOnClose>boolean</deleteDataOnClose>
</m:closeVersion>
```

with:

Element	*Description*	*Required*
deleteDataOnClose	This parameter is available for the close dataspace and close snapshot operations. Sets whether the specified snapshot, or dataspace and its associated snapshots, will be deleted upon closure. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine this default value by specifying the property `ebx.dataservices.dataDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 587]. See Deleting data and history [p 667] for more information.	No
deleteHistoryOnClose	This parameter is available for the close dataspace operation. Sets whether the history associated with the specified dataspace will be deleted upon closure. Default value is `false`. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.historyDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 587]. See Deleting data and history [p 667] for more information.	No

Close dataspace or snapshot response

Close dataspace response:

```
<ns1:closeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:closeBranch_Response>
```

Close snapshot request:

```
<ns1:closeVersion_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:closeVersion_Response>
```

Replication refresh

Replication refresh request

```
<m:replicationRefresh_${schema} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <instance>String</instance>
 <unitName>String</unitName>
</m:replicationRefresh_${schema}>
```

with:

Element	*Description*	*Required*
branch	See the description under Common parameters [p 1100].	Yes
instance	See the description under Common parameters [p 1100].	Yes
unitName	Name of the replication unit. **See also** *Replication refresh information [p 470]*	Yes

Replication refresh response

```
<ns1:replicationRefresh_${schema}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:replicationRefresh_${schema}Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully.

172.3 Operations on data workflows

Parameters for data workflows operations are retrieved from the SOAP header in the session.

Deprecated since version 5.7.0 to define parameters in the SOAP message body.

See session parameters [p 1061] for more information.

Element	Description	Required
parameters	*Deprecated since version 5.7.0* While it remains available for backward compatibility, it will eventually be removed in a future major version. **Note** The `parameters` element is ignored if at least one session parameter has been defined.	No
parameter (nested under the `parameters` element). Multiple `parameter` elements may be used.	An input parameter for the workflow.	No
name (nested under the `parameter` element)	Name of the parameter.	Yes
value (nested under the `parameter` element)	Value of the parameter.	No

Start a workflow

Start a workflow from a workflow launcher. It is possible to start a workflow with localized documentation and specific input parameters (with name and optional value).

> **Note**
>
> The workflow creator is initialized from the session and the workflow priority is retrieved from the last published version.

Sample request:

```
<m:workflowProcessInstanceStart xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <publishedProcessKey>String</publishedProcessKey>
 <documentation>
  <locale>Locale</locale>
  <label>String</label>
  <description>String</description>
 </documentation>
</m:workflowProcessInstanceStart>
```

with:

Element	Description	Required
publishedProcessKey	Identifier of the workflow launcher.	Yes
documentation	See the description under Common parameters [p 1100].	No
parameters	*Deprecated since version 5.7.0* See the description under Common parameters [p 1107].	No

Sample response:

```
<m:workflowProcessInstanceStart_Response xmlns:m="urn:ebx-schemas:dataservices_1.0">
```

```
 <processInstanceKey>String</processInstanceKey>
</m:workflowProcessInstanceStart_Response>
```

with:

Element	Description	Required
processInstanceId	*Deprecated since version 5.6.1* This parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No
processInstanceKey	Workflow identifier.	No

Resume a workflow

Resume a workflow in a wait step from a resume identifier. It is possible to define specific input parameters (with name and optional value).

Sample request:

```
<m:workflowProcessInstanceResume xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <resumeId>String</resumeId>
</m:workflowProcessInstanceResume>
```

with:

Element	Description	Required
resumeId	Resume identifier of the waiting task.	Yes
parameters	*Deprecated since version 5.7.0* See the description under Common parameters [p 1107].	No

Sample response:

```
<m:workflowProcessInstanceResume_Response xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <processInstanceKey>String</processInstanceKey>
</m:workflowProcessInstanceResume_Response>
```

with:

Element	Description	Required
status	'00' indicates that the operation has been executed successfully. '20' indicates that the workflow has not been found. '21' indicates that the event has already been received.	Yes
processInstanceKey	Identifier of the workflow. This parameter is returned if the operation has been executed successfully.	No

End a workflow

End a workflow from its identifier.

Sample request:

```
<m:workflowProcessInstanceEnd xmlns:m="urn:ebx-schemas:dataservices_1.0">
```

```
 <processInstanceKey>String</processInstanceKey>
</m:workflowProcessInstanceEnd>
```

with:

Element	Description	Required
processInstanceKey	Identifier of the workflow.	Either this parameter or 'publishedProcessKey' and 'processInstanceId' parameters must be defined.
publishedProcessKey	*Deprecated since version 5.6.1* Due to a limitation this parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No
processInstanceId	*Deprecated since version 5.6.1* Due to a limitation this parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No

Sample response:

```
<m:workflowProcessInstanceEnd_Response xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</m:workflowProcessInstanceEnd_Response>
```

with:

Element	Description	Required
status	'00' indicates that the operation has been executed successfully.	Yes

172.4 Administrative services

Directory services

The services on directory provide operations on the 'Users' and 'Roles' tables of the default directory. To execute an operation related to these services, the authenticated user must be a member of the built-in role 'Administrator'.

The technical dataspace and dataset must be set to `ebx-directory`. For all SOAP operation syntaxes, see Operations generated from a data model [p 1079] for more information.

Create a user in the directory

This example of a SOAP insert request adds a user to the EBX® directory.

```
<m:insert_user xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>ebx-directory</branch>
 <instance>ebx-directory</instance>
 <data>
 <directory>
  <users>
   <login>login</login>
   <lastName>lastname</lastName>
```

```
   <firstName>firstname</firstName>
   <email>firstname.lastname@email.com</email>
   <password>***</password>
   <passwordMustChange>true</passwordMustChange>
   <builtInRoles>
    <administrator>false</administrator>
    <readOnly>false</readOnly>
   </builtInRoles>
   <comments>a comment</comments>
  </users>
 </directory>
 </data>
</m:insert_user>
```

For the insert SOAP response syntax, see insert response [p 1089] for more information.

User interface operations

See Application locking [p 676] for more information.

Parameters for operations on the user interface are as follows:

Element	Description	Required
closedMessage	Message to be displayed to users when the user interface is closed to access.	No

Close user interface request

The close operation removes all user sessions that are not acceptable in this mode.

```
<m:close xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <closedMessage>Access is temporarily forbidden.</closedMessage>
</m:close>
```

Close user interface response

```
<ns1:close_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:close_Response>
```

Open user interface request

```
<m:open xmlns:m="urn:ebx-schemas:dataservices_1.0"/>
```

Open user interface response

```
<ns1:open_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:open_Response>
```

System information operation

This operation returns the EBX® system information. The information returned is the same as the information contained in the log header `kernel.log` or in the UI tab 'Administration' > 'System Information'. The response contains several keys, labels, and values representing the configuration and status of EBX®. To execute this operation, the authenticated user must be a member of the built-in role 'Administrator'.

Parameters

The following parameter is applicable.

Parameter	Description	Required
`details`	Defines attributes that must be applied to response messages. The attribute `locale` (default: EBX® default `locale`) defines the language in which the system item messages must be returned.	No, but if specified, the `locale` attribute must be provided.

System information request

This SOAP request will return all EBX® instance's system information and format them using "`en_US`" `locale`.

```
<m:systemInformation xmlns:m="urn:ebx-schemas:dataservices_1.0">
    <details locale="en_US" />
</m:systemInformation>
```

System information response

```
<ns1:systemInformation_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
    <bootInfoEBX>
        <label>String</label>
        <infoItem>
            <key>String</key>
            <label>String</label>
            <content>String</content>
            <content>String</content>
            ...
        </infoItem>
        ...
    </bootInfoEBX>
    <repositoryInfo>
        <label>String</label>
        <infoItem>
            <key>String</key>
            <label>String</label>
            <content>String</content>
            <content>String</content>
            ...
        </infoItem>
        ...
    </repositoryInfo>
    <bootInfoVM>
        <label>String</label>
        <infoItem>
            <key>String</key>
            <label>String</label>
            <content>String</content>
            ...
        </infoItem>
        ...
    </bootInfoVM>
</ns1:systemInformation_Response>
```

REST data services

CHAPTER 173

Introduction

This chapter contains the following topics:

1. Overview
2. Activation and configuration
3. Interactions
4. Security
5. Monitoring
6. SOAP and REST comparative
7. Limitations

173.1 Overview

REST data services allow external systems to interact with data governed in the TIBCO EBX® repository using the RESTful built-in services.

The request and response syntax for built-in services are described in the chapter Built-in RESTful services [p 1125].

Built-in REST data services allow performing operations such as:

- Selecting, inserting, updating, deleting, or counting records
- Selecting or counting history records
- Selecting, updating, or counting dataset values
- Selecting or updating dataspace or snapshot information
- Selecting children dataspaces or snapshots
- Creating, merging, or closing a dataspace
- Creating or closing a snapshot
- Administrative operations to manage access to the UI or to system information

> **Note**
>
> See SOAP and REST comparative [p 1122].

173.2 Activation and configuration

REST and SOAP Data services are activated by deploying the `ebx-dataservices` web application along with the other EBX® modules. See Jakarta EE deployment [p 533] for more information.

In case of specific deployment, for example using reverse-proxy mode, see URLs computing [p 590] for more information.

Only HTTP(S) protocols are supported.

173.3 Interactions

Body media types

Input and output messages must be *exclusively* in JSON format with UTF-8 encoding for REST built-in. However, input message can also be used to pass HTML form data.

See Content-Type [p 1129] for more information.

Tracking information

Depending on the data services operation being called, it may be possible to specify session-tracking information.

- Example for a RESTful operation, the extended JSON format request contains:

```
{
 "procedureContext":               // JSON Object (optional)
 {
  "trackingInformation": "String" // JSON String (optional)
 },
 ...
}
```

For more information, see `Session.getTrackingInfo`[API] in the Java API.

Session parameters

Depending on the data services operation being called, it is possible to specify session input parameters. They are defined in the request body.

Input parameters are available on custom Java components with a session object, such as: triggers, access rules, `custom` web services. They are also available on data workflow operations.

- Example for a RESTful operation, the extended JSON format request contains:

```
{
 "procedureContext":                // JSON Object (optional)
 {
  "trackingInformation": "String", // JSON String (optional)
  "inputParameters":                // JSON Array (optional)
  [
   // JSON Object for each parameter
   {
    "name": "String",            // JSON String (required)
    "value": "String"            // JSON String (optional)
   },
   ...
  ]
 },
 ...
}
```

For more information, see `Session.getInputParameterValue`[API] in the Java API.

Session channel

The session channel allows to filter what can be selected or modified, from the EBX® repository, when using a REST built-in or REST toolkit service. The filter is based on table, group or field configuration where the visibility is defined through the data model, by specifying a default view [p 963].

It can be specified through the query parameter ebx-channel [p 1130]. It's available values are:

- `dataServices`
- `ui`

The filter behavior is described by this combinatorial:

Data services channel

XML element	Value	Schema node type	View	Behaviour
`<hiddenInDataservices>`	`true`	`field node`	**Default tabular view**	Hidden for content and not sortable
			CustomView (tabular or hierarchical)	Hidden for meta, content, filter and sort
			Default form record	Hidden for meta and content
			Default form record field	Not found

User Interface channel

XML element	Value	Schema node type	View	Behaviour
<hidden>	true	table node	**Tree view**	hidden for meta and content
			Default tabular view	not found
			CustomView (tabular or hierarchical)	forbidden
			Default form record	not found
			Default form record field	
			Custom form record	
		field node	**Default tabular view**	hidden for content and not sortable
			Default form record	hidden for content
			Default form record field	not found
<hiddenInViews>	true	field node	**CustomView (tabular or hierarchical)**	hidden for meta, content, filter and sort
			Custom form record	hidden for meta and content
<hiddenInSearch>	true	field node	**Default tabular view**	not filterable
			CustomView (tabular or hierarchical)	
			Default form record	
			Default form record field	
			Custom form record	
	textSearchOnly		**Default tabular view**	not filterable except for text search
			CustomView (tabular or hierarchical)	

XML element	Value	Schema node type	View	Behaviour
			Default form record	
			Default form record field	
			Custom form record	

> **Note**
>
> The above field nodes can only be under table nodes.

Procedure context

Depending on the data services operation being called, it is possible to overwrite the default procedure context configuration. They are defined in the request body and are applied within the built-in operation.

Procedure context can be applied to custom REST Toolkit services.

- Example for a RESTful operation, the JSON request body contains:

```
{
 "procedureContext":            // JSON Object (optional)
 {
  "commitThreshold": Integer   // JSON Number (optional)
 },
 ...
}
```

For more information, see `ProcedureContext.setCommitThreshold`[API], `SessionContext.getProcedureUtility`[API] and `ProcedureUtility.execute`[API] in the Java API.

Exception handling

When an error occurs, a JSON exception response is returned to the caller. In most cases the exception takes JSON format like in the REST Toolkit.

See also *REST Toolkit exception handling [p 1295]*

```
{
  "code": 999,                      // JSON Number, HTTP error code or EBX® error code
                                    // in case of HTTP error 422 (optional)
  "errors": [                       // Additional messages to give details (optional).
    {
      "message": "Message 1"        // JSON String
    },
    {
      "message": "Message 2"
    }
  ]
}
```

If there are errors when inserting or updating multiple records [p 1152], an error report is returned.

```
{
  "code": 999,              // JSON Number, HTTP status code
  "errors": [
    {
      "level": "...",       // JSON String, severity level (optional)
      "rowIndex": 999,      // JSON Number, request row index (optional)
      "userCode": "...",    // JSON String, user code (optional)
      "message": "...",     // JSON String, message
```

```
      "blocksCommit": "...", // JSON String, Type of blocking constraints (optional)
      "details": "...",      // JSON String, URL (optional)
      "pathInRecord": "...", // JSON String, Path in record (optional)
      "pathInDataset": "..." // JSON String, Path in dataset (optional)
    }
  ]
}
```

The response contains an HTTP status code and a table of errors. The severity level of each error is specified by a character, with one of the possible values (`fatal`, `error`, `warning`, `info`).

The HTTP error `422` *(Unprocessable entity)* corresponds to a functional error. It contains a user code under the `userCode` key and is a JSON `String` type.

See also

HTTP codes *[p 1132]*

`Severity.getLabel`[API]

173.4 Security

Authentication

Authentication is mandatory to access to the content. Several authentication methods are available and described below. The descriptions are ordered by priority (EBX® applies the highest priority authentication method first).

- 'Token Authentication Scheme' method is based on the HTTP-Header `Authorization`, as described in RFC 2617.

```
> Authorization: <tokenType> <accessToken>
```

  For more information on this authentication scheme, see Token authentication operations [p 1135].

  See also *HTTP Authorization header policy* *[p 590]*

- 'Basic Authentication Scheme' method is based on the HTTP-Header `Authorization` in base64 encoding, as described in RFC 2617 (Basic Authentication Scheme).

```
If the user agent wishes to send the userid "Alibaba" and password "open sesame",
it will use the following header field:
> Authorization: Basic QWxpYmFiYTpvcGVuIHNlc2FtZQ==
```

  > **Note**
  >
  > The `WWW-Authenticate` `[p 1131]` header can be valued with this method.

  See also *HTTP Authorization header policy* *[p 590]*

- 'Bearer Authentication Scheme' method is similar to 'Token Authentication Scheme' but works with bearer tokens that may be issued from a third party provider.

```
> Authorization: bearer <accessToken>
```

  For more information on this authentication scheme, see Bearer authentication configuration. [p 580]

- 'Standard Authentication Scheme' is based on the HTTP `Request`. User and password are extracted from request parameters. For more information on request parameters, see Parameters [p 1074] section.

For more information on this authentication scheme, see `Directory.authenticateUserFromLoginPassword`[API].

- The 'REST Forward Authentication Scheme' is used only when calling a REST service from a user service [p 1024], that reuses the current authenticated session.

  For more information, see Implementing a user service [p 1035] making a call to REST data services [p 1042].

- 'Specific authentication Scheme' is based on the HTTP `Request`. For example, an implementation can extract a password-digest or a ticket from the HTTP `Request`. See `Directory.authenticateUserFromHttpRequest`[API] for more information.

- 'Anonymous authentication Scheme' is used only to access the REST services handling the authentication operations. The credentials acquisition, password changes, etc. implies that the user cannot be known yet.

Global permissions

Global access permissions can be independently defined for the REST built-in and REST OpenAPI services. See Global permissions [p 671] for more information.

Lookup mechanism

Because EBX® offers several authentication methods, a lookup mechanism based on conditions was set to know which method should be applied for a given request. The method application conditions are evaluated according to the authentication scheme priority. If the conditions are not satisfied, the server evaluates the next method. The following table presents the available authentication methods

for each supported protocol and their application conditions. They are ordered from the highest priority to the lowest.

Operation / Protocol	Authentication methods and application conditions
REST / HTTP	Token [p 1119] • The HTTP request must hold an `Authorization` header. • `Authorization` header value must start with the word `EBX`. • No `login` is provided in the URL parameters. Basic [p 1119] • The HTTP request must hold an `Authorization` header. • `Authorization` header value must start with the word `Basic`. • No `login` is provided in the URL parameters. Bearer Token [p 1119] • The HTTP request must hold an `Authorization` header. • `Authorization` header value must start with the word `Bearer`. • No `login` is provided in the URL parameters. Standard [p 1119] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters. Rest forward [p 1120] • The HTTP request must not contain an `Authorization` header. • No `login` is provided in the URL parameters. Specific [p 1120] • The HTTP request must not satisfy the conditions of the previous authentication methods. Anonymous [p 1120] • None of the previous authentication methods can be applied. • The requested REST service is handling an authentication operation.

In case of multiple authentication methods present in the same request, EBX® will return an HTTP code `401 Unauthorized`.

173.5 Monitoring

Data service events can be monitored through the log category `ebx.dataServices`, as declared in the EBX® main configuration file. For example, `ebx.log4j.category.log.dataServices= INFO, ebxFile:dataservices`.

See also

Configuring the EBX® logs [p 582]

TIBCO EBX® main configuration file [p 571]

173.6 **SOAP and REST comparative**

Operations	**SOAP**	**REST**
Data		
Select metamodel		X
Select or count records (with filter and/or view publication)	X	X
Select records with association values (first level)	X	
Select business objects		X
Update or delete records (with filter and/or view publication)		X
Selector for possible enumeration values (with filter)		X
Prepare for creation or duplication		X
Insert, update or delete records	X	X
Select or count history records (with filter and/or view publication)		X
Select node values from dataset	X	X
Update node value from dataset		X
Get table or dataset changes between dataspaces or snapshots	X	
Refresh a replication unit	X	
Get credentials for records	X	
Generate service contract	WSDL	OpenAPI
Views		
Look up published table views		X
Datasets		
Select dataset information		X
Select root or child datasets		X

Operations	SOAP	REST
Dataspaces		
Select dataspace or snapshot information		X
Select root or children dataspaces, or select snapshots		X
Create, close, merge a dataspace	X	X
Create, close a snapshot	X	X
Validate a dataspace or a snapshot	X	
Validate a dataset	X	
Locking, unlocking a dataspace	X	X
Workflow		
Start, resume or end a workflow	X	
Administration		
Manage the default directory content 'Users', 'Roles'... tables.	X	X
Open, close the user interface	X	X
Select, insert, update, delete operations for administration dataset		X
Select the system information	X	X
Staging API		X
Health operations		X
Other		
Develop custom services from the Java API (*)		X
Generate custom service contracts (*)		OpenAPI

(*) See REST Toolkit [p 1285] for more information.

173.7 **Limitations**

Date, time & dateTime format

Data services only support the following date and time formats:

Type	Format	Example
xs:date	yyyy-MM-dd	2007-12-31
xs:time	HH:mm:ss or HH:mm:ss.SSS	11:55:00
xs:dateTime	yyyy-MM-ddTHH:mm:ss or yyyy-MM-ddTHH:mm:ss.SSS	2007-12-31T11:55:00

JMS

- JMS protocol is not supported.

CHAPTER 174

Built-in RESTful services

This chapter contains the following topics:

1. Introduction
2. Request
3. Response
4. Administration operations
5. Token authentication operations
6. Data operations
7. Form data operations
8. Dataset operations
9. Dataspace operations
10. Health operations
11. OpenAPI operations
12. Staging operations
13. Data model operations
14. Limitations

174.1 Introduction

The architecture used is called ROA (Resource-Oriented Architecture), it can be an alternative to SOA (Service-Oriented Architecture). The chosen resources are readable and/or writable by third-party systems, according to the request content.

The HATEOAS approach of the built-in RESTful services also allows for an intuitive and straightforward navigation, which implies that the data details could be obtained through a link.

> **Note**
>
> All operations are stateless.

174.2 Request

This chapter describes the elements to use in order to build a conform REST request, such as: the HTTP method, the URL format, the header fields and the message body.

See also

Interactions *[p 1115]*

Security *[p 1119]*

HTTP method

Considered HTTP methods for built-in RESTful services, are:

- `GET`: used to select master data defined in the URL (the size limit of the URL depends on the application server and on the browser; it must be lower than or equal to 2kB).
- `POST`: used to insert records in a table, or to select the master data defined in the URL (the size limit is 2MB or more, depending on the application server. Each parameter is limited to a value containing 1024 characters).
- `PUT`: used to update the master data defined in the URL.
- `DELETE`: used to delete either the record defined in the URL, or multiple records defined by the table URL and the record keys in the message body.

URL

REST URL contains:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/{categoryVersion}/
{specificPath}[:{extendedAction}]?{queryParameters}
```

Where:

- `<ebx-dataservices>` corresponds to the 'ebx-dataservices.war' web application's path. The path is composed by multiple, or none, URI segments followed by the web application's name.
- `{category}` corresponds to the operation category [p 1127].
- `{categoryVersion}` corresponds to the category version: current value is `v1`.
- `{specificPath}` corresponds to a specific path inside the category.
- `{extendedAction}` corresponds to the extended action name (optional).
- `{queryParameters}` corresponds to common [p 1130] or dedicated operation parameters passed in the URL.

Operation category

Specializes the operation; it is added in the path of the URL in `{category}`, and takes one of the following values:

`admin`	Administration operations reserved to administrators. See Administration operations [p 1133] for more information.
`auth`	Manages token authentication method. See Token authentication operations [p 1135] and Token Authentication Scheme [p 1119] for more information.
`data` or **`data-compact`**	Lists dataset content; requests a table, a record or a field record content, including modified operations on dataset node, table, record and record field. A compact format is available, to ease interaction in simple use cases. Manages dataspace and snapshot life cycle. See Data operations [p 1138], Dataspace operations [p 1174] and Compact format limitations [p 1189] for more information.
`data-bo` or **`data-compact-bo`**	Requests a table, a record, or a field record content enables navigation through defined business objects [p 84]. A compact format is available to ease interaction in simple use cases. Deletes a record with its related records under other tables defined on business object [p 84] relationships. See Data operations [p 1138] for more information.
`form-data` or **`form-data-compact`**	Validates incoming data and generates a report before inserting or updating a dataset node, record, or record field. A compact format is also available to simplify interaction for straightforward use cases. See Form data operations [p 1167] and Compact format limitations [p 1189] for more information.
`health`	Server health information returns information for a monitoring tool or container probes. See Health operations [p 1182] for more information.
`history`	Lists the content of a history dataset; requests a history table, a history of a record, or a history record. See Data operations [p 1138] for more information.

See also *History* *[p 461]*

`api`	Generates the OpenAPI document of the selected resource. See OpenAPI operations [p 1184] for more information.
`datamodel`	Manages the publication of data models defined in the Data Model Assistant. See Data model operations [p 1188] for more information. **See also** *Introduction to data models* *[p 44]* *Publishing a data model* *[p 99]*

Header fields

These header field definitions are used by TIBCO EBX®.

`Accept`	Used to specify (by order of preference) content types to be used in the response: the first supported one will be chosen and specified in the response header `Content-Type`. Currently, the only supported one is `application/json`. If none is supported, the result depends on property `ebx.dataservices.rest.request.checkAccept`: • If `true`, an HTTP error response is returned with code `406`. • If `false`, the response is returned with the default content type, that is `application/json`. **See also** *Configuring data services* *[p 587]*
`Accept-Language`	Used to specify the preferred locale for the response. The supported locales are defined in the schema model. If none of the preferred locales is supported, the default locale for the current model is used.
`Authorization`	Supported authentication schemes include 'Basic Authentication Scheme' and 'Token Authentication Scheme'; if another scheme is used, the request is rejected. **See also** *Authentication* *[p 1119]*
`Content-Type`	Used to specify the request body media type. The supported types are `application/json` and `application/x-www-form-urlencoded`. If the request value is not supported, then an HTTP error message is returned with the code `415`*(Unsupported media type)*. **See also** *Configuring data services* *[p 587]*
`X-Requested-With`	If present and in case of authentication failure, prevents from adding the `WWW-Authenticate` header in the response. **See also** *Response header `WWW-Authenticate`* *[p 1131]*

See RFC2616 for more information about HTTP Header Field Definitions.

Common parameters

These optional parameters are available for all data service operations.

Parameter	Description
`disableRedirectionToLastBroadcast`	This parameter only has impact in a D3 architecture. If `true`, access to a delivery dataspace on a D3 primary node is not redirected to the last broadcast snapshot. Otherwise, access to such a dataspace is always redirected to the last broadcast snapshot. If the specified dataspace is not a delivery dataspace on a D3 primary node, this parameter is ignored. `Boolean` type value. If this parameter is not present, the default is `false` (redirection to a D3 master enabled), unless the configuration property ebx.dataservices.disableRedirectionToLastBroadcast.default [p 587] has been set. **See also** *Primary node* [p 753]
`ebx-indent` `indent` (*Deprecated since 6.0.0*)	Specifies whether the response should be indented, to be easier to read for a human. `Boolean` type, default value is `false`.
`ebx-channel`	Specifies the session channel [p 1116]. `String` type, possible values are: • `dataServices` • `ui` The default value is `dataServices`. **See also** `Constants.Data.PARAM_EBX_CHANNEL` API

Message body

It contains the request data using the JSON format, see Extended JSON request body [p 1193] and Compact JSON request body [p 1223].

> **Note**
>
> Requests may define a message body only when using `POST` or `PUT` HTTP methods.

174.3 Response

This chapter describes the responses returned by built-in RESTful services.

- See Exception handling [p 1118] for details on standard error handling (where the HTTP code is greater than or equal to `300`).

Header fields

These header field definitions are used by EBX®.

`Content-Language`	Indicates the locale used in the response for labels and descriptions.
`Content-Type`	Indicates the response body content type.
`Location`	If a new record has been successfully inserted, the query URL for this record is returned by this field.
`WWW-Authenticate`	This header field is added to the HTTP response when authentication fails with the 401 *(Unauthorized)* HTTP code. Its value consists of a list with at least one authentication method applicable to the request URI. It is present if and only if the following conditions are verified: • the 'Basic Authentication Scheme' method is enabled and • the `X-Requested-With` HTTP header is not present. If the client is able to interpret the authentication method, it is possible to resubmit the request providing the appropriate credentials. The administration property ebx.dataservices.rest.auth.tryBasicAuthentication [p 587] must be set to `true`. **See also** *Request header `X-Requested-With`* *[p 1129]* *Authentication* *[p 1119]*

HTTP codes

HTTP code	Description
`200` *(OK)*	The request has been successfully handled.
`201` *(Created)*	A new record has been created, in this case, the header field `Location` is returned with its resource URL.
`204` *(No content)*	The request has been successfully handled but no response body is returned.
`206` *(Partial Content)*	The request has been successfully handled but a consistent partial response body is returned.
`400` *(Bad request)*	The request URL or body is not well-formed or contains invalid content.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	Permission was denied to read or modify the specified resource for the authenticated user. This error is also returned when the user: • is not allowed to modify a field mentioned in the request message body. • is not allowed to access the REST connector. For more details, see Global permissions [p 1120].
`404` *(Not found)*	The resource specified in the URL cannot be found.
`406` *(Not acceptable)*	Content type defined in the request's `Accept` parameter is not supported. This error can be returned only if the EBX® property `ebx.rest.request.checkAccept` is set to `true`.
`409` *(Conflict)*	A concurrent modification has occurred. **See also** *Optimistic locking [p 1163]*
`415` *(Unsupported media type)*	The request content is not supported, the request header value `Content-Type` is not supported by the operation.
`422` *(Unprocessable entity)*	The new resource's content cannot be accepted for semantic reasons.
`500` *(Internal error)*	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

Message body

The response body content's format depends on the HTTP code value:

- HTTP codes from `200` included to `300` excluded: the content format depends on the associated request (Extended JSON [p 1196] and Compact JSON [p 1226] samples).

With the exception of code `204` *(No content)*.

- HTTP codes greater than or equal to `300`: the content describes the error. See JSON [p 1118] for details on the format.

174.4 Administration operations

Administration operations are related to:

- the administration category.
- the administration dataspaces accessible through the `data` category.

> **Note**
>
> administration category and administration dataspaces can only be used by administrators.

Directory operations

The EBX® default directory configuration is manageable with built-in RESTful services. The users and roles tables, the mailing lists and other objects are concerned. For more information, see Users and roles directory [p 703].

> **Note**
>
> Triggers are present on the directory's tables, ensuring the data consistency.

The URL format is:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/Bebx-directory/ebx-directory
```

Directory configuration operations

The EBX® default directory configuration is manageable like dataset nodes. It can be accessed and modified through the `data` category operations. Each field is self-described when metamodel is requested.

See select [p 1139] and update [p 1156] operations for more information.

Mailing lists operations

There are two default mailing lists that can be configured in the EBX® directory: one for everybody and one for administrators. These lists can be handled like dataset nodes through the `data` category operations.

See select [p 1139] and update [p 1156] operations for more information.

Directory users operations

Users can be manipulated like records of the `data` category using the operations of the latter. For security purposes, an administrator cannot delete himself. The user's salutation must be chosen among the ones available in the 'salutations' table.

See select [p 1139], update [p 1156], insert [p 1152] and delete [p 1158] operations for more information.

Directory roles operations

Roles are records of the `data` category and can be managed with its operations. EBX® roles are assigned to users through the 'usersRoles' association table. 'usersRoles' is automatically fed when the directory is administered through the user interface. However, it is not the case through data services and role assignments require manual operations. Roles inclusions are specified in the 'rolesInclusions' association table. As for the 'usersRoles' table, the management of roles inclusions requires manual operations. Each table is self-descriptive when metamodel is requested.

See select [p 1139], update [p 1156], insert [p 1152] and delete [p 1158] operations for more information.

User interface operations

The EBX® user interface can be opened or closed to users for maintenance needs. Handled information is similar to what is contained in the UI tab 'Administration' > 'User interface configuration' > 'Advanced perspective' > 'Graphical interface configuration' > 'Application locking'.

The URL format is:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/Bebx-manager/ebx-manager/
domain/toolStatus
```

See also *Application locking [p 676]*

Retrieve user interface state

User interface status and the unavailability message are accessible like dataset nodes.

See Select operation [p 1139] and the Extended JSON [p 1205] or Compact JSON [p 1230] example, for more information.

Open or close user interface

User interface status and the unavailability message can be modified like dataset nodes using the update operation. To open the user interface set the content of `toolStatus` to `true`, or to `false` to close it.

See Update operation [p 1156] and the Extended JSON [p 1196] or Compact JSON [p 1225] examples, for more information.

System information operation

This operation returns system information on the EBX® server. This is accepted for `GET` and `POST` HTTP methods. Warning: no update will be possible in the `POST` HTTP method because the request body is ignored. The information returned is the same as the information contained in the log header `kernel.log` or in the UI tab 'Administration' > 'System Information'. The response contains several keys, labels, and values representing the configuration and status of EBX®. The mode of representation of the response may be flat or hierarchical.

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/admin/v1/systemInformation
```

See also

TIBCO EBX® main configuration file [p 571]

Repository administration [p 660]

Parameters

The following parameter is applicable.

Parameter	Description
`systemInformationMode`	Specifies the returned mode: • `flat`: A flat representation under the following information groups: `bootInfoEBX`, `repositoryInfo` and `bootInfoVM`. • `hierarchical`: A hierarchical representation. `String` type, default value is `flat`.

HTTP codes

HTTP code	Description
`200` *(OK)*	The system information was successfully returned.
`400` *(Bad request)*	The request is not correct, it contains one of the following errors: • the HTTP method is not `GET` nor `POST`, • the HTTP parameter `systemInformationMode` is not correct, • the operation is not supported. • the request path is invalid.
`403` *(Forbidden)*	The user is not an administrator.

Response body

It is returned, if and only if, the HTTP code is `200 (OK)`. The content structure depends on the provided parameter `systemInformationMode` or its default value.

See the JSON [p 1244] example of the flat representation.

See the JSON [p 1244] example of the hierarchical representation.

174.5 Token authentication operations

These operations allow to create or revoke an authentication token. Authentication tokens have a timeout period. If a token is not used to access the EBX® server within this period, it will automatically be revoked. This timeout period is refreshed on each access to EBX® server.

> **Note**
>
> The token timeout is modifiable through the administration property ebx.dataservices.rest.auth.token.timeout [p 587] (the default value is 30 minutes).

> **Note**
>
> Tokens can only be generated for users defined in the main directory. For more information about directories, see Users and roles directory [p 703].

Create token operation

This operation requires using the `POST` HTTP method with a request containing the user's credentials and, optionally, session parameters [p 1115].

The URL format is:

`http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/token:create`

Message body

A message body must be defined in the HTTP request. It necessarily contains one of the following set of data:

- A `login` and a `password` value. Both JSON attributes are mandatory and of `String` types.

  See `Directory.authenticateUserFromLoginPassword`[API] for more information.

- The `specific` JSON attribute set to `true`. When activated, this flag allows to performed a user authentication against the whole HTTP request. Warning, even if `login` and `password` attributes are defined in the JSON request's body, setting `specific` to `true` lead to a specific user authentication.

  See `Directory.authenticateUserFromHttpRequest`[API] for more information.

See the JSON [p 1243] examples of a token creation request.

HTTP codes

HTTP code	Description
`200` *(OK)*	The token was successfully created.
`400` *(Bad request)*	For one of the following reasons: • the syntax is not correct, • the HTTP method is not `POST`, • the operation is not supported.
`401` *(Unauthorized)*	For one of the following reasons: • The `login` and/or `password` is/are incorrect. • Authentication data for other methods is defined.
`422` *(Unprocessable entity)*	For one of the following reasons: • `PasswordMustChange`: The password must be changed (only available with the default directory). See Change password operation [p 1137]. • `RestrictedAccess`: User access is closed for maintenance or other actions (reserved to administrators).

Response body

If the HTTP code is `200 (OK)`, the body holds the token value and its type.

See the JSON [p 1245] example of a token creation response.

The token can later be used to authenticate a user by setting the HTTP-Header `Authorization` accordingly.

See also *'Token authentication Scheme' method [p 1119]*

Change password operation

This operation modifies the password of an existing user account. It can be used in an authenticated context: `login` parameter, if present, is checked against the current session or taken from it, if absent. It could also be used in an unauthenticated context, for example when the Create token operation [p 1136] aborts with the HTTP code `422` *(Unprocessable entity)* with reason: `PasswordMustChange`.

It requires the use of:

- the EBX® default directory
- the `POST` HTTP method
- the message body containing the structure specified below

The URL format is:

`http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/user:changePassword`

Message body

The message body must be defined in the request. It necessarily contains a `password` and a `passwordNew`, the `login` is optional (all are `String`).

See the JSON [p 1243] example of a password change and token creation request.

HTTP codes

HTTP code	Description
`204` *(No content)*	The password has been changed.
`400` *(Bad request)*	For one of the following reasons: • the EBX® default directory is required, • the syntax is not correct, • the HTTP method is not `POST`, • the provided `login` and the user's session one mismatch • the operation is not supported.
`401` *(Unauthorized)*	For the following reason: • The `login` and/or `password` is/are incorrect.
`422` *(Unprocessable entity)*	For one of the following reasons: • `PasswordChangeAbort`: `passwordNew` is empty. • `PasswordChangeAbort`: `passwordNew` is equal to `password`. • `PasswordChangeAbort`: `password` is incorrect. • `RestrictedAccess`: User access is closed for maintenance or other actions (reserved to administrators).

Response body

If HTTP code `204 (No content)` is returned, then the password has been modified.

Revoke token operation

This operation requires using the `POST` HTTP method. No message body is needed.

The URL format is:

`http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/token:revoke`

Header fields

`Authorization`	This field is required, `tokenType` and `accessToken` fields must have the values returned from the "token create" operation. `> Authorization: <tokenType> <accessToken>`

HTTP codes

HTTP code	Description
`204` *(No content)*	The token has been revoked successfully.
`400` *(Bad request)*	For one of the following reasons: • the configuration is not activated, • the syntax is incorrect, • the HTTP method is not `POST`, • the operation is not supported.
`401` *(Unauthorized)*	Authentication has failed.

174.6 Data operations

The `data` category operations concern the datasets, the dataset fields, tables, records or record fields.

The `data-compact` category operations concern the dataset fields, tables, records or record fields.

The `data-bo` category operations concern `data` category operations with features to navigate through business objects [p 84].

The `data-compact-bo` category operations concern `data-compact` category operations with features to navigate through business objects [p 84].

The `history` category operations concern historized content from datasets, tables, records or record fields.

The `form-data` category operations concern the dataset fields, records or record fields when constraints must remain valid on content creation or update.

The `form-data-compact` category operations concern the dataset fields, records or record fields when constraints must remain valid on content creation or update.

See Form data operations [p 1167] for more information.

Select operation

Select operation returns hierarchical content. This operation may use one of the following methods:

- `GET` HTTP method,
- `POST` HTTP method without message body or
- `POST` HTTP method with a message body, with `:select` URL extended action and optionally session parameters [p 1115].

URL formats are:

- **Dataset tree**, depending on operation category [p 1127]:

  The `data`, or `data-compact`, category returns the hierarchy of the selected dataset which includes group and table nodes.

  The `history` category returns the hierarchy of the selected history dataset which includes the pruned groups for history table nodes only.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}
  ```

  > **Note**
  >
  > Terminal nodes and sub-nodes are not included.

- **Dataset node**: the `data`, or `data-compact`, category returns the terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}[:select]
  ```

  > **Note**
  >
  > Not applicable with the `history` category.

- **Table**, depending on operation category [p 1127]:

  the `data`, or `data-compact`, category returns the table content and/or metamodel, current page records and URLs for pagination.

  The `history` category returns the history table content and/or metamodel, current page records and URLs for pagination.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}[:select]
  ```

  > **See also**
  >
  > *Count operation [p 1160]*
  >
  > *Look up table views operation [p 1166]*

- **Record**, depending on operation category [p 1127]:

  the `data`, or `data-compact`, category returns the record content and/or metamodel.

  The `history` category returns history record content and/or metamodel.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}/{encodedPrimaryKey}[:select]
  ```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
{dataset}/{pathInDataset}[:select]?primaryKey={xpathExpression}
```

> **Note**
>
> The record access by the primary key (`primaryKey` parameter) is limited to its root node. It is recommended to use the encoded primary key, available in the `details` field in order to override this limitation. Similarly, for a history record, use the encoded primary key, available in the `historyDetails` field.

- **Field**, depending on operation category [p 1127]:

  the `data`, or `data-compact`, category returns the field record content where structure depends on its type.

  The `history` category returns the field history record content where structure depends on its type.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}[:select]
  ```

  > **Note**
  >
  > The field must be either an association node, a selection node, a terminal node or above.

Where:

- `{category}` corresponds to the operation category [p 1127] (possible values are: `data`, `data-compact`, `data-bo`, `data-bo-compact` or `history`).
- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the dataset node, that can be a group node or a table node.
- `{encodedPrimaryKey}` corresponds to the percent encoded representation of the primary key (see RFC-3986 Uniform Resource Identifier).
- `{xpathExpression}` corresponds to the record primary key, using the XPath expression.
- `{pathInRecord}` corresponds to the path starting from the table node.
- `:select` extended action is required when POST HTTP method is used with a body message.

Business object [p 84] **categories** (`data-bo` and `data-compact-bo`) enrich the responses of `Table`, `Record`, and `Field` requests by adding data from other tables.

Parameters

The following parameters are applicable to the select operation.

Parameter	Description
`includeContent`	Includes the `content` field with the content corresponding to the selection. `Boolean` type, default value is `true`.
`includeDetails`	Includes the `details` field in the metamodel and in the content, for each indirect reachable resource. The returned value corresponds to its URL resource. Type `Boolean`, default value is `true`. **See also** *includeMetamodel* [p 1142]
`includeHistory`	Includes those fields for a historized content: • The `history` property in the metamodel. The returned value corresponds to a boolean value. • The `historyDetails` property in the content and for each indirectly reachable resource. This point is coupled to the includeDetails [p 1141] parameter. The returned value corresponds to its URL resource. `Boolean` type, default value is `false`. **Note** The `includeHistory` parameter is ignored in the `history` category, the default value is `true`. **See also** *includeMetamodel* [p 1142] *includeDetails* [p 1141]
`includeLabel`	Includes the `label` field associated with each simple type `content`. Possible values are: • `yes`: the `label` is included for the foreign key, enumeration, record and selector values. • `all`: the `label` field is included, as for the `yes` value and also for the Content of simple type [p 1219]. **See also** *`SchemaNode.displayOccurrence`*[API] • `no`: the `label` field is not included (integration use case). `String` type, default value is `yes`. **Note** The `label` field is not included if it is equal to the `content` field.
`includeMeta`	*Deprecated since 6.1.0*, replaced by includeMetamodel [p 1142].
`includeMetadata`	The response contains the specified metadata. Each returned record contains additional elements called steps under the `ebx-metadata` root element.

Parameter	Description
	For instance, the 'system' step value will contain technical data. See optimistic locking [p 1163] for more information. String value (default value is empty) steps must be separated by a comma: `system, teamUp`. All steps can be returned using the `ebx-all` value. **See also** *Technical data* [p 1240]
`includeMetamodel`	Includes the `meta` field corresponding to the description of the structure returned in the `content` field. `Boolean` type, default value is `false`. **See also** *Metamodel* [p 1205] *includeHistory* [p 1141] *includeDetails* [p 1141]
`includeMergeInfo`	Includes the `merge_info` corresponding to a field of the technical data of an history transaction and has a potentially high access cost. `Boolean` type, default value is `true`. **Note** This parameter is ignored with the `data` category. **See also** *REST access to history table* [p 621]
`includeOpenApiDetails`	Includes the OpenAPI specification URL for each describable node. Type `Boolean`, default value is `false`. **Note** This query parameter is ignored if the includeDetails [p 1141] one is set to `false`. **See also** *OpenAPI operations* [p 1184] *includeDetails* [p 1141]
`includeSelector`	Includes the `selector` field in the response, for each indirect reachable resource. The returned value corresponds to its URL resource. Type `Boolean`, default value is `true`. **See also** *selector* [p 1148]
`includeSortCriteria`	Includes the `sortCriteria` field corresponding to the list of sort criteria applied. The sort criteria parameters are added by using: • sort [p 1145] • sortOnLabel [p 1146] • viewPublication [p 1147] `Boolean` type, default value is `false`.

Parameter	Description
	Example JSON [p 1213]
`includeTechnicals`	*Deprecated since 6.1.0*, replaced by `includeMetadata` [p 1141]. **Note** This parameter is ignored with the `history` category.
`includeValidation`	Includes the validation report corresponding to the selection. `Boolean` type, default value is `false`. **Note** This parameter is ignored with the `history` category. **See also** *includeDetails* [p 1141]

Table parameters

The following parameters are applicable to tables, associations and selection nodes.

Parameter	Description
`filter`	Quick search predicate or complete XPath predicate [p 441] expression defines the field values to which the request is applied. If empty, all records will be retrieved. `String` type value. **Note** The history code operation value is usable with `ebx-operationCode` path field from the `meta` section associated with this field. **See also** *sort by relevancy* [p 1146] *`includeMetadata` parameter* [p 1141] **See also** *Search predicate* [p 442]
`historyMode`	Specifies the filter context applied on table. `String` type, possible values are: • `CurrentDataSpaceOnly`: history in current dataspace • `CurrentDataSpaceAndAncestors`: history in current dataspace and ancestors • `CurrentDataSpaceAndMergedChildren`: history in current dataspace and merged children • `AllDataSpaces`: history in all dataspaces The default value is `CurrentDataSpaceOnly`. **See also** *`history`* [p 1127] **Note** This parameter is ignored with the `data` category.
`includeOcculting`	Includes the records in occulting mode. `Boolean` type, default value is `false`. **See also** *Inheritance* [p 1164]
`primaryKey`	Search a record by a primary key, using the XPath expression. The XPath predicate [p 441] expression should only contain field(s) of the primary key and all of them. Fields are separated by the operator `and`. A field is represented by one of the following possibilities according to its simple type: • For the `date`, `time` or `dateTime` types: use the `date-equal(path, value)` • For other types: indicate the `path`, the = operator and the `value`. Example with a composed primary key: `./pk1i=1 and date-equal(./pk2d,'2015-11-13')` The response will only contain the corresponding record, otherwise an error is returned. Consequently, the other table parameters are ignored (as filter [p 1144], viewPublication [p 1147], sort [p 1145], etc.)

Parameter	Description
	`String` type value.
`pageFirstRecordFilter`	*Deprecated since 5.9.0*, replaced by `pageRecordFilter`
`pageRecordFilter`	Specifies a server side built filter, for pagination, pointing to a record. This filter is **strongly** linked to the `pageAction` value and should not be modified on the client side. The filter takes the form of a record XPath predicate [p 441] expression used to figure out the pagination contexts. `String` type. **See also** *Pagination* *[p 1236]*
`pageAction`	Specifies the pagination action to perform from the identifier held by `pageRecordFilter`. `String` type, default value is `first`. The possible values are: • `first` • `previous` • `next` • `last` **See also** *`RequestPagination`*[API]
`pageSize`	Specifies the maximum number of records per page. `Integer` type, default value is based on the user preferences. If the specified value is `0`, selects with the maximum recommended page size, default is 10000. **See also** *ebx.dataservices.pagination.maxSize.default* *[p 587]* **Note** If the specified page size value exceeds the maximum page size, the maximum page size is selected.

`sort`	Specifies that the operation result will be sorted according to the specified criteria. The criteria are composed of one or more criterion, the result will be sorted by priority from the left. A criterion is composed of the field path and, optionally, the sorting order (ascending or descending, on value or on label). This parameter can be combined with:
	1. the sortOnLabel [p 1146] parameter as a new criteria added after the `sort`.
	2. the viewPublication [p 1147] parameter as a new criteria added after the `sort`.
	The value structure is as follows:
	`<path1>:<order>;...;<pathN>:<order>`
	Where:
	• `<path1>` corresponds to the field path in priority `1`.
	• `<order>` corresponds to the sorting order, with one of the following values:
	  • `asc`: ascending order on value (default),
	  • `desc`: descending order on value,
	  • `lasc`: ascending order on label,
	  • `ldesc`: descending order on label.

Parameter	Description
	`String` type, the default value orders according to the primary key fields (ascending order on value). **Note** The history code operation value is usable with the `ebx-operationCode` path field from the `meta` section associated with this field. **Note** This parameter is ignored when the sort by relevancy [p 1146] is activated. **Note** `includeMetadata` parameter [p 1141] **See also** *`Request.setSortCriteria`*[API]
`sortByRelevancy`	Specifies that the operation result will be sorted by relevancy, only if these conditions are fulfilled: • The Quick Search [p 128] is activated by specifying an `osd:search` predicate directly in the filter [p 1144] parameter. • The request is not applied on a history table or mapped table. `String` type. The possible values are: • `lasc`: ascending order on label, • `ldesc`: descending order on label. If the sort by relevancy is activated, the following parameters are ignored: sort [p 1145], sortOnLabel [p 1146], sortPriority [p 1147] and sort criteria defined through the viewPublication [p 1147]. **See also** *Search predicate [p 442]*

`sortOnLabel`	Specifies that the operation result will be sorted according to the record label. This parameter can be combined with:
	1. the sort [p 1145] parameter as a new criteria added before the `sortOnLabel`.
	2. the viewPublication [p 1147] parameter as a new criteria added after the `sortOnLabel`.
	The value structure is as follows:
	`<order>`
	Where:
	• `<order>` corresponds to the sorting order, with one of the following values:
	  • `lasc`: ascending order on label,
	  • `ldesc`: descending order on label.
	The behavior of this parameter is described in the section defaultLabel [p 915].
	`String` type value.
	
	See also *Limitation [p 1189]*
	
	Note

Parameter	Description
	This parameter is ignored when the sort by relevancy [p 1146] is activated.
`sortPriority`	Overrides the default priority of sort groups: • `sort` • `sortOnLabel` • `sortFromView` Comma separated `String` type, default value is: `sort,sortOnLabel,sortFromView`. **Note** This parameter is ignored when the sort by relevancy [p 1146] is activated.
`viewPublication`	Specifies the name of the published view. This parameter can be combined with: 1. the filter [p 1144] parameter as the logical `and` operation. 2. the sort [p 1145] parameter as a new criteria added before the `viewPublication`. 3. the sortOnLabel [p 1146] parameter as new criteria added before the `viewPublication`. The behavior of this parameter is described in the section EBX® as a Web Component [p 422]. `String` type value. **Note** The sort criteria, defined through this parameter, is ignored when the sort by relevancy [p 1146] is activated. **See also** *Look up table views operation [p 1166]*

Selector parameters

The following parameters are only applicable to fields that return an enumeration, foreign key or `osd:resource` (Example JSON [p 1220]). By default, a pagination mechanism is always enabled. Some

selector's select operations require input values. Thus, the uses of `POST` HTTP method message body [p 1194] allows to provide a record content.

Parameter	Description
`selector`	Specifies whether: • `true`: returns all possible values (includes their labels) • `false`: returns the current values for the current field. `Boolean` type, default value is `false`. **Note** This parameter is ignored with the `history` category.
`firstElementIndex`	Specifies the index of the first element returned. It must be an integer higher than or equal to `0`. `Integer` type, default value is `0`.
`pageSize`	Specifies the maximum number of elements per page. `Integer` type, default value is based on the user preferences. If the specified value is `0`, selects with the maximum recommended page size, default is 10000. **See also** *ebx.dataservices.pagination.maxSize.default* [p 587] **Note** If the specified page size value exceeds the maximum page size, the maximum page size is selected.
`selectorFilter`	Specifies the filter of the selector. `String` type value, the syntax complies with the Quick Search [p 128].

HTTP codes

HTTP code	Description
`200` *(OK)*	The selected resource is successfully retrieved.
`206` *(Partial Content)*	Partial content is returned. It happens on business object categories when the size of the response exceeds the defined limit. See `ebx.dataservices.rest.bo.maxResponseSizeInKB` and performance [p 621] for more information. **Note** The serialized records in the partial content are not truncated.
`400` *(Bad request)*	The request is incorrect. This occurs when: • The selected field in a record or a dataset is sub-terminal, • The XPath predicate of the `filter` parameter is malformed or contains unfilterable nodes. • The XPath predicate of the `primaryKey` parameter is malformed or is not a record primary key. • The sort criteria of the `sort` parameter have an invalid syntax or contain unsortable nodes. • `pageAction` parameter value is not included in allowed values, or `pageRecordFilter` is malformed or non-existent when selecting next or previous page. • `pageSize` parameter value is below 2. • The table view for the `viewPublication` parameter is either hierarchical, non-existent or non-published. • The `selector` parameter is used for a non-enumerated node, or the `firstElementIndex` is negative, higher than or equal to the number of values.
`403` *(Forbidden)*	The selected resource is hidden for the authenticated user.
`404` *(Not found)*	The selected resource is not found.

Response body

After a successful dataset, table, record or field selection, the result is returned in the response body. The content depends on the provided parameters and selected data.

Examples: Extended JSON [p 1197], Compact JSON [p 1226].

Prepare operations

Prepare for creation or duplication operations are used to create a new transient record with an initial content or with the content of a record to duplicate. A transient record corresponds to a content that is not persisted yet. Default values and fields, initialized by table triggers, are considered. Only field values with, at least, read-only access permissions are returned. These operations can optionally return metamodel to improve and assist data capture on the client side. Auto-incremented fields are only returned in the metamodel. By enabling the selector parameter, a transient record's field can be queried to the retrieve possible values of an enumerated field, a foreign key, etc. Furthermore, selector's select

operations allows to provide input record data [p 1194] to manage use cases like getting metamodel on a custom programmatic enumeration, etc.

See also `TableTrigger.handleNewContext`[API]

See also *Default value [p 63]*

After being modified on the client side, the record can be submitted to be persisted using the built-in Form insert operation [p 1167] or Insert operation [p 1152] operations.

These prepare operations may use one of the following methods:

- `GET` HTTP method,
- `POST` HTTP method without message body or
- `POST` HTTP method with a message body and optionally session parameters [p 1115].

Available URL formats are:

- **Prepare for creation**
  - **Record:**

    ```
    http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
    {dataset}/{tablePath}:prepareForCreation
    ```

  - **Record field:**

    ```
    http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
    {dataset}/{tablePath}:prepareForCreation/{pathInRecord}
    ```

- **Prepare for duplication**
  - **Record:**

    ```
    http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
    {dataset}/{tablePath}/{encodedPrimaryKey}:prepareForDuplication
    ```

  - **Record field:**

    ```
    http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
    {dataset}/{tablePath}/{encodedPrimaryKey}:prepareForDuplication/{pathInRecord}
    ```

Where:

- `{category}` corresponds to the operation category [p 1127] (possible values are: `data` or `data-compact`).
- `{dataspace}` corresponds to `B` followed by the dataspace identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{tablePath}` corresponds to the table path node.
- `{encodedPrimaryKey}` corresponds to the percent encoded representation of the primary key (see RFC-3986 Uniform Resource Identifier).
- `{pathInRecord}` corresponds to the path starting from the table node.

Parameters

The following parameters are applicable to the prepare operations:

Parameter	Description
`includeDetails`	Includes the `details` field in the metamodel and the content, for each indirect reachable resource. The returned value corresponds to its URL resource. Type `Boolean`, default value is `true`. **See also** *includeMetamodel* [p 1151]
`includeMeta`	*Deprecated since 6.1.0*, replaced by includeMetamodel [p 1151].
`includeMetamodel`	Includes the `meta` field holding the description of the structure returned in the `content` field. `Boolean` type, default value is `false`. **See also** *Metamodel* [p 1205]
`includeSelector`	Includes the `selector` field in the content, for each indirect reachable resource. The returned value corresponds to its URL resource. Type `Boolean`, default value is `true`. **See also** *selector* [p 1148]

Selector parameters

The available parameters are similar to the selector operation ones.

See also *Selector parameters* [p 1147]

HTTP codes

HTTP code	Description
`200` *(OK)*	The selected resource has been successfully retrieved.
`400` *(Bad request)*	The request is incorrect. This occurs when: • The selected field in a record is sub-terminal, • `pageSize` parameter value is below 2, or is a string different from `unbounded`. • The `selector` parameter is used for a non-enumerated node, or the `firstElementIndex` is negative, higher than or equal to the number of values.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	The selected resource is hidden from the authenticated user.
`404` *(Not found)*	The selected resource could not be found.

Response body

After a successful prepare for a creation or duplication request, the transient record is returned in the response body. The content depends on the provided parameters and selected data. However, it takes a format similar to the select operation record.

Examples: Extended JSON [p 1203], Compact JSON [p 1229].

Insert operation

Insert operation uses the `POST` HTTP method. A body message is required to specify data. This operation supports the insertion of one or more records in a single transaction. Moreover, it is also possible to update record(s) through parameterization.

- **Record**: insert a new record or modify an existing one in the selected table.
- **Record table**: insert or modify one or more records in the selected table, while securing a consistent answer. Operations are executed sequentially, in the order defined on the client side. When an error occurs during a table operation, all updates are cancelled and the client receives an error message with detailed information.

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/
{pathInDataset}[:mass]
```

Where:

- `{category}` corresponds to the operation category [p 1127] (possible values are: `data` or `data-compact`).
- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the table node.
- `:mass` extended action is required when updating a filtered set of records (see filter parameter [p 1154]).

Parameters

The following parameters are applicable with the insert operation.

Parameter	Description
`includeDetails`	Includes the `details` field in the content to access to the details of the data. The returned value corresponds to its URL resources. Type `Boolean`, the default value is `false`. **Note** Only applicable on the **record table**. **See also** *Insert operation report* [p 1239]
`includeForeignKey`	Includes the `foreignKey` field in the answer for each record. The returned value corresponds to the value of a foreign key field that was referencing this record. `Boolean` type, the default value is `false`. **Note** Only applicable on the **record table**. **See also** *Insert operation report* [p 1239]
`includeLabel`	Includes the `label` field in the answer for each record. Possible values are: • `yes`: the `label` field is included. • `no`: the `label` field is not included (use case: integration). `String` type, the default value is `no`. **Note** Only applicable on the **record table**. **See also** *Insert operation report* [p 1239]
`updateOrInsert`	Specifies the behavior when the record to insert already exists: • If `true`: the existing record is updated with new data. For a request on a **record table**, the `code` field is added to the report in order to specify if this is an insert `201` or an update `204`. • If `false` (default value): a client error is returned and the operation is aborted. `Boolean` type value.
`byDelta`	Specifies the behavior for setting value of nodes that are not defined in the request body. This is described in the Update modes [p 1241] section. `Boolean` type, the default value is `true`. **Note**

Parameter	Description
	Applicable on record in update mode and if the updateOrInsert [p 1153] parameter is `true`.
`filter`	Determines the records updated using a quick search predicate, or a complete XPath predicate [p 441] expression when a large number of records are updated. The request body must be a **record** structure with the fields to update. This parameter must use the **:mass** extended action. Otherwise, an error is returned. Use the **ebx-all** value to select all records. When the viewPublication [p 1154] query parameter is used, the scope is limited to the filtered view. `String` type value that must not be empty. Otherwise, an error is returned. **Note** When using the query parameters includeDetails [p 1153], includeLabel [p 1153] or includeForeignKey [p 1153], only the first 1000 report entries are returned in the report.
`viewPublication`	Specifies the name of the published view to be considered during a massive update. The behavior of this parameter is described in the EBX® as a Web Component [p 422] section. `String` type value.
`blockingConstraintsDisabled`	Specifies whether blocking constraints are ignored. If they are, the operation is committed regardless of the validation error created. Otherwise, the operation is aborted. `Boolean` type with a default value of `false`. See Blocking and non-blocking constraints [p 945] for more information.

Message body

The request must define a message body. The format depends on the inserted object type:

- **Record**: similar to the select operation of a record but without the record's header (example Extended JSON [p 1194], Compact JSON [p 1224]).
- **Record table**: Similar to the select operation on a table but without the pagination information (example Extended JSON [p 1195], Compact JSON [p 1224]).

See also *Inheritance [p 1164]*

HTTP codes

HTTP code	Description
`200` *(OK)*	If the request relates to a **record table**. The insert request was applied successfully, an optional report is returned in the response body.
`201` *(Created)*	If the request relates to a **record**. A new record has been created, in this case, the header field `Location` is returned with its resource URL.
`204` *(No content)*	If the request relates to a **record**. Only available if `updateOrInsert` is `true`, an existing record has been successfully updated, in this case, the header field `Location` is returned with its resource URL.
`400` *(Bad request)*	The request is incorrect. This occurs when the body message structure does not comply with what was mentioned in Message body [p 1154].
`403` *(Forbidden)*	Authenticated user is not allowed to create a record or the request body contains a read-only field.
`404` *(Not found)*	The selected resource is not found.
`409` *(Conflict)*	Concurrent modification, only available if `updateOrInsert` is `true`, the Optimistic locking [p 1163] is activated and the content has changed in the meantime, it must be reloaded before update.
`422` *(Unprocessable entity)*	The request cannot be processed. This occurs when: • A blocking validation error occurs (only available if `blockingConstraintsDisabled` is `false`). • The record cannot be inserted because a record with the same primary key already exists (only available if `updateOrInsert` is `false`). • The record cannot be inserted because the definition of the primary key is either non-existent or incomplete. • The record cannot be updated because the value of the primary key cannot be modified.

Response body

The response body format depends on the inserted object type:

- **Record**: is empty if the operation was executed successfully. The header field `Location` is returned with its URL resource.
- **Record table**: (optional) contains a table of element(s), corresponding to the insert operation report (example JSON [p 1239]). This report is automatically included in the response body, if at least one of the following options is set:
  - `includeForeignKey`
  - `includeLabel`
  - `includeDetails`

See also *Inheritance [p 1164]*

Update operation

This operation allows the modification of a single dataset or record. The `PUT` HTTP method must be used. Available URL formats are:

- **Dataset node**: modifies the values of terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}
  ```

- **Record**: modifies the content of selected record.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}/{encodedPrimaryKey}
  ```

  > **Note**
  >
  > Also available for `POST` HTTP method. In this case, the URL must point to the table and the parameter `updateOrInsert` must be set to `true`.

  > **Note**
  >
  > Use Insert operation [p 1152] with `updateOrInsert=true` and `byDelta=true` parameters to modify the content of several records.

- **Field**: update of a single field of the selected record.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
  ```

  > **Note**
  >
  > The field must be either a terminal node or above.

Where:

- `{category}` corresponds to the operation category [p 1127] (possible values are: `data` or `data-compact`).
- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the dataset node:
  - For dataset node operations, this must be any terminal node or above except table node,
  - For record and field operations, this corresponds to the table node.
- `{encodedPrimaryKey}` corresponds to the percent encoded representation of the primary key (see RFC-3986 Uniform Resource Identifier).
- `{pathInRecord}` corresponds to the path starting from the table node.

Parameters

Here are the parameters applicable with the update operation.

Parameter	Description
`blockingConstraintsDisabled`	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. `Boolean` type, default value is `false`. See Blocking and non-blocking constraints [p 945] for more information.
`byDelta`	Specifies the behavior for setting value of nodes that are not defined in the request body. This is described in the Update modes [p 1241] section. `Boolean` type, the default value is `true`.
`checkNotChangedSinceLastUpdateTime`	Timestamp in datetime format used to ensure that the record has not been modified since the last read. Also see the Optimistic locking [p 1163] section. `DateTime` type value.

Message body

The request must define a message body.

The structure is the same as for:

- the dataset node (sample Extended JSON [p 1193], Compact JSON [p 1223]),
- the record (sample Extended JSON [p 1194], Compact JSON [p 1224]),
- the record fields (sample Extended JSON [p 1195], Compact JSON [p 1224]),

depending on the updated scope, by only keeping the `content` entry.

See also *Inheritance [p 1164]*

HTTP codes

HTTP code	Description
`204` *(No content)*	The record, field or dataset node has been successfully updated.
`400` *(Bad request)*	The request is incorrect. This occurs when the body request structure does not comply.
`403` *(Forbidden)*	Authenticated user is not allowed to update the specified resource or the request body contains a read-only field.
`404` *(Not found)*	The selected resource is not found.
`409` *(Conflict)*	Concurrent modification, the Optimistic locking [p 1163] is activated and the content has changed in the meantime, it must be reloaded before the update.
`422` *(Unprocessable entity)*	The request cannot be processed. This occurs when: • A blocking validation error occurs (only available if `blockingConstraintsDisabled` is `false`). • The record cannot be updated because the value of the primary key cannot be modified.

Delete operation

The operation uses the `DELETE` HTTP method.

Two URL formats are available:

- **Record**: delete a record specified in the URL.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
  {pathInDataset}/{encodedPrimaryKey}
  ```

- **Record table**: deletes several records in the specified table, while providing a consistent answer. This mode requires a body message containing a record table. The deletions are executed sequentially, according to the order defined in the table. When an error occurs during a table operation, all deletions are cancelled and an error message is displayed with detailed information.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
  {pathInDataset}[:mass]
  ```

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the table node.
- `{encodedPrimaryKey}` corresponds to the percent encoded representation of the primary key (see RFC-3986 Uniform Resource Identifier).
- `:mass` extended action is required when deleting a filtered set of records (see filter parameter [p 1159]).

In a child dataset context, this operation modifies the `inheritanceMode` property value of the record as follows:

- A record with inheritance mode set to `inherit` or `overwrite` becomes `occult`.
- A record with inheritance mode set to `occult` becomes `inherit` if the `inheritIfInOccultingMode` operation parameter is set to `true` or is undefined, otherwise there is no change.
- A record with inheritance mode set to `root` is simply deleted.

See also *Inheritance [p 1164]*

Business object [p 84] categories (`data-bo` and `data-compact-bo`) will do cascade deletion of all related records to the record to delete through business object relations.

Parameters

Here are the following parameters applicable with delete operation.

Parameter	Description
`includeOcculting`	Includes occulted records. `Boolean` type, the default value is `false`.
`inheritIfInOccultingMode`	*Deprecated since 5.8.1* While it remains available for backward compatibility reasons, it will eventually be removed in a future version. Inherits the record if it is in occulting mode. `Boolean` type, the default value is `true`.
`checkNotChangedSinceLastUpdateTime`	Timestamp in datetime format used to ensure that the record has not been modified since the last read. Also see the Optimistic locking [p 1163] section. `DateTime` type value.
`blockingConstraintsDisabled`	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. `Boolean` type, default value is `false`. See Blocking and non-blocking constraints [p 945] for more information.
`filter`	Quick search predicate or complete XPath predicate [p 441] expression that defines the records to which the massive delete is applied. This parameter must imperatively be used with **:mass** extended action otherwise an error would be returned. The **ebx-all** value is used to select all records. When the viewPublication [p 1159] query parameter is used, the scope is limited to the filtered view. `String` type value, it must not be empty otherwise an error would be returned.
`viewPublication`	Specifies the name of the published view to be considered during the massive delete. The behavior of this parameter is described in the section EBX® as a Web Component [p 422]. `String` type value.

Message body

The request must define a message body only when deleting several records without using `filter` or `deleteAll` query parameter:

- **Record table**: The message contains a table of elements related to a record, with for each element one of the following properties:
  - `details`: corresponds to the record URL, it is returned by the select operation.
  - `primaryKey`: corresponds to the primary key of the record, using the XPath expression.
  - `foreignKey`: corresponds to the value that a foreign key would have if it referred to a record.

  See also *`PrimaryKey`*[API]

  Examples: Extended JSON [p 1196], Compact JSON [p 1225].

HTTP codes

HTTP code	Description
`200` *(OK)*	The operation has been executed successfully. A report is returned in the response body.
`400`*(Bad request)*	The request is incorrect. This occurs when: • the structure of the message body does not comply with Message body [p 1160]. • the message body contains a record table while the URL specifies a record.
`403` *(Forbidden)*	Authenticated user is not allowed to delete or occult the specified record.
`404` *(Not found)*	The selected record is not found. In the child dataset context, it should be necessary to use the `includeOcculting` parameter.
`409` *(Conflict)*	Concurrent modification, The Optimistic locking [p 1163] is activated and the content has changed in the meantime, it must be reloaded before deleting the record. The parameter value `checkNotChangedSinceLastUpdateTime` exists but does not correspond to the actual last update date of the record.
`422` *(Unprocessable entity)*	Only available if `blockingConstraintsDisabled` is `false`, the operation fails because of a blocking validation error.

Response body

After a successful record deletion or occulting, a report is returned in the response body. It contains the number of deleted, occulted and inherited record(s).

Example JSON [p 1240].

Count operation

Count operation may use one of the following methods:

- `GET` HTTP method,

- `POST` HTTP method without message body or
- `POST` HTTP method with message body but without `content` field on root.

The URL formats are:

- **Dataset node**: the `data` category returns the number of terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}:count
  ```

  > **Note**
  >
  > Not applicable with the `history` category.

- **Table** depending on the operation category [p 1127]:

  the `data` category returns the number of table records.

  The `history` category returns the number of table history records.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}:count
  ```

- **Field** depending on the operation category [p 1127]:

  the `data` category counts the record fields.

  The `history` category counts the history record field.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}:count
  ```

  > **Note**
  >
  > The field must be either an association node, a selection node, a terminal node or above.

Where:

- `{category}` corresponds to the operation category [p 1127] (possible values are: `data` or `data-compact`).
- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the dataset node, that can be a group node or a table node.
- `{encodedPrimaryKey}` corresponds to the percent encoded representation of the primary key (see RFC-3986 Uniform Resource Identifier).
- `{pathInRecord}` corresponds to the path starting from the table node.

Parameters

The following parameters are applicable to the count operation.

Parameter	Description
`count`	*Deprecated since 6.0.0*, replaced by the extended action from the URL. It is used to specify whether this is a count operation or a selection operation. `Boolean` type, default value is `false`.

Table parameters

The following parameters are applicable to tables, associations and selection nodes.

Parameter	Description
`filter`	Quick search predicate or complete XPath predicate [p 441] expression defines the field values to which the request is applied. If empty, all records will be considered. `String` type value. **Note** The history code operation value is usable with the `ebx-operationCode` path field from the `meta` section associated with this field.
`historyMode`	Specifies the filter context applied on table. `String` type, possible values are: • `CurrentDataSpaceOnly`: history in current dataspace • `CurrentDataSpaceAndAncestors`: history in current dataspace and ancestors • `CurrentDataSpaceAndMergedChildren`: history in current dataspace and merged children • `AllDataSpaces`: history in all dataspaces The default value is `CurrentDataSpaceOnly`. **See also** *`history` [p 1127]* **Note** This parameter is ignored with the `data` category.
`includeOcculting`	Includes the records in occulting mode. `Boolean` type, default value is `false`.
`viewPublication`	Specifies the name of the published view to be considered during the count execution. This parameter can be combined with: • the filter [p 1162] parameter as the logical `and` operation. The behavior of this parameter is described in the section EBX® as a Web Component [p 422]. `String` type value.

Selector parameters

The following parameters are only applicable to fields that return an enumeration, foreign key or `osd:resource`.

Parameter	Description
`selector`	Specifies whether: • `true`: returns the number of all possible values • `false`: returns the number of possible values for the current field. `Boolean` type, default value is `false`. **Note** This parameter is ignored with the `history` category.
`selectorFilter`	Specifies the filter of the selector. `String` type value, the syntax complies with the Quick Search [p 128].

HTTP codes

HTTP code	Description
`200` *(OK)*	The selected resource is successfully counted.
`400` *(Bad request)*	The request is incorrect. This occurs when: • The selected field in a record or a dataset is sub-terminal. • The selected dataset field is a dataset tree. • The quick search predicate or complete XPath predicate of the `filter` parameter is malformed or contains unfilterable nodes. • The table view for the `viewPublication` parameter is either hierarchical, non-existent or non-published. • The `selector` parameter is used for a non-enumerated node, or the `firstElementIndex` is negative, higher than or equal to the number of values.
`403` *(Forbidden)*	The selected resource is hidden for the authenticated user.
`404` *(Not found)*	The selected resource is not found.

Optimistic locking

To prevent an update or a delete operation on a record that was previously read but may have changed in the meantime, an optimistic locking mechanism is provided.

To enable optimistic locking, a select request must set the `includeMetadata` parameter including the `system` value.

See Technical data [p 1240] for more information.

The `update_time` property value must be included in the following request. If the record has been changed since the specified time, the update or delete operation will be cancelled.

- **Record**: update whole or partial content of the selected record.

  The `update_time` value should be added to the request body to prevent update of a modified record.

  See the JSON [p 1194] example of a record.

- **Field**: update of a single field of the selected record.

  The `update_time` value must be declared in the request URL by the `checkNotChangedSinceLastUpdateTime` parameter to prevent the update of a modified record.

The `update_time` property value can also be used in the request URL `checkNotChangedSinceLastUpdateTime` parameter to prevent deletion on a modified record.

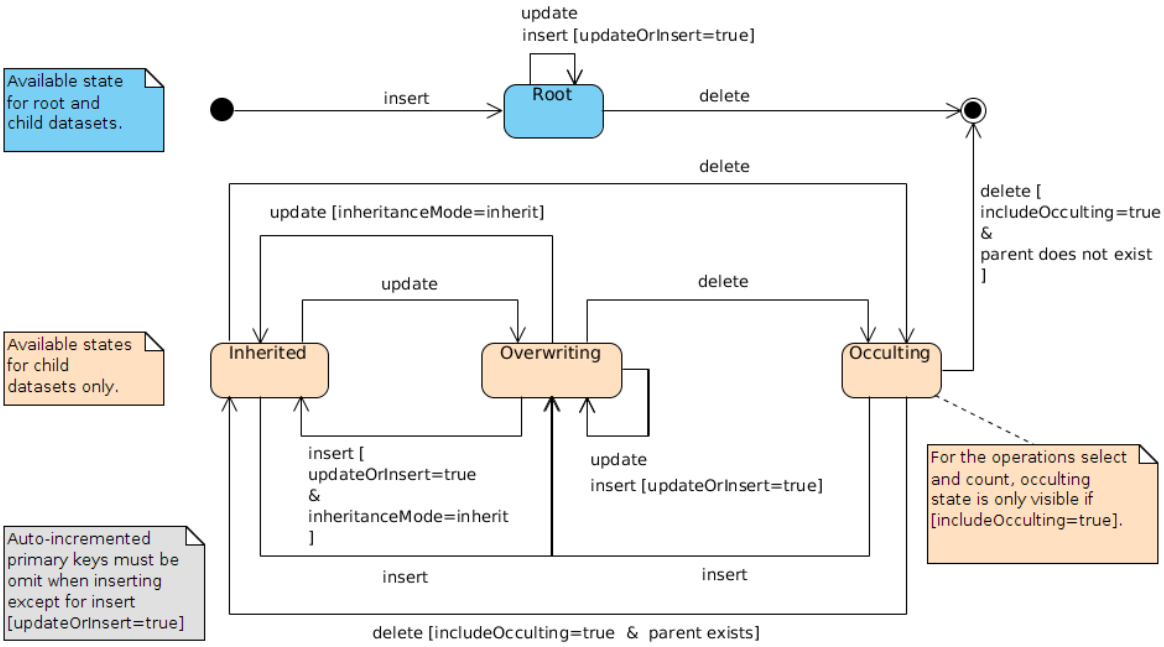
> **Note**
>
> The `checkNotChangedSinceLastUpdateTime` parameter may be used more than once but only on the same record. This implies that if the request URL returns more than one record, the request will fail.

Inheritance

EBX® inheritance features are supported by built-in RESTful services using specific properties and automatic behaviors. In most cases, the inheritance state will be automatically computed by the server according to the record and field definition or content. Every action that modifies a record or a field may have an indirect impact on those states. In order to fully handle the inheritance life cycle, direct modifications of the state are allowed under certain conditions. Forbidden or incoherent explicit alteration attempts are ignored.

See also *Inheritance and value resolution* *[p 480]*

Record inheritance life cycle in built-in RESTful services



Inheritance properties

The following table describes properties related to the EBX® inheritance features.

Property	Location	Description
`inheritance`	record or table metamodel	Specifies if dataset inheritance is activated for the table. The value is computed from the data model and cannot be modified through built-in RESTful services. **See also** *`inheritance` property in metamodel.* [p 1206]
`inheritedField`	field metamodel	Specifies the field's value source. The source data are directly taken from the data model and cannot be modified through built-in RESTful services. **See also** *`inheritedField` property in metamodel.* [p 1207]
`inheritanceMode`	record in child dataset	Specifies the record's inheritance state. To set a record's inheritance from `overwrite` to `inherit`, its `inheritanceMode` value must be explicitly provided in the request. In this specific case, the `content` property will be ignored if present. `occult` and `root` explicit values are always ignored. An `overwrite` explicit value without a `content` property is ignored. **Note** Inherited record's fields are necessarily `inherit`. **Note** Root records in child dataset will always be `root`. Possible values are: `root`, `inherit`, `overwrite`, `occult`. For more information, see Record lookup mechanism [p 482].
	field in `overwrite` record	Specifies the field's inheritance state. To set a field's inheritance to `inherit`, its `inheritanceMode` value must be explicitly provided in the request. The `content` property will be ignored in this case. `overwrite` of an explicit value without a `content` property is ignored. **Note** `inheritanceMode` at the field level is not available for `root`, `inherit` and `occult` records, primary keys, and read-only fields. **Note** `inheritedFieldMode` and `inheritanceMode` properties cannot be both set on the same field. Possible values are: `inherit`, `overwrite`. For more information, see Inheritance and value resolution [p 480].

Property	Location	Description
`inheritedFieldMode`	inherited field	Specifies the inherited field's inheritance state. To set a field's inheritance to `inherit`, its `inheritedFieldMode` value must be explicitly provided in the request. The `content` property will be ignored in this case. `overwrite` explicit values without a `content` property are ignored. **Note** `inheritedFieldMode` does not display for read-only fields. **Note** `inheritedFieldMode` and `inheritanceMode` properties cannot be both set on the same field. **Note** `inheritedFieldMode` has priority over the `inheritanceMode` property. Possible values are: `inherit`, `overwrite`. For more information, see Value lookup mechanism [p 483].

Look up table views operation

The "look up published views" operation may use one of the following methods:

- `GET` HTTP method or
- `POST` HTTP method without message body.

The URL format is:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
{tablePath: [^:]*}:publishedViews
```

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the image identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{tablePath: [^:]*}` corresponds to the table path.

Parameters

No specific parameter for this operation.

Codes HTTP

HTTP code	Description
`200` *(OK)*	Information successfully returned.
`400` *(Bad request)*	Incorrect request, contains an error.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	The specified resource read permission has been denied to the authenticated user.
`404`*(Not found)*	The selected resource cannot be found.

Response body

It contains a collection of authorized view information (including the access URI).

Example: JSON [p 1246].

174.7 Form data operations

The form-data category operations concern the dataset fields, records or record fields when creating or modifying content must comply with constraints on user form submit. They are intended to be used in a user form management context.

The request body of an operation is very similar to the equivalent operation of the `data` category in the sens that an incoming data validation feature and a result report has been added to the response. The data validation can not be deactivated, a parameter `blockingConstraintsDisabled` from the `data` category operations is not applicable. The validation phase fails when, at least, one constraint with a `blocksCommit` level set to `onInsertUpdateOrDelete` or `onUserSubmit-checkModifiedValues` is violated.

See Blocking and non-blocking constraints [p 945] for more information.

Form insert operation

Insert form uses the `POST` HTTP method and requires a message body holding the data. This operation supports one or multiple records in a single transaction. Moreover, it is also possible to update record(s).

- **Record**: validate and insert a new record or modify an existing one in the selected table.
- **Record table**: validate and insert or modify one or more records in the selected table, while securing a consistent response. Operations are executed sequentially, in the order defined on the client side. When an error occurs during a table operation, all updates are cancelled and the client receives an error message with detailed information. The maximum number of records is limited to 100.

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/{dataset}/
{pathInDataset}[:mass]
```

Where:

- `{category}` corresponds to the operation category [p 1127] (possible values are: `form-data` or `form-data-compact`).
- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the table node.
- `:mass` extended action is required when updating a filtered set of records (see filter parameter [p 1169]).

Parameters

The following parameters are applicable with this insert operation.

Parameter	Description
`includeDetails`	Includes the `details` field in the answer to access the data details. The returned value corresponds to its URL resources. Type `Boolean`, the default value is `false`. **Note** Only applicable for a multiple records insertion.
`includeForeignKey`	Includes the `foreignKey` field in the answer for each record. The returned value corresponds to the value of a foreign key field that was referencing this record. `Boolean` type, the default value is `false`. **Note** Only applicable for a multiple records insertion.
`includeLabel`	Includes the `label` field in the answer for each record. Possible values are: • `yes`: the `label` field is included. • `no`: the `label` field is not included (use case: integration). `String` type, the default value is `no`. **Note** Only applicable for a multiple records insertion.
`updateOrInsert`	Specifies the behavior when the record to insert already exists: • If `true`: the existing record is updated with new data. • If `false`: a client error is returned and the operation is aborted. `Boolean` type, default value is `false`.
`filter`	Determines the records updated using a quick search predicate, or complete XPath predicate [p 441] expression when a large number of records are updated. The request body must be a **record** structure with the fields to update. This parameter must use the**:mass** extended action. Otherwise, an error is returned. Use the **ebx-all** value to select all records. When the viewPublication [p 1170] query parameter is used, the scope is limited to the filtered view. `String` type value that must not be empty. Otherwise, an error is returned. **Note** When using the query parameters includeDetails [p 1169], includeLabel [p 1169] or includeForeignKey [p 1169], only the first 1000 report entries are returned in the report.

Parameter	Description
`viewPublication`	Specifies the name of the published view used during a massive update. The behavior of this parameter is described in the section EBX® as a Web Component [p 422]. `String` type value.

Message body

The request must define a message body. The format is similar to the `data` category insert operation's message body [p 1154].

HTTP codes

HTTP code	Description
`200` *(OK)*	• The request body holds multiple **records**: the insert/update request was applied successfully, a report is returned in the response body. • The request body holds only one **record** and `updateOrInsert` is `true`: the update request was applied successfully, a report is returned in the response body.
`201` *(Created)*	The request body holds only one non-existing **record**: a new record has been created and the header field `Location` is returned with its resource URL. Moreover, a report is returned in the response body.
`400` *(Bad request)*	The request is incorrect. This occurs when the body message structure: • does not comply with what was mentioned in Message body [p 1170]. • complies with what was mentioned in Message body [p 1170], but number of record insert/update more than 100.
`403` *(Forbidden)*	Authenticated user is not allowed to create a record or the request body contains a read-only field.
`404` *(Not found)*	The selected resource is not found.
`409` *(Conflict)*	Concurrent modification, only available if `updateOrInsert` is `true`, the Optimistic locking [p 1163] is activated and the content has changed in the meantime, it must be reloaded before update.
`422` *(Unprocessable entity)*	The request cannot be processed. This occurs when: • A blocking constraint has been violated. In this case an appropriate report is returned in the response body. • The record cannot be inserted because another one with the same primary key already exists (only available if `updateOrInsert` is `false`). • The record cannot be inserted because the definition of the primary key is either non-existent or incomplete. • The record cannot be updated because the value of the primary key cannot be modified.

Response body

The response body will always hold a validation report. However in case of failure, the response body corresponds to a JSON `Exception handling` response.

- **Record**: The header field `Location` is returned with its URL resource. The validation report will be include in response body.
- **Record table**: (optional) Contains a list of validations report for each element, corresponding to the multiple validation report.

See also *Form data category* *[p 1233]*

Form update operation

As for the `data` or `data-compact` category update operation, it allows the modification of a single dataset or record and the `PUT` HTTP method must be used. Available URL formats are:

- **Dataset node**: validates and modifies the values of terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}
  ```

- **Record**: validates and modifies the content of the selected record.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}/{encodedPrimaryKey}
  ```

  > **Note**
  >
  > Also available for `POST` HTTP method. In this case, the URL must point to the table and the parameter `updateOrInsert` must be set to `true`.

- **Field**: validates and update of a single field of the selected record.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
  ```

  > **Note**
  >
  > The field must be either a terminal node or above.

Where:

- `{category}` corresponds to the operation category [p 1127] (possible values are: `form-data` or `form-data-compact`).
- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the dataset node:
  - for dataset node operations, this must be any terminal node or above except table node,
  - for record and field operations, this corresponds to the table node.
- `{encodedPrimaryKey}` corresponds to the percent encoded representation of the primary key (see RFC-3986 Uniform Resource Identifier).
- `{pathInRecord}` corresponds to the path starting from the table node.

Parameters

Here are the parameters applicable with the update operation.

Parameter	Description
`byDelta`	Specifies the behavior for setting value of nodes that are not defined in the request body. This is described in the Update modes [p 1241] section. `Boolean` type, the default value is `true`.
`checkNotChangedSinceLastUpdateTime`	Timestamp in datetime format used to ensure that the record has not been modified since the last read. Also see the Optimistic locking [p 1163] section. `DateTime` type value.

Message body

The request must define a message body. The format is the same as for the `data` category update operation message body [p 1157].

HTTP codes

HTTP code	Description
`200` *(Ok)*	The update request was applied successful and a report is returned in the response body.
`400` *(Bad request)*	The request is incorrect. This occurs when the body request structure does not comply.
`403` *(Forbidden)*	Authenticated user is not allowed to update the specified resource or the request body contains a read-only field.
`404` *(Not found)*	The selected resource is not found.
`409` *(Conflict)*	Concurrent modification, the Optimistic locking [p 1163] is activated and the content has changed in the meantime, it must be reloaded before the update.
`422` *(Unprocessable entity)*	The request cannot be processed. This occurs when: • A blocking constraint has been violated. In this case an appropriate report is returned in the response body. • The record cannot be updated because the value of the primary key cannot be modified.

Response body

The response body has the same format and behavior as the Form Insert operation [p 1171].

174.8 Dataset operations

These operations perform selection over datasets.

See also *Datasets [p 122]*

Select datasets

Select operations can use the `GET` or `POST` methods.

> **Note**
>
> When `POST` is used, the message body is always ignored.

URL formats are:

- **Root**: returns the root datasets of a given dataspace.

  A pagination mechanism is always enabled.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}
  ```

- **Children**: returns the children datasets of a given dataset.

  A pagination mechanism is always enabled.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
  {dataset}:children
  ```

- **Information**: returns dataset information.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
  {dataset}:information
  ```

Where:

- `{category}` corresponds to the operation category [p 1127] (possible values are: `data` or `data-compact`).
- `{dataspace}` corresponds to `B` followed by the dataspace's identifier or to `V` followed by the snapshot's identifier.
- `{dataset}` corresponds to the dataset identifier.

Parameters

The following query parameters are applicable to **Root** and **Children** operations.

Parameter	Description
`includeLabel`	See includeLabel [p 1141] for more information.
`includeDetails`	See includeDetails [p 1141] for more information.
`includeOpenApiDetails`	See includeOpenApiDetails [p 1142] for more information.
`includeValidation`	See includeValidation [p 1143] for more information.
`includeHistory`	See includeHistory [p 1141] for more information.
`firstElementIndex`	See firstElementIndex [p 1148] for more information.
`pageSize`	See pageSize [p 1148] for more information.

The following query parameters are applicable to **Information** operation.

Parameter	Description
`includeLabel`	See includeLabel [p 1141] for more information.

HTTP codes

HTTP code	Description
`200` *(OK)*	The request has been successfully handled.
`400` *(Bad request)*	The request is incorrect. This occurs when: • `firstElementIndex` parameter's value is malformed or out of bound. • `pageSize` parameter's value is malformed or out of bound.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	The selected resource is hidden from the authenticated user.
`404` *(Not found)*	The selected resource could not be found.

Response body

After a successful selection, the result is returned in the response body. The content depends on the provided parameters and selected data.

The format is linked to the selected object type:

- For **Root** and **Children**, see the extended JSON [p 1199] and compact JSON [p 1226] example.
- For **Information**, see the extended JSON [p 1200] and compact JSON [p 1227] example.

174.9 Dataspace operations

These operations perform a selection or life cycle management over dataspaces.

See also *Dataspaces [p 104]*

Beta feature: Select dataspaces or snapshots

Select operation may use one of the following methods: `GET` or `POST`. A pagination mechanism is always enabled.

URL formats are:

- **Root**: returns root dataspaces
  `http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/`
- **Children**: returns children dataspaces of a given dataspace
  `http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/{dataspace}:children`

- **Snapshots**: returns snapshots of a given dataspace
  `http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/{dataspace}:snapshots`
- **Information**: returns a dataspace, or a snapshot, information
  `http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/{dataspace}:information`

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace's identifier or to `V` followed by the snapshot's identifier.

Parameters

The following query parameters are applicable to **Root**, **Children** and **Snapshots** operations.

Parameter	Description
`includeClosed`	Includes the closed dataspaces to the selection. `Boolean` type, default value is `false`.
`includeAdministration`	Includes the administration dataspaces to the selection. `Boolean` type, default value is `false`.
`pageRecordFilter`	Specifies a server side built filter, for pagination, pointing to a record. This filter is **strongly** linked to the `pageAction` value and should not be modified on the client side. The filter takes the form of a record XPath predicate [p 441] expression used to figure out the pagination contexts. `String` type. **See also** *Pagination* *[p 1236]*
`pageAction`	Specifies the pagination action to perform from the identifier held by `pageRecordFilter`. `String` type, default value is `first`. The possible values are: • `first` • `previous` • `next` • `last` **See also** *`RequestPagination`*[API]
`pageSize`	Specifies the maximum number of records per page. `Integer` type, default value is based on the user preferences. The value should be between `2` and `100`.

HTTP codes

HTTP code	Description
`200` *(OK)*	The request has been successfully handled.
`400` *(Bad request)*	The request is incorrect. This occurs when: • `pageAction` parameter's value is not included in allowed values. • `pageRecordFilter` is malformed or non-existent when selecting next or previous page. • `pageSize` parameter's value is out of bound.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	The selected resource is hidden from the authenticated user.
`404` *(Not found)*	The selected resource could not be found.

Response body

After a successful selection, the result is returned in the response body. The content depends on the provided parameters and selected data.

The format is linked to the selected object type:

- For **Root**, **Children**, or **Snapshots**, see the JSON [p 1247] example.
- For **Information**, see the JSON [p 1249] example.

Beta feature: Create a child dataspace or a snapshot

Creates the dataspace or snapshot as specified. This operation use the `POST` method with a body request (with no specific query parameter).

See also *`Repository.createHome`*[API]

URL format are:

- **Dataspace**:
  `http[s]://<host>[:<port>]/.../data/v1/{dataspace}:createDataspace`
- **Snapshot**:
  `http[s]://<host>[:<port>]/.../data/v1/{dataspace}:createSnapshot`

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier.

Request body

The body specifies the features of the dataspace or the snapshot to create.

See also *`HomeCreationSpec`*[API]

See the JSON [p 1250] example.

HTTP codes

HTTP code	Description
`201` *(Created)*	A new dataspace or snapshot has been created, in this case, the header field `Location` is returned with its resource URL.
`400` *(Bad request)*	The request body is not well-formed or contains invalid content.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	Permission was denied to create the specified resource for the authenticated user.
`404` *(Not found)*	The resource specified in the request body cannot be found.
`500` *(Internal error)*	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

Beta feature: Locking a dataspace

Locks the specified dataspace. If the dataspace is already locked:

- by the same user, then the lock is kept,
- by another user and during the wait duration, then the lock is acquired,
- otherwise, the lock is rejected.

This operation uses the `POST` method and consumes `Content-Type` header set to:

- `application/x-www-form-urlencoded`: with HTTP parameters in the body or
- `application/json`: with HTTP parameters in the URL and Session parameters [p 1115] into JSON body.

When it succeeds, no response body is returned.

See also

Permissions for locking or unlocking a dataspace [p 489]

`LockSpec.lock`[API]

The URL format is:

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:lock
```

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier.

Parameters

The following query parameters are applicable to the operation.

Parameter	Description
`durationToWaitForLock`	When the dataspace is locked by another user, specifies the maximum duration to wait for acquiring the lock. The duration is specified in seconds. If the value is set to `0`, then the locking attempt is perform immediately. Due to several reasons, the wait duration can not exceed `60` seconds. Otherwise, the value is overwritten with its maximum value. `Integer` type, default value is `0`.

HTTP codes

HTTP code	Description
`204` *(No content)*	The request has been successfully handled but no response body is returned.
`400` *(Bad request)*	The request URL or body is not well-formed or contains invalid content.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	Permission was denied to lock the specified resource for the authenticated user.
`404` *(Not found)*	The resource specified in the URL cannot be found.
`409` *(Conflict)*	The resource is already locked by another user, the lock is rejected after `durationToWaitForLock` parameter's value.
`500` *(Internal error)*	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

Beta feature: Unlocking a dataspace

Unlocks the specified dataspace. If the dataspace is:

- locked by the same user, then the lock is released,
- locked by another user, the current user is an administrator, and the `forceByAdministrator` query parameter is set to `true`, then the lock is released,
- not locked, the lock status is unchanged,
- otherwise, the unlock is rejected.

This operation uses the `POST` method and consumes `Content-Type` header set to:

- `application/x-www-form-urlencoded`: with HTTP parameters in the body or
- `application/json`: with HTTP parameters in the URL and Session parameters [p 1115] into JSON body.

When it succeeds, no response body is returned.

See also

Permissions for locking or unlocking a dataspace [p 489]

`LockSpec.unlock`[API]

The URL format is:

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:unlock
```

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier.

Parameters

The following query parameters are applicable to the operation.

Parameter	Description
`forceByAdministrator`	When the dataspace is locked by another user, an administrator can force the unlocking. `Boolean` type, default value is `false`.

HTTP codes

HTTP code	Description
`204` *(No content)*	The request has been successfully handled but no response body is returned.
`400` *(Bad request)*	The request URL or body is not well-formed or contains invalid content.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	Permission was denied to unlock the specified resource for the authenticated user.
`404` *(Not found)*	The resource specified in the URL cannot be found.
`409` *(Conflict)*	The resource is already locked by another user, or the current user is administrator but `forceByAdministrator` parameter is `false`, the unlock is rejected.
`500` *(Internal error)*	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

Beta feature: Merge a dataspace

Merges the specified dataspace to its parent. It is possible to perform deletion, after the merge, on history and / or on data.

This operation uses the `POST` method and consumes `Content-Type` header set to:

- `application/x-www-form-urlencoded`: with HTTP parameters in the body or
- `application/json`: with HTTP parameters in the URL and Session parameters [p 1115] into JSON body.

When it succeeds, no response body is returned.

> **Note**
>
> The merge decision step is bypassed for merges performed through data services. In such cases, the data in the child dataspace automatically overrides the data in its parent.

See also

`ProcedureContext.doMergeToParent`[API]

Deleting data and history [p 667]

The URL format is:

`http[s]://<host>[:<port>]/.../data/v1/{dataspace}:merge`

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier.

Parameters

The following query parameters are applicable to the operation.

Parameter	Description
`deleteHistoryOnMerge`	Sets whether the history associated with the specified dataspace will be deleted upon merge. `Boolean` type, default value is `false`. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.historyDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 587].
`deleteDataOnMerge`	Sets whether the specified dataspace and its associated snapshots will be deleted upon merge. `Boolean` type, default value is `false`. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.dataDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 587].

HTTP codes

HTTP code	Description
`204` *(No content)*	The request has been successfully handled but no response body is returned.
`400` *(Bad request)*	The request URL contains invalid content.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	Permission was denied to merge the specified resource for the authenticated user.
`404` *(Not found)*	The resource specified in the URL cannot be found.
`500` *(Internal error)*	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

Beta feature: Close a dataspace or a snapshot

Closes the specified dataspace or snapshot. It is possible to perform deletion, after close, on history and / or on data.

This operation uses the `POST` method and consumes `Content-Type` header set to:

- `application/x-www-form-urlencoded`: with HTTP parameters in the body or
- `application/json`: with HTTP parameters in the URL and Session parameters [p 1115] into JSON body.

When it succeeds, no response body is returned.

See also

`Repository.closeHome`[API]

Deleting data and history [p 667]

The URL format is:

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:close
```

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.

Parameters

The following query parameters are applicable to the operation.

Parameter	Description
`deleteHistoryOnClose`	Sets whether the history associated with the specified dataspace will be deleted upon close. `Boolean` type, default value is `false`. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.historyDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 587].
`deleteDataOnClose`	Sets whether the specified dataspace and its associated snapshots will be deleted upon close. `Boolean` type, default value is `false`. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.dataDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 587].

HTTP codes

HTTP code	Description
`204` *(No content)*	The request has been successfully handled but no response body is returned.
`400` *(Bad request)*	The request URL or body is not well-formed or contains invalid content.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	Permission was denied to close the specified resource for the authenticated user.
`404` *(Not found)*	The resource specified in the URL cannot be found.
`422` *(Unprocessable entity)*	A blocking constraint has been violated.
`500`*(Internal error)*	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

174.10 Health operations

Overview

Operations health helps monitor the EBX® server.

Started operation

Returns status code 200 (Ok) when EBX® is started and initialized, otherwise it returns 503 (Service unavailable).

The URL format is:

```
http[s]://<host>[:<port>]/.../health/v1/started
```

HTTP codes

HTTP code	Description
`200` *(OK)*	The request was successfully handled and a response body is returned.
`503` *(Service unavailable)*	EBX® is not yet started and initialized. A message is returned to help with diagnostics.

Response body

See the JSON [p 1251] example.

Check operation

Returns the status code 200 (Ok) when EBX® is started and checked, otherwise it returns 503 (Service unavailable).

The URL format is:

```
http[s]://<host>[:<port>]/.../health/v1/check
```

HTTP codes

HTTP code	Description
`200` *(OK)*	The request was successfully handled and a response body is returned.
`503` *(Service unavailable)*	EBX® is not yet started and initialized. A message is returned to help with diagnostics.

Response body

See the JSON [p 1251] example.

Stopped operation

Returns the status code 200 (Ok) when the EBX® repository is shutdown, otherwise it returns 503 (Service unavailable).

The URL format is:

```
http[s]://<host>[:<port>]/.../health/v1/stopped
```

HTTP codes

HTTP code	Description
`200` *(OK)*	The request was successfully handled and a response body is returned.
`503` *(Service unavailable)*	EBX® is not yet started and initialized. A message is returned to help with diagnostics.

Response body

See the JSON [p 1251] example.

174.11 OpenAPI operations

Overview

The `api` [p 1127] category operations comply with the OpenAPI specification 3.0.X to generate JSON documents. These documents facilitate development and consumption by structuring and describing the available REST built-in resources and the operations associated with them.

OpenAPI document HATEOAS links are available through the select data operations when using the includeOpenApiDetails [p 1142] query parameter.

The REST OpenAPI services permissions are globally defined in Global permissions [p 1120].

Making the proper types mapping for generating a client, in a specific language, is essential. The OpenAPI format can be used to validate the input or to map the value to a specific type, in the chosen programming language. See Content of simple type [p 1219] for more information.

> **Note**
>
> Tools that do not support a specific format may default back to the `type` alone.

Generate OpenAPI documents

The operations use the `GET` or `POST` HTTP method to generate the OpenAPI document of a dataset, a table, or a schema node.

The URL formats are:

- **Dataset:**

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/{category}/v1/{dataspace}/{dataset}
  ```

- **Table and Schema node:**

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/{category}/v1/{dataspace}/{dataset}/{pathInDataset}
  ```

Where:

- `{category}` corresponds to an operation category [p 1127] among data [p 1138], data-compact [p 1138] or form-data [p 1167].

- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of:
  - a table node,
  - a dataset terminal node or above.

> **Note**
>
> The generated document will not depend on user permissions. The whole fields will be presented.

HTTP codes

HTTP code	Description
`200`*(OK)*	The request has been successfully handled.
`401`*(Unauthorized)*	Authentication has failed.
`403`*(Forbidden)*	Permission was denied to read the specified resource.
`404`*(Not found)*	The resource, specified in the URL, cannot be found.

174.12 Staging operations

The Staging [p 777] REST API is designed to interact with source and target servers. It allows users to manage domains using export and import of staging archives.

The complete description of the services is available via OpenAPI on the following link:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/staging/v1
```

In the development run mode [p 599], the Swagger UI is available on the following link.

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/staging/v1/ui
```

See also *Visual documentation [p 1299]*

Domain management operations

The *domains* service allows the users to manage domains. The provided features are:

- Select the list of available domains.
- Select a domain by unique name.
- Create a new domain.
- Select the components of a domain.
- Add a component to a domain.
- Remove a component from a domain, the component is deleted.

Staging element navigation operations

The *elements* service is designed to display the whole elements of an instance in a hierarchical way. It is mainly designed for a UI interface that supports an hierarchical view with levels handling. The purpose of using query parameters instead of a structured body is to use links for easier navigation. The provided features are:

- Expands the content of an element.
- Expands the content of a specific type of elements under a selected element.

Archive manipulation operations

The *archives* service is designed for importing/exporting staging archives to/from a running instance in a single request. It is also used for uploading archives for asynchronous import. The provided features are:

- Export the archive of a domain.
- Import an archive to a running instance.
- Upload an archive for an asynchronous import.

The easiest way to export a staging archive via cURL

```
curl --request GET 'http[s]://<host>[:<port>]/ebx-dataservices/rest/staging/v1/archives/<domain>?
login=<login>&password=<password>'
```

The easiest way to import a staging archive via cURL

```
curl --request POST 'http[s]://<host>[:<port>]/ebx-dataservices/rest/staging/v1/archives?
login=<login>&password=<password>'
--header 'Content-Type: application/zip'
--data-binary '@/<archive-path-on-disk>'
```

Export operations

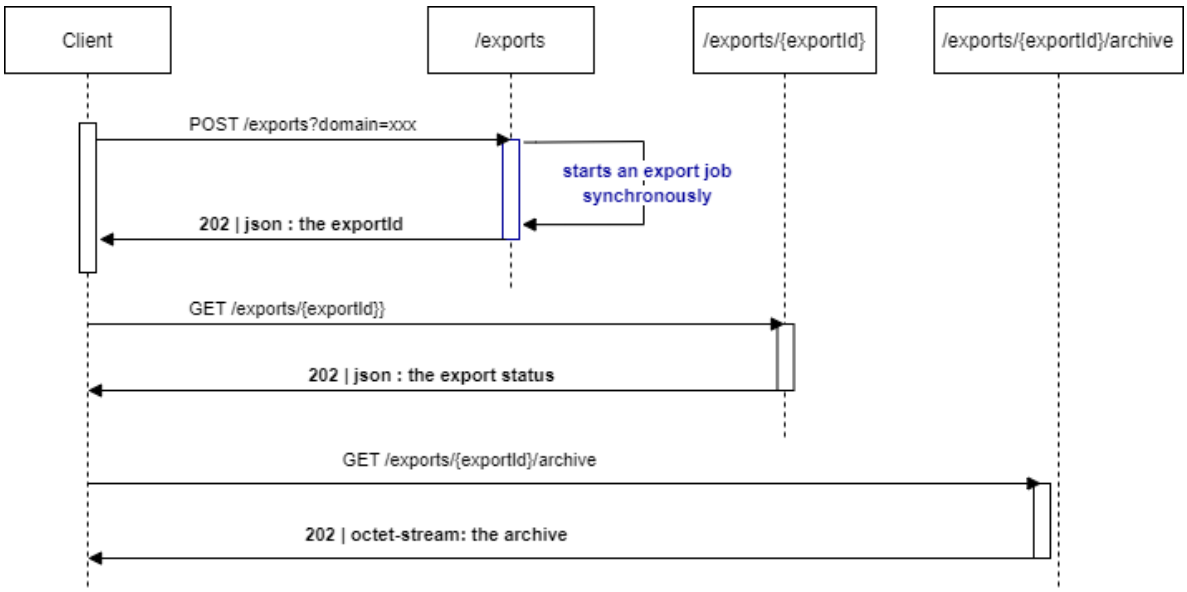
The *exports* service is designed to handle advanced export features. The archive is exported and stored on the server for a predefined period. The user can retrieve information about the archive, and it will be available to download it until it expires. The provided features are:

- Initiate an export.
- Get the status of an initiated export.
- Download an exported archive.

This endpoint can work in both synchronous and asynchronous modes.

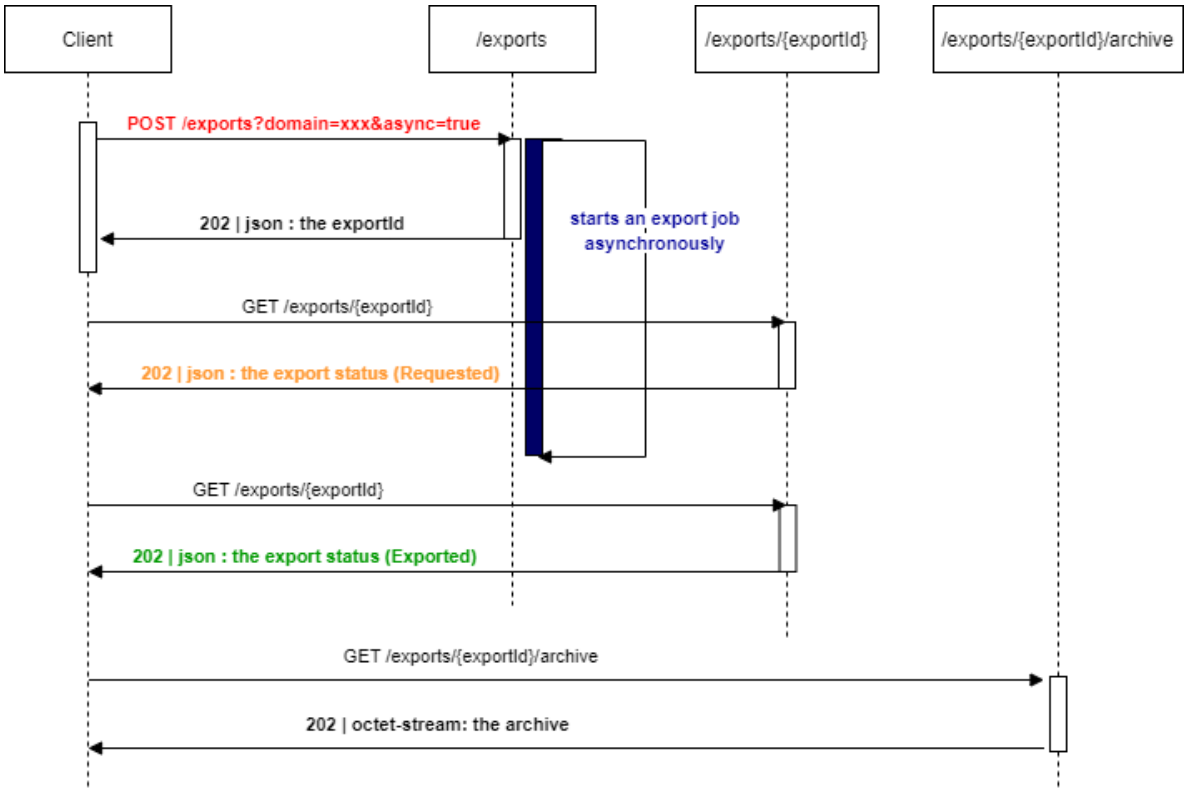
Synchronous mode

A blocking mode that responds after exporting the whole archive on the server. The Archive will be immediately downloadable.



Asynchronous mode

A non-blocking mode that responds with an export ID and launches a task to export the archive. The client must poll the status endpoint to retrieve the status of the export. When the status changes to "Exported", the archive would be downloadable.



For consistency, any staging asynchronous job is aborted if it does not complete before 60 minutes of work.

Import operations

The *imports* service is designed to handle advanced import features. The archive is uploaded and stored on the server for a predefined period. The user can initiate an import asynchronously or retrieve information about an in-progress or completed import.

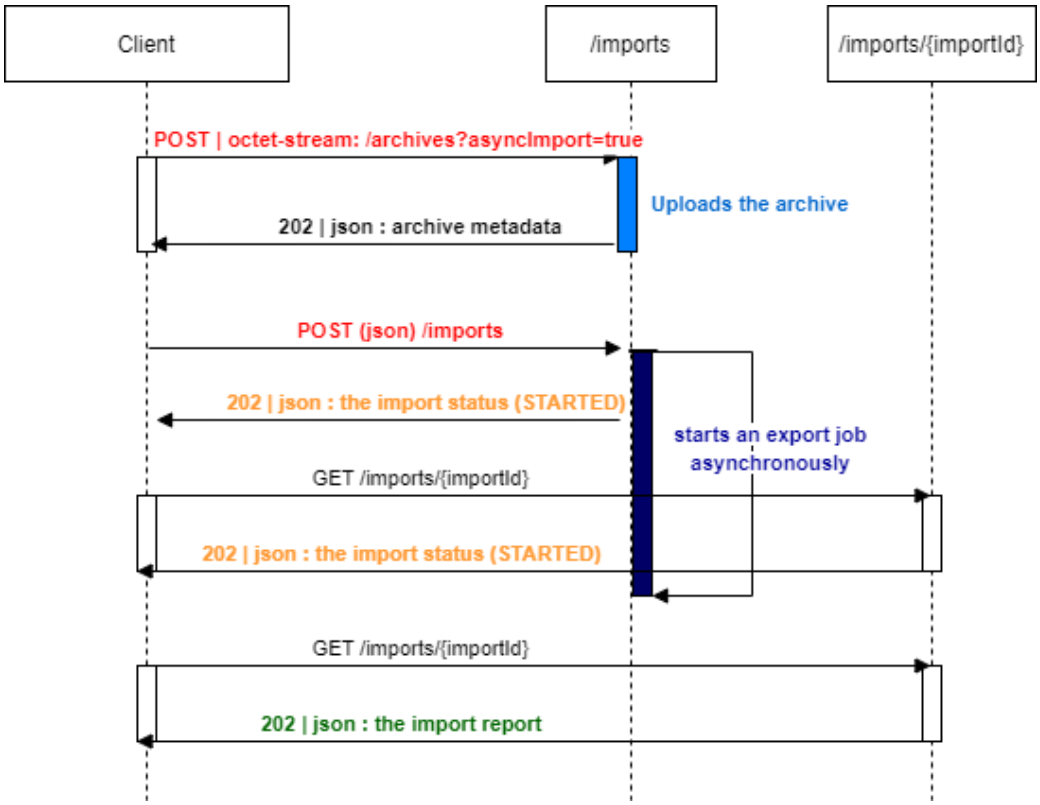
The archive is uploaded using the *archives* service with the option `asyncImport=true`.

The provided features are:

- Initiate an import by using an already uploaded archive.
- Get the status/report of an initiated import.

The archive name can be retrieved from an old import to initiate a re-import of the same archive with different options. It will be available until it expires.

This endpoint works only in asynchronous mode:



174.13 Data model operations

The REST API allows you to publish existing data models defined in the Data Model Assistant (DMA). See Publish DMA data model [p 1011] for more information about the available features.

174.14 Limitations

General limitations

- Indexes, in the request URL `{pathInDataset}` or `{pathInRecord}`, are not supported.

- Dataset nodes and field operations applied to nodes that are sub-terminal are not supported.

  See Access properties [p 961] for more information about terminal nodes.

Compact format limitations

- Inheritance [p 1164], tracking information [p 1115], session parameters [p 1115], procedure context [p 1118], and a field as a selection node are not handled in the compact format [p 1223]. Use the extended [p 1193] format instead.
- `History category` [p 1127] is not supported in the compact format [p 1223].

Business object category limitations

- In business object category [p 1127]: sort, filter, and `history` [p 1127] selection features can not be applied to a field under a relationship defined in a business object.

Read operations

- Within the `selector`, the pagination context is limited to the `nextPage` property.
- When the `sortByRelevancy` [p 1146] parameter is activated, the ones below are ignored: sort [p 1145], sortOnLabel [p 1146], sortPriority [p 1147] and sort criteria defined through the viewPublication [p 1147].
- Within the `viewPublication` parameter, the hierarchical and tile views are not supported.
- The `sortOnLabel` parameter ignores programmatic labels.
- The system information response's properties cannot be browsed through the REST URL with the hierarchical representation.

  See System information operation [p 1134] for more information.
- The prepare for creation operation does not support a dataset node.
- The select dataspaces operation [p 1174] does not sort them by label.
- The select snapshots operation [p 1174] does not sort them by creation date.
- The select snapshots operation [p 1174] can not include the initial snapshots.

Write operations

- Association fields cannot be updated, therefore, the list of associated records cannot be modified directly.
- Control policy `onUserSubmit-checkModifiedValues` of the user interface is not supported. To retrieve validation errors, invoke the select operation on the resource by including the `includeValidation` parameter.

  See Blocking and non-blocking constraints [p 945] for more information.

Directory operations

- Changing or resetting a password for a user is not supported.

OpenAPI operations

- The document generation is not available through the user interface.

- The document generation REST services does not support the YAML format.
- Supports only Data operations [p 1138] and Form data operations [p 1167] except:
  - Select **Dataset tree**, **Dataset node** and **Field** operations.
  - Insert or delete multiple records in a single request.
  - Use of the HTTP header `content-type: x-www-form-urlencoded` to send query parameters in the body request.
- The 'Basic Authentication Scheme' [p 1119] is the only described method.

CHAPTER 175

Introduction

This chapter contains the following topics:

1. Overview

175.1 Overview

The JSON (JavaScript Object Notation) is the data-interchange format used by TIBCO EBX®'s RESTful operations [p 1125].

This format is lightweight, self-describing and can be used to design UIs or to integrate EBX® in company's information system.

The following JSON formats are available:

- Using the extended format [p 1193], the data context is exhaustive and contains features to retrieve metamodel data, labels, validation report, inheritance data, URLs (OpenAPI, details, selector, historyDetails, etc.), history advanced data, except for association fields and selection nodes. However, these fields are reachable from the response through URL links called details included by default.
- Using the compact format [p 1223], the data context is limited to master data without metamodel data except links for enumerated fields.

See also *Hiding a field in Data Services [p 964]*

The amount of retrieved data is limited by a pagination [p 1236] mechanism which can be configured.

URL links allow reaching:

- Tables, records, dataset non-terminal nodes, foreign keys, resource fields through the `details` property. See includeDetails [p 1141] for more information.
- Possible values for foreign keys or enumerations, by activating the `selector` parameter. See includeSelector [p 1142] for more information.

See also *Activation and configuration [p 1115]*

> **Note**
>
> JSON data are always encoded with the UTF-8 charset.

CHAPTER 176

Extended

This chapter contains the following topics:

1. Introduction
2. Global structure
3. Metamodel
4. Sort criteria
5. Validation report
6. Constraints
7. Content

176.1 Introduction

The JSON extended format is used to retrieve master data, technical information and metadata. It is designed in an expanded way that allows to include several features such as validation, sorting and so on. To activate the extended format, the unsuffixed REST category, like `data` or `form-data`, must be used in the URL.

176.2 Global structure

JSON Request body

The request body is represented by a JSON `Object` whose content varies according to the operation and the category.

Data category

The request body contains at least a `content` property which hold master data values.

- **Dataset node**

  Specifies the target values of terminal nodes under the specified node. This request is used for the dataset node update operation.

```
{
  "content": {
    "nodeName1": {
      "content": true
    },
    "nodeName2": {
      "content": 2
```

```
    },
    "nodeName3": {
      "content": "Hello"
    }
  }
}
```

See also *Update operation* *[p 1156]*

- **Record**

  Specifies the target record content by setting the value of each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is used for table record insert, record update or selector's select operation when local enumeration dependency field values are declared.

  See also *Inheritance* *[p 1164]*

  Some technical data can be added beside the `content` property such as `lastUpdateDate`.

  See also *Optimistic locking* *[p 1163]*

```
{
  ...
  "lastUpdateDate": "2015-12-25T00:00:00.001",
  ...
  "content": {
    "gender": {
      "content": "Mr."
    },
    "lastName": {
      "content": "Chopin"
    },
    "lastName-en": {
      "content": "Chopin",
      "inheritedFieldMode": "inherit"
    },
    "firstName": {
      "content": "Fryderyk"
    },
    "firstName-en": {
      "content": "Frédéric",
      "inheritedFieldMode": "overwrite"
    },
    "birthDate": {
      "content": "1810-03-01"
    },
    "deathDate": {
      "content": "1849-10-17"
    },
    "jobs": {
      "content": [
        {
          "content": "CM"
        },
        {
          "content": "PI"
        }
      ]
    },
    "infos": {
      "content": [
        {
          "content": "https://en.wikipedia.org/wiki/Chopin"
        }
      ]
    }
  }
}
```

See also

Insert operation *[p 1152]*

Update operation [p 1156]

- **Record fields**

  Specifies the target values of fields under the record terminal node by setting the value of each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is only used for table record updates.

  See also *Inheritance [p 1164]*

```
{
  "content": [
    {
      "content": "CM"
    },
    {
      "content": "PI"
    }
  ]
}
```

  See also *Update operation [p 1156]*

- **Record table**

  Defines the content of one or more records by indicating the value of each field. For missing fields, the behavior depends on the `byDelta` parameter of the request. This structure is used upon insert or update records in the table.

```
{
  "rows": [
    {
      "content": {
        "gender": {
          "content": "M"
        },
        "lastName": {
          "content": "Saint-Saëns"
        },
        "firstName": {
          "content": "Camille"
        },
        "birthDate": {
          "content": "1835-10-09"
        },
        ...
      }
    },
    {
      "content": {
        "gender": {
          "content": "M"
        },
        "lastName": {
          "content": "Debussy"
        },
        "firstName": {
          "content": "Claude"
        },
        "birthDate": {
          "content": "1862-10-22"
        },
        ...
      }
    }
  ]
}
```

  See also

  Insert operation [p 1152]

  Update operation [p 1156]

- **Record table to deleted**

  Defines one or more records. This structure is used upon deleting several records from the same table.

```
{
  "rows": [
    {
      "details": "http://.../root/table/1"
    },
    {
      "details": "http://.../root/table/2"
    },
    {
      "primaryKey": "./oid=3"
    },
    {
      "foreignKey": "4"
    },
    ...
  ]
}
```

  See also *Delete operation* *[p 1158]*

- **Field**

  Specifies the target field content. This request is used for field update.

  The request has the same structure as defined in node value [p 1218] by only keeping the `content` entry. Additional entries are simply ignored.

  See also *Update operation* *[p 1156]*

- **Open or close user interface**

  Specifies whether the user interface is open or close and the unavailability message.

```
{
  "content": {
    "toolStatus": {
      "content": true    // or false
    },
    "toolStatusCloseMessage": {
      "content": "Access is temporarily forbidden for maintenance."
    }
  }
}
```

  See also *User interface operations* *[p 1134]*

Only writable fields can be mentioned in the request, this excludes the following cases:

- Association node,
- Selection node,
- Value function,
- JavaBean field that does not have a setter,
- Unwritable permission on node for authenticated user.

JSON Response body

The response body is represented by a JSON `Object` whose content depends on the operation and the category.

Data category

The selection operation contains two different parts.

The first one named `meta` contains the exhaustive structure of the response.

The second, regrouping `content`, `rows`, `pagination`... etc, contains the values corresponding to the request.

- **Dataset tree**

  Contains the hierarchy of `table` and non-terminal `group` nodes.


```
{
  "meta": {
    "fields": [
      {
        "name": "rootName",
        "label": "Localized label",
        "description": "Localized description",
        "type": "group",
        "pathInDataset": "/rootName",
        "fields": [
          {
            "name": "settings",
            "label": "Settings",
            "type": "group",
            "pathInDataset": "/rootName/settings",
            "fields": [
              {
                "name": "settingA",
                "label": "A settings label",
                "type": "group",
                "pathInDataset": "/rootName/settings/settingA"
              },
              {
                "name": "settingB",
                "label": "B settings label",
                "type": "group",
                "pathInDataset": "/rootName/settings/settingB"
              }
            ]
          },
          {
            "name": "table1",
            "label": "Table1 localized label",
            "type": "table",
            "minOccurs": 0,
            "maxOccurs": "unbounded",
            "pathInDataset": "/rootName/table1"
          },
          {
            "name": "table2",
            "label": "Table2 localized label",
            "type": "table",
            "minOccurs": 0,
            "maxOccurs": "unbounded",
            "pathInDataset": "/rootName/table2"
          }
        ]
      }
    ]
  },
  "validation": [
    {
      "level": "error",
      "message": "Value must be greater than or equal to 0.",
      "details": "http://.../rootName/settings/settingA/settingA1?includeValidation=true"
    },
    {
      "level": "error",
      "message": "Field 'Settings A2' is mandatory.",
      "details": "http://.../rootName/settings/settingA/settingA2?includeValidation=true"
    }
  ],
  "content": {
    "rootName": {
      "details": "http://.../rootName",
      "openApiDetails": "http://.../api/.../rootName",
      "content": {
        "settings": {
```

```
          "details": "http://.../rootName/settings",
          "openApiDetails": "http://.../api/.../rootName/settings",
          "content": {
            "weekTimeSheet": {
              "details": "http://.../rootName/settings/settingA",
              "openApiDetails": "http://.../api/.../rootName/settings/settingA"
            },
            "vacationRequest": {
              "details": "http://.../rootName/settings/settingB",
              "openApiDetails": "http://.../api/.../rootName/settings/settingB"
            }
          }
        },
        "table1": {
          "details": "http://.../rootName/table1",
          "openApiDetails": "http://.../api/.../rootName/table1",
          "boDetails": "http://.../data-bo/.../rootName/table1"
        },
        "table2": {
          "details": "http://.../rootName/table2",
          "openApiDetails": "http://.../api/.../rootName/table2",
          "boDetails": "http://.../data-bo/.../rootName/table2"
        }
      }
    }
  }
}
```

The `meta` and `validation` properties are optional.

See also

Metamodel *[p 1205]*

Validation report *[p 1214]*

Select operation *[p 1139]*

- **Dataset node**

  Contains the list of terminal nodes under the specified node.

```
{
  "meta": {
    "fields": [
      {
        "name": "nodeName1",
        "label": "Localized label of the field node 1",
        "description": "Localized description",
        "type": "boolean",
        "minOccurs": 1,
        "maxOccurs": 1,
        "pathInDataset": "/rootName/.../nodeName1"
      },
      {
        "name": "nodeName2",
        "label": "Localized label of the field node 2",
        "type": "int",
        "minOccurs": 1,
        "maxOccurs": 1,
        "pathInDataset": "/rootName/.../nodeName2"
      }
    ]
  },
  "content": {
    "nodeName1": {
      "content": true
    },
    "nodeName2": {
      "content": -5,
      "validation": [
        {
          "level": "error",
          "message": "Value must be greater than or equal to 0."
        }
      ]
    }
  }
}
```

See also *Select operation* *[p 1139]*

- **Datasets List**

  JSON `Object` containing the list of root or children datasets for respectively the specified dataspace or dataset.

  Every dataset JSON `Object` has the following properties:

JSON property	JSON format	Description	Required
`key`	`String`	Specifies the dataset formatted key. Format: `[a-zA-Z_][-a-zA-Z0-9_.]*`	Yes
`label`	`String`	Documentation label in the current locale. It is only present if the value is different from the key.	No
`isSelectAllowed`	`Boolean`	Specifies whether the dataset can be selected, according to the user's permissions.	Yes
`details`	`String`	Specifies the dataset's REST resource URL.	No
`information`	`String`	Specifies the dataset's information REST resource URL.	No
`hasChildren`	`Boolean`	Specifies whether the dataset has children, according to the user's permissions.	Yes
`children`	`String`	Specifies the dataset's children REST resource URL. It is only present when the `hasChildren` property key's value is `true` and a details link is requested.	No
`childrenHistory`	`String`	Specifies the history dataset's children REST resource URL. It is only present when the `hasChildren` property key's value is `true` and a details link is requested. **See also** *`includeHistory` [p 1141]*	No

```
{
  "rows": [
    {
      "key": "mydataset",
      "isSelectAllowed": true,
      "details": "http://.../data/v1/BReference/mydataset",
      "information": "http://.../data/v1/BReference/mydataset:information",
      "hasChildren": true,
      "children": "http://.../data/v1/BReference/mydataset:children"
    },
    {
      // Another dataset
    }
  ],
  "pagination": {
    "firstPage": "http://.../data/v1/BReference?pageSize=4",
    "nextPage": "http://.../data/v1/BReference?pageSize=4&firstElementIndex=4",
    "previousPage": null,
    "lastPage": "http://.../data/v1/BReference?pageSize=4&firstElementIndex=8"
  }
}
```

See also *Select datasets* *[p 1173]*

- **Dataset information**

JSON `Object` containing the following properties:

JSON property	JSON format	Description	Required
`content`	`Object`	Corresponds to the dataset information.	Yes
`key`	`String`	Specifies the dataset formatted key value. Format: `[a-zA-Z_][-a-zA-Z0-9_.]*`, limited to 64 characters maximum.	Yes
`documentation`	`Array` of JSON `Object`	Corresponds to the localized documentation with a JSON `Object` by locale.	No
`locale` (nested under the `documentation` property)	`String`	Documentation locale.	Yes
`label` (nested under the `documentation` property)	`String`	Documentation label.	No
`description` (nested under the `documentation` property)	`String`	Documentation description.	No
`dataModel`	JSON `Object`	Specifies the dataset schema location. Only packaged in module schemas and publications are supported.	Yes
`module` (nested under the `dataModel` property)	`String`	Specifies the module name where the dataset schema is located.	No
`pathInModule` (nested under the `dataModel` property)	`String`	Specifies the path in the module where the dataset schema is located.	No
`publication` (nested under the `dataModel` property)	`String`	Specifies the publication name of the dataset schema.	No
`dataModel` (nested under the `dataModel` property)	`String`	Specifies the data model name of the dataset schema publication.	No
`versionName` (nested under the `dataModel` property)	`String`	Specifies the version name of the dataset schema publication.	No

JSON property	JSON format	Description	Required
`parent`	`String`	Specifies the parent dataset formatted key value. If there is no parent, then it holds a `null` value.	Yes
`activated`	`Boolean`	Specifies whether the dataset is activated.	Yes
`owner`	`String`	Specifies the owner profile.	Yes
`creator`	`String`	Specifies the creator profile.	Yes
`creation_time`	`Date`	Specifies the creation time.	Yes
`updater`	`String`	Specifies the updater profile.	Yes
`update_time`	`Date`	Specifies the update time.	Yes

```
{
  "content": {
    "key": "mydataset",
    "documentation": [
      {
        "locale": "en-US",
        "label": "My label",
        "description": "My description"
      },
      {
        "locale": "fr-FR",
        "label": "Mon libellé",
        "description": "Ma description"
      }
    ],
    "dataModel": {
       "module": "modulename",
       "pathInModule": "/WEB-INF/ebx/schemas/mydataset.xsd"
    },
    "parent": null,
    "activated": true,
    "owner": "Beveryone",
    "creator": "Uadmin",
    "creation_time": "2023-10-10T15:49:03.023",
    "updater": "Uadmin",
    "update_time": "2023-10-10T15:49:03.005"
  }
}
```

See also *Select datasets* *[p 1173]*

- **Table**

  JSON `Object` containing the following properties:

  - (Optional) The table metamodel [p 1205],
  - (Optional) The sort criteria applied,
  - (Optional) The table validation report,
  - The `rows` property corresponding to a JSON `Array` of the selected records. Each record is represented by an JSON `Object`. If no record is selected then the JSON `Array` is empty.
  - (Optional) the `pagination` property, containing pagination [p 1236] data.

```
{
  "rows": [
    {
```

```
      "label": "Claude Levi-Strauss",
      "details": "http://.../root/individu/1",
      "content": {
        "id": {
          "content": 1
        },
        ...
      }
    },
    {
      "label": "Sigmoud Freud",
      "details": "http://.../root/individu/5",
      "content": {
        "id": {
          "content": 2
        },
        ...
      }
    },
    ...
    {
      "label": "Alfred Dreyfus",
      "details": "http://.../root/individu/10",
      "content": {
        "id": {
          "content": 30
        },
        ...
      }
    }
  ],
  "sortCriteria": [
    {
      "path": "/name",
      "order": "lasc"
    },
    ...
  ],
  "pagination": {
    "firstPage": null,
    "previousPage": null,
    "nextPage": "http://.../root/individu?pageRecordFilter=./id=9&pageSize=9&pageAction=next",
    "lastPage": "http://.../root/individu?pageSize=9&pageAction=last"
  }
}
```

See also *Select operation* *[p 1139]*

- **Record**

  JSON `Object` containing:

  - The label,
  - (Optional) The record URL,
  - (Optional) The technical data [p 1240],
  - (Optional) The table metamodel [p 1205],
  - (Optional) The record validation report,
  - (Optional) The inheritance mode of the record, which can be: `root`, `inherit`, `overwrite` or `occult`.

    See also

    Record lookup mechanism *[p 482]*

    Inheritance *[p 1164]*

  - The record content.

```
{
  "label": "Name1",
  "details": "http://.../rootName/table1/pk1",
  "creationDate": "2015-02-02T19:00:53.142",
```

```
  "creationUser": "admin",
  "lastUpdateDate": "2015-09-01T17:22:24.684",
  "lastUpdateUser": "admin",
  "inheritanceMode": "root",
  "meta": {
    "name": "table1",
    "label": "Table1 localized label",
    "type": "table",
    "minOccurs": 0,
    "maxOccurs": "unbounded",
    "primaryKeys": [
      "/pk"
    ],
    "inheritance": "true",
    "fields": [
      {
        "name": "pk",
        "label": "Identifier",
        "type": "string",
        "minOccurs": 1,
        "maxOccurs": 1,
        "pathInRecord": "pk",
        "filterable": true,
        "sortable": true
      },
      {
        "name": "name",
        "label": "Name",
        "type": "string",
        "minOccurs": 1,
        "maxOccurs": 1,
        "pathInRecord": "name",
        "filterable": true,
        "sortable": true
      },
      {
        "name": "name-fr",
        "label": "Nom",
        "type": "string",
        "minOccurs": 1,
        "maxOccurs": 1,
        "inheritedField": {
          "sourceNode": "./name"
        },
        "pathInRecord": "name-fr",
        "filterable": true,
        "sortable": true
      },
      {
        "name": "parent",
        "label": "Parent",
        "description": "Localized description.",
        "type": "foreignKey",
        "minOccurs": 1,
        "maxOccurs": 1,
        "foreignKey": {
          "tablePath": "/rootName/table1",
          "details": "http://.../rootName/table1"
        },
        "enumeration": "foreignKey",
        "pathInRecord": "parent",
        "filterable": true,
        "sortable": true
      }
    ]
  },
  "content": {
    "pk": {
      "content": "pk1"
    },
    "name": {
      "content": "Name1"
    },
    "name-fr": {
      "content": "Name1",
      "inheritedFieldMode": "inherit"
    },
    "parent": {
      "content": null,
      "selector":"http://.../rootName/table1?selector=true",
      "validation": [
        {
          "level": "error",
          "message": "Field 'Parent' is mandatory."
        }
```

```
      ]
    }
  },
  "validation": {
    ...
  }
}
```

See also

Select operation [p 1139]

Prepare operations [p 1149]

- **Fields**

  For association or selection nodes, contains the target table with the associated records if, and only if, the `includeDetails` parameter is set to `true`.

  For other kinds of nodes, contains the current node value [p 1218].

  See also *Select operation [p 1139]*

- **Retrieve the user interface state**

  Contains the user interface status and the unavailability message.

```
{
  "content": {
    "toolStatus": {
      "content": true,
      "label": "Open",
      "selector": "http://.../domain/toolStatus/toolStatus?selector=true"
    },
    "toolStatusCloseMessage": {
      "content": "Access is temporarily forbidden for maintenance."
    }
  }
}
```

See also *User interface operations [p 1134]*

> **Note**
>
> Nodes, records and fields, property and values may be hidden depending on their resolved permissions *(see permissions [p 487])* .

176.3 Metamodel

This section can be activated on demand with the `includeMetamodel` [p 1142] parameter. It describes the structure and the JSON typing of the `content` section.

This section is deactivated by default for selection operations.

See also

Select operation [p 1139]

Prepare operations [p 1149]

Structure for table

Table metamodel is represented by a JSON `Object` with the following properties:

JSON property	JSON format	Description	Required
`name`	`String`	Specifies the name of the authorized table defined in the model.	Yes
`label`	`String`	Specifies the table's label. If undefined, the name of the schema node is returned.	Yes
`description`	`String`	Specifies the table's description.	No
`type`	`String`	Specifies the node's type, the value is always equal to: `table`.	Yes
`minOccurs`	`Number`	Specifies the number of minimum authorized record(s).	Yes
`maxOccurs`	`Number` or `String`	Specifies the number of maximum authorized record(s) or `unbounded`.	Yes
`history`	`Boolean`	Specifies if the table content is historized. Its value is `true` if history is activated, `false` otherwise. **See also** *History* [p 461]	No
`primaryKeyFields`	`Array`	Specifies the primary key composition which is represented as an array of the paths.	Yes
`inheritance`	`Boolean`	Specifies whether the dataset inheritance is activated for the table. Its value is `true` if inheritance is activated, `false` otherwise. **See also** *Inheritance and value resolution* [p 480]	No
`fields`	`Array`	Specifies the direct children of the record node which are represented as an array of fields. Each field may also recursively contain sub-fields.	Yes

Structure for field

Each authorized field is represented by a JSON `Object` with the following properties:

JSON property	JSON format	Description	Required
`name`	`String`	Specifies the name of the authorized field defined in the model.	Yes
`label`	`String`	Specifies the node's label. If undefined, the name of the schema node is returned.	Yes
`description`	`String`	Specifies the field's description.	No
`type`	`String`	Specifies the field's type, which can be a: simple type [p 1219], `group`, `table`, foreignKey [p 1207], association [p 1208], etc.	Yes
`minOccurs`	`Number`	Specifies the number of minimum authorized occurrence(s).	Yes
`maxOccurs`	`Number` or `String`	Specifies the number of maximum authorized occurrence(s) or `unbounded`.	Yes
`readOnly`	`Boolean`	Specifies whether the field access permission is defined as read-only from the data model access properties. That is, if its data can be read or written. Its value is `true` if the field is read only, `false` otherwise. **See also** *Access properties [p 961]* *`AccessPermission`*[API]	Yes
`autoIncrement`	`Boolean`	Specifies whether the field is auto-incremented. This property is only available for integer field type. Its value is `true` if the field is auto-incremented, `false` otherwise. **See also** *Auto-incremented values [p 953]*	No
`checkNullInput`	`Boolean`	Specifies whether the check of the constraints on `null` fields, at user input, is activated or not. Its value is `true` when activated, `false` otherwise. The constraint check is activated when the field has the property `osd:checkNullInput` [p 948] set to `true`.	Yes
`inheritedField`	`Object`	Holds information related to the inherited field's value source. `{ "inheritedField": { "sourceRecord": "/path/to/record", // (optional) "sourceNode": "./path/to/Node" } }` **See also** *Inheritance and value resolution [p 480]*	No
`foreignKey`	`Object`	Holds information related to the target table.	No (*)

JSON property	JSON format	Description	Required
		See also *Structure for foreign key field* [p 1209]	
`association`	`Object`	Holds information related to the association table. See also *Structure for the association field* [p 1210]	No (*)
`enumeration`	`String`	Specifies if the field is an enumeration value. Possible values are: • `dynamic` • `foreignKey` • `nomenclature` • `programmatic` • `resource` • `static` See also *`SchemaFacetEnumeration`*[API] According to its value, indicates if the field can be used for retrieving possible values by using the `selector` [p 1219] request parameter.	No
`enumerationDependencies`	`Array` of JSON `Object`	Specifies the metadata enumeration dependencies list. Only available for enumeration types among: `foreignKey`, `dynamic` or `programmatic`. See also *Structure for the enumeration dependencies* [p 1212]	No
`valueFunction`	`Boolean`	Specifies if the field is a computed value. See also *Computed values* [p 951]	No
`linkedField`	`Object`	Holds information related to the linked field. See also ***SchemaNode.getSchemaLinkedField()*** *`SchemaNode.getSchemaLinkedField`*[API] *Structure for the linked field* [p 1211]	No
`pathInDataset`	`String`	Specifies the relative field's path starting from the schema node.	No (**)
`pathInRecord`	`String`	Specifies the relative field's path starting from the table node.	No (*)
`filterable`	`Boolean`	Specifies whether the field can be used to filter records using the `filter` [p 1144] request parameter.	No (*)
`hiddenFilterPolicy`	`String`	Specifies the hidden filter policy. The possible values are: `textSearchOnly`, `true` or `false`.	No (*)

JSON property	JSON format	Description	Required
		See also *Hiding a field in structured search tools* [p 964] `SchemaNodeDefaultView.getHiddenInSearch`[API]	
`sortable`	`Boolean`	Specifies whether the field can be used in sort criteria using the `sort` [p 1145] request parameter.	No (*)
`constraints`	`Array` of JSON `Object`	Specifies the field's non-programmatic constraints. See also *Constraints* [p 1214]	No
`fields`	`Array` of JSON `Object`	Holds the structure and typing of each field group.	No
`isBusinessObjectRelation`	`Boolean`	Specifies whether the field structure is in the current table or given through a business object.	No

(*) Only available for table, record and record field operations.

(**) Only available for dataset tree operations.

Structure for foreign key field

The foreign key field metamodel is represented by a JSON `Object`.

```
{
  "dataspace": "BAuthors",
  "dataset": "Authors",
  "tablePath": "/root/Authors",
  "details": "http://.../BAuthors/Authors/root/Authors"
}
```

It holds the following properties:

JSON property	JSON format	Description	Required
`dataspace`	`String`	Specifies the target dataspace or snapshot identifier. If not specifies then it corresponds to the same dataspace as the table's one.	No
`dataset`	`String`	Specifies the target dataset's identifier. If not specifies then it corresponds to the same dataset as the table's one.	No
`tablePath`	`String`	Specifies the target table's path.	Yes
`details`	`String`	Specifies the target table's REST resource URL. See also `includeDetails` parameter [p 1141]	No
`historyDetails`	`String`	Specifies the history target table's REST resource URL. See also `includeDetails` parameter [p 1141]	No

Structure for the association field

The association field metamodel is represented by a JSON `Object`. This object holds properties which depend on the association type:

- `tableRefInverse`

```
{
  "type": "tableRefInverse",
  "dataspace": "BTitles",
  "dataset": "Titles",
  "tablePath": "/root/Titles",
  "details": "http://.../BTitles/Titles/root/Titles",
  "fieldToSource": "/root/Titles/au_id",
  "filter": "./unit_price > 10"
}
```

- `linkTable`

```
{
  "type": "linkTable",
  "tablePath": "/root/Inventory",
  "details": "http://.../BInventory/Inventory/root/Inventory",
  "linkTablePath": "/root/Inventory",
  "fieldToSource": "/root/Inventory/store",
  "fieldToTarget": "/root/Inventory/item",
  "filter": "./price > 10"
}
```

- `xpathLink`

```
{
  "type": "xpathLink",
  "tablePath": "/root/Inventory",
  "details": "http://.../BAuthors/Authors/root/Inventory",
  "predicate": "/root/Inventory[./price < 100]",
  "filter": "./price > 10"
}
```

It holds the following properties:

JSON property	JSON format	Description	Required
`type`	`String`	Specifies the association type. The possible values are: `tableRefInverse`, `linkTable`, `xpathLink` **See also** *Associations* [p 923]	Yes
`dataspace`	`String`	Specifies the target dataspace or snapshot identifier. If not specifies, then it corresponds to the same dataspace as the table's one.	No
`dataset`	`String`	Specifies the target dataset's identifier. If not specifies, then it corresponds to the same dataset as the table's one.	No
`tablePath`	`String`	Specifies the path of the target table.	Yes
`details`	`String`	Specifies the target table's REST resource URL. **See also** *`includeDetails` parameter* [p 1141]	No
`fieldToSource`	`String`	Specifies the field which refers to the source table of the association. Defined if the association type is `tableRefInverse` or `linkTable`.	No
`fieldToTarget`	`String`	Specifies the path of the field to the target table. Defined if the association type is `linkTable`.	No
`linkTablePath`	`String`	Specifies the path of the link table. Defined if the association type is `linkTable`.	No
`predicate`	`String`	Specifies the criteria of the association, relative to the current node. Defined if the association type is `xpathLink`.	No
`filter`	`String`	Specifies the XPath predicate expression to filter the associated objects.	No

Structure for the linked field

The linked field metamodel is represented by a JSON `Object`.

```
{
  "relationshipField": "/au_id",
  "tablePath": "/root/Authors",
  "linkedFieldPath": "/birth_date",
  "details": "http://.../BAuthors/Authors/root/Authors"
}
```

It holds the following properties:

JSON property	JSON format	Description	Required
`relationshipField`	`String`	Specifies the field's path holding the relationship used by this linked field. **See also** *`SchemaLinkedField.getRelationshipNode`*[API]	Yes
`tablePath`	`String`	Specifies the table's path referred by this linked field.	Yes
`linkedFieldPath`	`String`	Specifies the path referred by this linked field. **See also** *`SchemaLinkedField.getLinkedField`*[API]	Yes
`details`	`URI`	Specifies the target table's REST resource URL. **See also** *`includeDetails` parameter* [p 1141]	No

See also

Linked fields [p 933]

`SchemaLinkedField`[API]

Structure for the enumeration dependencies

The metamodel enumeration dependency object is represented by a JSON `Object`.

```
{
  "localModify": false,
  "dataspace": "BAuthors",
  "dataset": "Authors",
  "targetPath": "/root/Authors",
  "details": "http://.../BAuthors/Authors/root/Authors"
}
```

It holds the following properties:

JSON property	JSON format	Description	Required
`localModify`	`Boolean`	Specifies whether the enumeration dependency field is impacted by modifications performed on a field in the same record.	Yes
`dataspace`	`String`	Specifies the target dataspace or snapshot identifier of the enumeration dependency. If not specified then it corresponds to the same dataspace as the record's one.	No
`dataset`	`String`	Specifies the target dataset's identifier of the enumeration dependency. If not specifies then it corresponds to the same dataset as the record's one.	No
`targetPath`	`String`	Specifies the target path of the enumeration dependency.	Yes
`details`	`String`	Specifies the target REST resource table URL of the enumeration dependency. **See also** *`includeDetails` parameter* *[p 1141]*	No

176.4 Sort criteria

The sort criteria, applied to the request, can be returned on demand, by using the `includeSortCriteria` [p 1142] parameter (deactivated by default). If it is activated, a `sortCriteria` property is directly added to the response root node.

A `sortCriteria` property is represented as a JSON `Array` that contains ordered sort criterion. Each sort criterion corresponds to a JSON `Object` with the following properties:

JSON property	JSON format	Description	Required
`path`	`String`	Field's path.	Yes
`order`	`String`	Possible values are: `asc`, `lasc`, `desc` or `ldesc`.	Yes

The `sortByRelevancy` property is represented by a JSON `Array`, itself holding a JSON `Object` with the following properties:

JSON property	JSON format	Description	Required
`order`	`String`	Defines whether the sort is done in ascending or descending order. Possible values are: `lasc` or `ldesc`. See sortByRelevancy [p 1146] for more information.	Yes

176.5 Validation report

The validation can be activated on demand with the `includeValidation` [p 1143] parameter (deactivated by default). If it is activated, `validation` properties are directly added on target nodes with one or several messages. For messages without target node's path, a `validation` property is added on the root node.

A `validation` property is represented by a JSON `Array`, holding a JSON `Object` per message, corresponding to a validation item, with the following properties:

JSON property	JSON format	Description	Required
`level`	`String`	Severity level of the validation item. The possible values are: `info`, `warn`, `error`, `fatal`.	Yes
`message`	`String`	Description of the validation item.	Yes
`details`	`String` corresponding to an absolute URL.	URL of the resource associated with the validation item. Only available on table and dataset scopes, if the associated resource exist and if it is included. **See also** *`includeDetails` parameter* *[p 1141]*	No

176.6 Constraints

This section is automatically activated when the node has a read-write permission. It provides the non-programmatic constraints declared on the data model.

The `constraints` property is represented by a JSON `Array`. Every constraint is taking the form of a JSON `Object`. They have the following properties:

JSON property	JSON format	Description	Required
`type`	`String`	Specifies the type of the constraint **See also** *Constraint Types* *[p 1216]*	Yes
`level`	`String`	Specifies the severity level of the constraint. The possible values are: `fatal`, `error`, `warning` or `info`.	Yes
`content`	`Object`	Specifies the value of the non-programmatic constraint. This value can be a simple type, like a `String`, or a JSON `Object`.	No
`min`	`Integer`	Specifies the minimum value for `excludeSegment` [p 1216] constraint type.	No
`max`	`Integer`	Specifies the maximum value for `excludeSegment` [p 1216] constraint type.	No
`message`	`String`	Specifies the validation message returned when the value of the field does not comply with the constraint.	Yes
`blocksCommit`	`String`	Specifies the control policy management for blocking or non-blocking constraint. See Blocking and non-blocking constraints [p 945] for more information.	No

Constraint Types

This section lists the non-programmatic constraints types.

Type	Description
`mandatory`	Defines the field is mandatory
`fixedLengthStatic`	Defines the exact number of characters required for this field.
`fixedLengthDynamic`	Points out a field which provides the exact number of characters required for this one.
`minLengthStatic`	Defines the minimum number of characters allowed for this field.
`minLengthDynamic`	Points out a field which provides the minimum number of characters allowed for this one.
`maxLengthStatic`	Defines the maximum number of characters allowed for this field.
`maxLengthDynamic`	Points out a field which provides the maximum number of characters allowed for this field.
`excludeSegment`	Defines an inclusive range of not allowed values for this field.
`excludeValue`	Defines a list of not allowed values for this field.
`fractionDigits`	Defines the maximum number of decimal allowed for this field.
`pattern`	Defines the regular expression pattern that must be match by the field's value.
`totalDigits`	Defines the maximum number of digits allowed for this integer or decimal field.
`boundaryMinInclusiveStatic`	Defines the minimum value allowed (including the minimum value) for this field.
`boundaryMinExclusiveStatic`	Defines the minimum value allowed (excluding the minimum value) for this field.
`boundaryMinInclusiveDynamic`	Defines a field that provide the minimum value allowed (including the minimum value) for this field.
`boundaryMinExclusiveDynamic`	Defines a field that provide the minimum value allowed (excluding the minimum value) for this field.
`boundaryMaxInclusiveStatic`	Defines the maximum value allowed (including maximum value) for this field.
`boundaryMaxExclusiveStatic`	Defines the maximum value allowed (excluding the maximum value) for this field.
`boundaryMaxInclusiveDynamic`	Defines a field that provide the maximum value allowed (including the maximum value) for this field.

Type	Description
`boundaryMaxExclusiveDynamic`	Defines a field that provide the maximum value allowed (excluding the maximum value) for this field.

See also

XML Schema supported facets *[p 935]*

Extended facets *[p 939]*

176.7 Content

This section can be deactivated on demand with the `includeContent` [p 1141] parameter (activated by default). It provides the content of the record values, dataset, or field of one of the content fields for an authorized user. It also has additional information, including labels, technical information, URLs...

The content is represented by a JSON `Object` with a property set for each sub-node.

Node value

JSON property	JSON format	Description	Required
`content`	Content of simple type [p 1219] Content of group and list [p 1220]	Contains the node value. Available for all nodes except for `association` and `selection`. However, their content can be retrieved by invoking the URL provided in their `details` property.	`No`
`details`	`String` corresponding to an absolute REST resource URL.	Returns the node details when invoked. The response type depends on the meta type. • `foreignKey`: target record (available on table, record and field operations). • `resource`: target resource [p 940] (available on dataset node, table, record and field operations). • `association`: target table containing the associated records (available on table and record operations). • `selection`: target table containing the associated records (available on table and record operations). • `group`: target dataset group node (available on dataset tree operation). • `table`: target table (available on dataset tree operation). Example: `http://.../BReference/dataset/root/table/pk/associationField`	`No`
`historyDetails`	`String` corresponding to an absolute REST resource URL.	Returns the node history details when invoked. **See also** *`includeHistory`* *[p 1141]*	`No`
`label`	`String`	Contains the foreign key or enumeration label in the current locale. The default label is returned if the current locale is not supported.	`No`
`inheritanceMode`	`String`	Contains the node's inheritance state, considering only dataset inheritance. `inheritedFieldMode` and `inheritanceMode` properties cannot be both defined on the same node. **See also** *`inheritedFieldMode`* *[p 1218]* *Record lookup mechanism* *[p 482]*	`No`
`inheritedFieldMode`	`String`	Contains the node's field inheritance state, considering dataset and field inheritance. When both inheritances are used, field inheritance has priority over the dataset one. `inheritedFieldMode` and `inheritanceMode` properties cannot be both defined on the same node. **See also**	`No`

JSON property	JSON format	Description	Required
		`inheritanceMode` [p 1218] *Value lookup mechanism* [p 483]	
`selector`	`String` corresponding to an absolute URL.	Contains the appropriate URL for selector [p 1219] operation. Example: `http://.../BReference/dataset/root/table/pk/enumField?selector=true`	`No`
`validation`	`Array`	Contains the validation report that concerns the current node context. **See also** *Validation report* [p 1214]	`No`

Content of simple type

A simple field value is stored in a JSON `Object` and its content is the value of the `content` property.

See also *Simple types formats* [p 1238]

Content of group and list

XML Schema	JSON format	Examples	Meta type
Group `xs:complexType`	`Object` Contains a property per sub-node.	Example for a simple-occurrence group. `{ "road": { "content": "11 rue scribe" }, "zipcode": { "content": "75009" }, "country": { "content": "France" } }`	`group`

List
`maxOccurs > 1`
	`Array`
	Contains an array of all field occurrences represented by a JSON `Object`.
	Each object is represented as a node value [p 1218].
		Example for a multi-occurrence field of `xs:int` type.
		`[ { "content": 0 }, { "content": 1 }, { "content": 2 }, { "content": 3 } ]`
		Example for a multi-occurrence group.
		`[ { "content": { "road": { "content": "11 rue scribe" }, "zipcode": { "content": "75009" }, "country": { "content": "France" } } }, { "content": { "road": { "content": "711 Atlantic Ave" }, "zipcode": { "content": "MA 02111" }, "country": { "content": "United States" } } } ]`
			Meta of simple type [p 1238],
			or
			`group`

Selector

By invoking the URL represented by the `selector` property on a field that provides an enumeration, it returns a JSON `Object` containing the following properties:

- `rows` corresponding to an `Array` of JSON `Object` where each one contains two entries: the returned `content` that can be persisted and the corresponding `label`. The list of established possible items depends on the current context.
- (Optional) `pagination` containing pagination [p 1236] data (activated by default).

```
{
  "rows": [
    {
      "content": "F",
      "label": "feminine"
    },
    {
      "content": "M",
      "label": "masculine"
    }
  ],
  "pagination": {
    "firstPage": null,
    "nextPage": null,
    "previousPage": null,
    "lastPage": null
  }
}
```

See also *includeSelector* *[p 1142]*

CHAPTER 177

Compact

This chapter contains the following topics:

1. Introduction
2. Global structure
3. Content

177.1 Introduction

The JSON compact format purpose is to retrieve the master data using a lightweight structure. It follows the key-value design to display data in the simplest way and without any technical information. To activate the compact format, the `compact` suffixed REST category, like `data-compact` or `form-data-compact`, must be used in URL.

177.2 Global structure

JSON Request body

The request body is represented by a JSON `Object` whose content varies according to the operation and the category.

Data category

- **Dataset node**

  Specifies the target values of terminal nodes under the specified node. This request is used for the dataset node update operation.

```
{
  "nodeName1": true,
  "nodeName2": 2,
  "nodeName3": "Hello"
}
```

  To ensure a JSON symmetric structure between the HTTP request and the HTTP response, enumeration node request format is like the following:

```
{
  "enumerationNode": {
    "key": "a key value"
  }
}
```

See also *Update operation* *[p 1156]*

- **Record**

  Specifies the target record content by setting the value of each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is used for table record insert, record update or selector's select operation when local enumeration dependency field values are declared.

```
{
  "gender": {
    "key": "M"
  },
  "lastName": "Chopin",
  "lastName-en": "Chopin",
  "firstName": "Fryderyk",
  "firstName-en": "Frédéric",
  "birthDate": "1810-03-01",
  "deathDate": "1849-10-17",
  "jobs": [
    {
      "key": "CM"
    },
    {
      "key": "PI"
    }
  ],
  "infos": [
    "https://en.wikipedia.org/wiki/Chopin"
  ]
}
```

  See also

  Insert operation *[p 1152]*

  Update operation *[p 1156]*

- **Record fields**

  Specifies the target values of fields under the record terminal node by setting the value of each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is only used for table record updates.

```
{
  "jobs": [
    {
      "key": "CM"
    },
    {
      "key": "PI"
    }
  ]
}
```

  See also *Update operation* *[p 1156]*

- **Record table**

  Defines the content of one or more records by indicating the value of each field. For missing fields, the behavior depends on the parameter of the request `byDelta`. This structure is used upon insert or update records in the table.

```
[
  {
    "gender": {
      "key": "M"
    },
    "lastName": "Saint-Saëns",
    "firstName": "Camille",
    "birthDate": "1835-10-09",
```

```
    ...
  },
  {
    "gender": {
      "key": "M"
    },
    "lastName": "Debussy",
    "firstName": "Claude",
    "birthDate": "1862-10-22",
    ...
  }
]
```

See also

Insert operation *[p 1152]*

Update operation *[p 1156]*

- **Record table to delete**

  Defines one or more records. This structure is used upon deleting several records from the same table.

```
[
  {
    "details": "http://.../root/table/1"
  },
  {
    "details": "http://.../root/table/2"
  },
  {
    "primaryKey": "./oid=3"
  },
  {
    "foreignKey": "4"
  },
  ...
]
```

  See also *Delete operation* *[p 1158]*

- **Field**

  Specifies the target field content. This request is used upon field update.

  The request has the same structure as defined in node value [p 1230].

  See also *Update operation* *[p 1156]*

- **Open or close user interface**

  Specifies whether the user interface is open or close and the unavailability message.

```
{
  "toolStatus": {
    "key": true,     // or false
    "label": "Open"
  },
  "toolStatusCloseMessage": "Access is temporarily forbidden for maintenance."
}
```

  See also *User interface operations* *[p 1134]*

Only writable fields can be mentioned in the request, this excludes the following cases:

- Association node,
- Selection node,
- Value function,

- JavaBean field that does not have a setter,
- Unwritable permission on node for the authenticated user.

JSON Response body

The response body is represented by a JSON `Object` whose content depends on the operation and the category.

Data category

- **Dataset tree**

  Contains the hierarchy of `table` and non-terminal `group` nodes.

```
{
  "hasChildren": false,
  "rootName": {
    "settings": {
      "weekTimeSheet": {},
      "vacationRequest": {}
    },
    "table1": {},
    "table2": {}
  }
}
```

  See also *Select operation* *[p 1139]*

- **Dataset node**

  Contains the list of terminal nodes under the specified node.

```
{
  "nodeName1": true,
  "nodeName2": 2,
  "nodeName3": "Hello"
}
```

  See also *Select operation* *[p 1139]*

- **Datasets list**

  JSON `Object` containing the list of root or children datasets for the respectively specified dataspace or dataset.

  Every dataset JSON `Object` has the following properties:

JSON property	JSON format	Description	Required
`key`	`String`	Specifies the dataset formatted key. Format: `[a-zA-Z_][-a-zA-Z0-9_.]*`	Yes
`isSelectAllowed`	`Boolean`	Specifies whether the dataset can be selected, according to the user's permissions.	Yes
`hasChildren`	`Boolean`	Specifies whether the dataset has children, according to the user's permissions.	Yes

```
{
  "rows": [
    {
      "key": "mydataset",
```

```
      "isSelectAllowed": true,
      "hasChildren": true
    },
    {
      // Another dataset
    }
  ],
  "pagination": {
    "firstPage": "http://.../data/v1/BReference?pageSize=4",
    "nextPage": "http://.../data/v1/BReference?pageSize=4&firstElementIndex=4",
    "previousPage": null,
    "lastPage": "http://.../data/v1/BReference?pageSize=4&firstElementIndex=8"
  }
}
```

See also *Select datasets* *[p 1173]*

- **Dataset information**

JSON `Object` containing the following properties:

JSON property	JSON format	Description	Required
`key`	`String`	Specifies the dataset formatted key value. Format: `[a-zA-Z_][-a-zA-Z0-9_.]*`, limited to 64 characters maximum.	Yes
`dataModel`	JSON `Object`	Specifies the dataset schema location. Only packaged in module schemas and publications are supported.	Yes
`module` (nested under the `dataModel` property)	`String`	Specifies the module name where the dataset schema is located.	No
`pathInModule` (nested under the `dataModel` property)	`String`	Specifies the path in the module where the dataset schema is located.	No
`publication` (nested under the `dataModel` property)	`String`	Specifies the publication name of the dataset schema.	No
`dataModel` (nested under the `dataModel` property)	`String`	Specifies the data model name of the dataset schema publication.	No
`versionName` (nested under the `dataModel` property)	`String`	Specifies the version name of the dataset schema publication.	No
`parent`	`String`	Specifies the parent dataset formatted key value. If there is no parent, then it holds a `null` value.	Yes
`activated`	`Boolean`	Specifies whether the dataset is activated.	Yes
`owner`	`String`	Specifies the owner profile.	Yes
`creator`	`String`	Specifies the creator profile.	Yes
`creation_time`	`Date`	Specifies the creation time.	Yes
`updater`	`String`	Specifies the updater profile.	Yes
`update_time`	`Date`	Specifies the update time.	Yes

```
{
  "key": "mydataset",
  "dataModel": {
    "module": "modulename",
```

```
    "pathInModule": "/WEB-INF/ebx/schemas/mydataset.xsd"
  },
  "parent": null,
  "activated": true,
  "owner": "Beveryone",
  "creator": "Uadmin",
  "creation_time": "2023-10-10T15:49:03.023",
  "updater": "Uadmin",
  "update_time": "2023-10-10T15:49:03.005"
}
```

See also *Select datasets [p 1173]*

- **Table**

  JSON `Object` containing the following properties:

  - `rows` corresponding to JSON `Array` of selected records. Each record is represented by a JSON `Object`. If no record is selected, then the JSON `Array` is empty.
  - (Optional) `pagination` containing pagination [p 1236] data.

```
{
  "rows": [
    {
      "id": 1,
      "firstName": "Claude",
      "lastName": "Levi-Strauss"
    },
    {
      "id": 2,
      "firstName": "Sigmoud",
      "lastName": "Freud"
    },
    {
      "id": 3,
      "firstName": "Alfred",
      "lastName": "Dreyfus"
    }
  ],
  "pagination": {
    "firstPage": null,
    "previousPage": null,
    "nextPage": "http://.../root/individu?pageRecordFilter=./id=9&pageSize=9&pageAction=next",
    "lastPage": "http://.../root/individu?pageSize=9&pageAction=last"
  }
}
```

See also *Select operation [p 1139]*

- **Record**

  JSON `Object` containing the record content.

```
{
  "pk": "pk1",
  "name": "Name1",
  "name-fr": "Name1",
  "parent": null
}
```

See also

Select operation [p 1139]

Prepare operations [p 1149]

- **Fields**

  Except for selection nodes, they contain the current node value [p 1230].

See also *Select operation [p 1139]*

- **Retrieve the user interface state**

  Contains the user interface status and the unavailability message.

```
{
  "toolStatus": {
    "key": true,     // or false
    "label": "Open"
  },
  "toolStatusCloseMessage": "Access is temporarily forbidden for maintenance."
}
```

See also *User interface operations* *[p 1134]*

> **Note**
>
> Nodes, records and fields property and value may be hidden depending on their resolved permissions *(see permissions [p 487])* .

177.3 Content

This section is always included and contains master data without any additional fields.

Node value

The node value contains only the data or the label and the details link in case of enumeration. It is available for all nodes except association and selection.

Content of simple types

Corresponds to a key-value JSON entry which describes the node content.

See also *Simple types formats* *[p 1238]*

Content of group, list and enumeration

XML Schema	JSON format	Examples
Group `xs:complexType`	`Object` Contains a property per sub-node.	Example for a simple-occurrence group. `{ "road": "11 rue scribe", "zipcode": "75009", "country": "France" }`

List
`maxOccurs > 1`
	`Array`
	Contains an array of all field occurrences represented by a JSON `Object` or a simple type.
	Each JSON `Object` is composed of node values [p 1230].
		Example for a multi-occurrence field of `xs:int` type.
		`[ 0, 1, 2, 3, 4 ]`
		Example for a multi-occurrence group.
		`[ { "road": "11 rue scribe", "zipcode": "75009", "country": "France" }, { "road": "711 Atlantic Ave", "zipcode": "MA 02111", "country": "United States" } ]`

Enumeration `xs:string`	`Object` Contains `key`, `link` and `label` properties.	Example for a foreign key. `{ "key":"1", "details":"http://.../Bdataspace/dataset/root/nationality/FRA", "label":"Française" }`

CHAPTER 178

Common

This chapter contains the following topics:

1. Introduction
2. Global structure
3. Form validation report
4. Content
5. Update modes
6. Known limitations

178.1 Introduction

The common specifications are used in both compact [p 1223] and Extended [p 1193] formats.

178.2 Global structure

JSON Response body

Form data category

The response body contains a JSON `Object` per handled resources. That is, the JSON root `Object` for a single handled resource is a JSON `Object` as well as for multiple handled resources. The several properties are directly placed into the JSON `Object`.

- If the insert/update of a single record request or update field in table/dataset request was successful, the `code` JSON property may be absent and if not, it corresponds to the HTTP response code associated to the handled resource. The validation report is always present. Some properties are optional and added when requested using request parameters. See Form data operations [p 1167] for more information.

```
{
  "validation": [
    {
      "level": "error",
      "message": "Field 'Value' is mandatory.",
      "blocksCommit": "never",
      "pathInRecord": "/value"
    },
    {
      "level": "error",
      "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
      "blocksCommit": "onUserSubmit-checkModifiedValues",
```

```
      "pathInRecord": "/code"
    },
    {
      "level": "warning",
      "message": "Value 'AD150' is prohibited.",
      "blocksCommit": "never",
      "pathInRecord": "/code"
    }
  ]
}
```

- If the insert multiple record request was successful, the `code` JSON property may be absent and if not, it corresponds to the HTTP response code associated to the handled resource. The validation report is always present. Some properties are optional and added when requested using request parameters.

```
{
  "rows": [
    {
      "label": "My Lable1",
      "details": "http://.../root/table/1",
      "foreignKey": "1",
      "validation": [
        {
          "level": "error",
          "message": "Field 'Value' is mandatory.",
          "blocksCommit": "never",
          "pathInRecord": "/value"
        },
        {
          "level": "error",
          "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
          "blocksCommit": "onUserSubmit-checkModifiedValues",
          "pathInRecord": "/code"
        },
        {
          "level": "warning",
          "message": "Value 'AD150' is prohibited.",
          "blocksCommit": "never",
          "pathInRecord": "/code"
        }
      ]
    },
    {
      "label": "My Lable2",
      "details": "http://.../root/table/2",
      "foreignKey": "2",
      "validation": [
        {
          "level": "error",
          "message": "Field 'Value' is mandatory.",
          "blocksCommit": "never",
          "pathInRecord": "/value"
        },
        {
          "level": "error",
          "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
          "blocksCommit": "onUserSubmit-checkModifiedValues",
          "pathInRecord": "/code"
        },
        {
          "level": "warning",
          "message": "Value 'AD150' is prohibited.",
          "blocksCommit": "never",
          "pathInRecord": "/code"
        }
      ]
    }
  ]
}
```

- If the insert/update of a single record request or update field in table/dataset failed, the response body corresponds to a JSON `Exception handling` response.

```
{
  "code": 422,
  "errors": [
    {
      "level": "error",
      "userCode": "Validation",
      "message": "Field 'Category' is mandatory.",
```

```
      "blocksCommit": "never",
      "pathInRecord": "/category"
    },
    {
      "level": "error",
      "userCode": "Validation",
      "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
      "blocksCommit": "onUserSubmit-checkModifiedValues",
      "pathInRecord": "/code"
    }
  ]
}
```

- If the insert multiple record request failed, the response body corresponds to a JSON `Exception handling` response.

```
{
  "code": 422,
  "errors": [
    {
      "level": "error",
      "rowIndex": 0,
      "userCode": "Validation",
      "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
      "blocksCommit": "onUserSubmit-checkModifiedValues",
      "pathInRecord": "/code"
    },
    {
      "level": "error",
      "rowIndex": 0,
      "userCode": "Validation",
      "message": "Field 'Value Less Than' is mandatory.",
      "blocksCommit": "never",
      "pathInRecord": "/less_than_value"
    },
    {
      "level": "error",
      "rowIndex": 1,
      "userCode": "Validation",
      "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
      "blocksCommit": "onUserSubmit-checkModifiedValues",
      "pathInRecord": "/code"
    },
    {
      "level": "error",
      "rowIndex": 1,
      "userCode": "Validation",
      "message": "Field 'Value Less Than' is mandatory.",
      "blocksCommit": "never",
      "pathInRecord": "/less_than_value"
    }
  ]
}
```

See also

Form validation report *[p 1235]*

Exception handling *[p 1118]*

178.3 Form validation report

This report can be retrieve by using the Form data operations [p 1167]. It provides every validation constraint on the requested resource.

Those constraints have the following properties:

JSON property	JSON format	Description	Required
`message`	`String`	Description of the constraint.	Yes
`blocksCommit`	`String`	Control policy of the constraint. The possible values are: `onInsertUpdateOrDelete`, `onUserSubmit-checkModifiedValues`, `never`. See Blocking and non-blocking constraints [p 945] for more information.	Yes
`level`	`String`	Severity level of the constraint. The possible values are: `info`, `warning`, `error` or `fatal`.	Yes
`pathInDataset`	`String`	Relative field path starting from the schema node.	No(**)
`pathInRecord`	`String`	Relative field path starting from the table node.	No (*)

(*) Only available for record and record field operations.

(**) Only available for dataset operations.

See also *Form data category* *[p 1233]*

178.4 Content

Pagination

Use pagination to return a limited and parameterizable number of data. Pagination can be applied to the following data types: records, association values, selection node values, selectors, dataspaces and datasets. A context named `pagination` is always returned. This context allows you to browse data in a similar manner to the UI.

The pagination is always enabled.

See also

Select operation *[p 1139]*

Select datasets *[p 1173]*

Beta feature: Select dataspaces or snapshots *[p 1174]*

The following provides detailed information related to this context:

JSON property	JSON format	Description	Required
`firstPage`	`String` or `null` (*)	URL to access the first page.	Yes
`previousPage`	`String` or `null` (*)	URL to access the previous page.	Yes
`nextPage`	`String` or `null` (*)	URL to access the next page.	Yes
`lastPage`	`String` or `null` (*)	URL to access the last page.	Yes

> **Note**
>
> (*) Only defines whether data is available in this context and not in the response.

Content of simple type

XML Schema	JSON format	Examples	Meta type	OpenAPI

`xs:string`
`xs:Name`
`osd:text`
	`String` (Unicode characters, cf. RFC4627)	"A text"
		"The escape of \"special character\" is preceded by a backslash."
		null
			`string`
			`name`
			`text`
				type: `string`
				format: `n/a`

`osd:html`	`String` (Unicode characters, cf. RFC4627)	" An HTML tag can thus be written without trouble "	`html`	type: `string` format: `html`
`osd:email`	`String` (Unicode characters, cf. RFC4627)	"employee@mycompany.com"	`email`	type: `string` format: `email`
`osd:locale`	`String` (Language tag, cf. RFC1766)	"en-US"	`locale`	type: `string` format: `locale`
`xs:string` (Foreign key)	`String` contains the value of the formatted foreign key.	`"0"` `"true|99"`	`foreignKey`	type: `string` format: `n/a`
`xs:boolean`	`Boolean`	true false null	`boolean`	type: `boolean` format: `n/a`
`xs:decimal`	`Number` or `null`	-10.5 20.001 15 -1e-13	`decimal`	type: `number` format: `double`
`xs:date`	`String` with format: "yyyy-MM-dd"	"2015-04-13"	`date`	type: `string` format: `date`
`xs:time`	`String` with format: • "HH:mm:ss" • "HH:mm:ss.SSS"	"11:55:00" "11:55:00.000"	`time`	type: `string` format: `date-time`
`xs:dateTime`	`String` with format: • "yyyy-MM-ddTHH:mm:ss" • "yyyy-MM-ddTHH:mm:ss.SSS"	"2015-04-13T11:55:00" "2015-04-13T11:55:00.000"	`dateTime`	type: `string` format: `date-time`
`xs:anyURI`	`String` (Uniform Resource Identifier, cf. RFC3986)	"https://fr.wikipedia.org/wiki/René_Descartes"	`anyURI`	type: `string` format: `uri`

XML Schema	JSON format	Examples	Meta type	OpenAPI
`xs:int` `xs:integer`	`Number` or `null`	1596	`int`	type: `integer` format: `int32`
`osd:resource`	`String` contains the resource formatted value.	"ebx-tutorial:ext-images:frontpages/Bach.jpg"	`resource`	type: `string` format: `n/a`
`osd:color`	`String` with format: "#[A-Fa-f0-9]{6}" contains the formatted value for the color.	"#F6E0E0"	`color`	type: `string` format: `n/a`
`osd:datasetName`	`String` with format: "[a-zA-Z_][-a-zA-Z0-9_.]*" and 64 characters max. contains the formatted value of the dataset name.	"ebx-tutorial"	`dataset`	type: `string` format: `n/a`
`osd:dataspaceKey`	`String` with format: `[BV][a-zA-Z0-9_:.\-|]+` and 33 characters max. contains the formatted key value of the dataspace.	"Bebx-tutorial"	`dataspace`	type: `string` format: `n/a`

Insert operation report

When invoking the insert operation with a record table, it can optionally return a report. The report includes a JSON `Object` that contains the following properties:

- `count` contains an `int` of JSON `Number` type, that defines the number of processed records since the listed rows can not be exhaustive in several cases.
- `isPartialList` contains a `boolean` of JSON `Boolean` type, that is true if the listed rows are only a partial list of the whole processed records, false otherwise.
- `rows` contains a JSON `Array`, where each element corresponds to the result of a request element.
- `code` contains an `int` of JSON `Number` type, and allows to know whether the record has been inserted or updated. This property is included if, and only if, the `updateOrInsert` parameter is set to `true`.
- `foreignKey` contains a `string` of JSON `String` type, corresponding to the content to be used as a foreign key for this record. This property is included if, and only if, the parameter `includeForeignKey` is set to `true`.
- `label` contains a `string` of JSON `String` type, and allows to retrieve the record label. This property is included if, and only if, the parameter `includeLabel` is set to `yes`.
- `details` containing a `string` of JSON `String` type, corresponding to the resource URL. This property is included if, and only if, the parameter `includeDetails` is set to `true`.

```
{
  "count": 2,
  "isPartialList": false,
  "rows": [
    {
      "code": 204,
```

```
      "foreignKey": "62",
      "label": "Claude Debussy",
      "details": "http://.../root/individu/62"
    },
    {
      "code": 201,
      "foreignKey": "195",
      "label": "Camille Saint-Saëns",
      "details": "http://.../root/individu/195"
    }
  ]
}
```

See also *Insert operation* *[p 1152]*

Delete operation report

When invoking the delete operation, a report is returned. The report includes a JSON `Object` that contains the following properties:

- `deletedCount` containing an integer of JSON `Number` type, corresponds to the number of deleted records.
- `occultedCount` containing an integer of JSON `Number` type, corresponds to the number of occulted records.
- `inheritedCount` containing an integer of JSON `Number` type, corresponds to the number of inherited records.

```
{
  "deletedCount": 1,
  "inheritedCount": 0,
  "occultedCount": 0
}
```

See also *Delete operation* *[p 1158]*

Technical data

Each returned record can contain its system records metadata under `ebx-metadata/system` properties corresponding to the technical data. System metadata are requestable and sortable.

To include records metadata, see `includeMetadata=system` [p 1141] parameter value for select operation.

JSON property	JSON format	Description	Required
`uuid`	`String`	Universally Unique Identifier of the record	Yes
`creator`	`String`	Creation user's identifier. See `UserReference`[API] for more information.	Yes
`creation_time`	`String`	Creation date	Yes
`updater`	`String`	Last update user's identifier. See `UserReference`[API] for more information.	Yes
`update_time`	`String`	Last update date.	Yes

For extended format:

```
{
```

```
  ...,
  "ebx-metadata":{
    "content":{
      "system":{
        "content":{
          "uuid":{
            "content":"7FE03810-6A67-11ED-A892-00FF20216100"
          },
          "creator":{
            "content":"Uadmin"
          },
          "creation_time":{
            "content":"2022-11-22T14:13:47.793"
          },
          "updater":{
            "content":"Uadmin"
          },
          "update_time":{
            "content":"2022-11-29T17:46:55.686"
          }
        }
      }
    }
  }
}
```

For compact format:

```
{
  ...,
  "ebx-metadata":{
    "system":{
      "uuid":"7FE03810-6A67-11ED-A892-00FF20216100",
      "creator":"Uadmin",
      "creation_time":"2022-11-22T14:13:47.793",
      "updater":"Uadmin",
      "update_time":"2022-11-29T17:46:55.686"
    }
  }
}
```

See also *Optimistic locking* *[p 1163]*

178.5 Update modes

The `byDelta` mode allows to ignore data model elements that are missing from the JSON source document. This mode is enabled (by default) through RESTful operations. The following table summarizes the behavior of insert and update operations when elements are not included in the source document.

See the RESTful data services operations update [p 1156] and insert [p 1152], as well as `ImportSpec.setByDelta`[API] in the Java API for more information.

State in the JSON source document	Behavior
The property does not exist in the source document	**If the `byDelta` mode is activated (default):** • For the `update` operation, the field value remains unchanged. • For the `insert` operation, the behavior is the same as when the `byDelta` mode is disabled. **If the `byDelta` mode is disabled through the RESTful operation parameter:** The target field is set to one of the following values: • If the element defines a default value, the target field is set to that default value. • If the element is of a type other than a string or list, the target field value is set to `null`. • If the element is an aggregated list, the target field value is set to an empty list value. • If the element is a string that differentiates `null` from an empty string, the target field value is set to `null`. If it is a string that does not differentiate the two, an empty string. • If the element (simple or complex) is hidden in the data services, the target value remains unchanged. **See also** *Hiding a field in Data Services* *[p 964]* **Note** The user performing the import must have the required permissions to create or change the target field value. Otherwise, the operation will be aborted.
The element is present and its value is `null` (for example, `"content": null`)	The target field is always set to `null` except for lists, in which case it is not supported.

178.6 Known limitations

Field values

The value of fields `xs:date`, `xs:time` and `xs:dateTime` does not contain a time zone associated with the JSON-primitive type.

Indexing and search strategy

Filterable [p 1208] and sortable [p 1209] values from the metamodel are limited to the default search strategy.

See also *Search* *[p 511]*

CHAPTER 179

Others

This chapter contains the following topics:

1. Introduction
2. Global structure

179.1 Introduction

The Extended [p 1193] and Compact [p 1223] JSON formats can handle most of the use cases, however some operations and REST categories require specific formats like the followings.

179.2 Global structure

JSON Request body

The Request body is represented by a JSON `Object` whose content varies according to the operation and the category.

Auth category

The request body holds several properties directly placed in the root JSON `Object`.

- **Token creation**

  Specifies the `login` and `password` to use for an authentication token creation attempt.

```
{
  "login": "...",                 // JSON String
  "password": "..."               // JSON String
}
```

  Specifies the `specific` attribute, to activate the user authentication against the HTTP request, for an authentication token creation attempt.

```
{
  "specific": true                // JSON Boolean
}
```

  See also *Create token operation [p 1136]*

- **Password change**

  Specifies the `login`, `password` and `passwordNew` to use for the password change.

```
{
```

```
  "login": "...",            // JSON String
  "password": "...",         // JSON String
  "passwordNew": "..."       // JSON String
}
```

See also *Change password operation* *[p 1137]*

JSON Response body

The response body is represented by a JSON `Object` whose content depends on the operation and the category.

Admin category

The selection operation for this category only provides the requested values under a `content` property.

- **System information**

  Contains EBX® instance's system information. The representation of these data can be flat or hierarchical.

  Flat representation:

```
{
  "content": {
    "bootInfoEBX": {
      "label": "EBX® configuration",
      "content": {
        "product.version": {
          "label": "EBX® product version",
          "content": "5.8.1 [...] Enterprise Edition"
        },
        "product.configuration.file": {
          "label": "EBX® main configuration file",
          "content": "System property [ebx.properties=./ebx.properties]"
        },
        // others keys
      }
    },
    "repositoryInfo": {
      "label": "Repository information",
      "content": {
        "repository.identity": {
          "label": "Repository identity",
          "content": "00905A5753FD"
        },
        "repository.label": {
          "label": "Repository label",
          "content": "My repository"
        },
        // others keys
      }
    },
    "bootInfoVM": {
      "label": "System information",
      "content": {
        "java.home": {
          "label": "Java installation directory",
          "content": "C:\\JTools\\jdk1.8.0\\jre"
        },
        "java.vendor": {
          "label": "Java vendor",
          "content": "Oracle Corporation"
        },
        // others keys
      }
    }
  }
}
```

  Hierarchical representation:

```
{
  "content": {
    "bootInfoEBX": {
      "label": "EBX® configuration",
```

```
      "content": {
        "product": {
          "content": {
            "version": {
              "label": "EBX® product version",
              "content": "5.8.1 [...] Enterprise Edition"
            },
            "configuration": {
              "content": {
                "file": {
                  "label": "EBX® main configuration file",
                  "content": "System property [ebx.properties=./ebx.properties]"
                }
              }
            }
          }
        },
        "vm": {
          "content": {
            "startTime": {
              "label": "VM start time",
              "content": "2017/09/11-10:04:17-0729 CEST"
            },
            "identifier": {
              "label": "VM identifier",
              "content": "1"
            }
          }
        },
        // other hierarchical keys
      }
    }
  }
}
```

See also *System information operation* *[p 1134]*

Auth category

The response body contains several properties directly placed in the root JSON `Object`.

- **Token creation**

  Contains the token value and its type.

```
{
  "accessToken": "...",            // JSON String
  "tokenType": "..."               // JSON String
}
```

See also *Create token operation* *[p 1136]*

Data category

Look up table views

The response body contains a `content` property holding a JSON `Array`, itself composed by JSON `Objects` with the following properties:

JSON property	JSON format	Description	Required
`details`	`String`	Corresponds to the view access URL.	Yes
`label`	`String`	View's label.	No
`viewPublication`	`String`	Published view's name. **See also** *Table `viewPublication` parameter* [p 1147]	Yes
`viewType`	`String`	Enumeration whose value corresponds to one of the following: • `SimpleTabular`: Simple tabular view. • `Hierarchy`: Hierarchical view. • `Tile`: Tile view.	Yes

```
{
  "content": [
    {
      "details": "http://.../data/v1/Bebx-directory/ebx-directory/directory/users?viewPublication=custom-
directory",
      "label": "My custom directory view",
      "viewPublication": "custom-directory",
      "viewType": "SimpleTabular"
    },
    {
      ...
    }
  ]
}
```

See also *Look up table views operation* [p 1166]

Beta feature: Dataspaces selection

The returned response body contains dataspaces in a `rows` JSON `Array` property, where each inner JSON `Object` corresponds to a dataspace with the following properties:

JSON property	JSON format	Description	Required
`label`	`String`	Documentation label in the current locale.	No
`description`	`String`	Documentation description in the current locale.	No
`details`	`String`	Specifies the dataspace's REST resource URL.	Yes
`information`	`String`	Specifies the dataspace's information REST resource URL.	Yes
`key`	`String`	Specifies the dataspace or snapshot formatted key. Format: `[BV][a-zA-Z0-9_:.\-|]{1,33}` and percent encoded afterward.	Yes
`isSelectAllowed`	`Boolean`	Specifies if the dataspace can be selected, according to the user's permissions.	Yes
`hasChildren`	`Boolean`	Specifies if the dataspace has children, according to the user's permissions. **Note** Not applicable for **snapshots**.	Yes
`children`	`String`	Specifies the dataspace's children REST resource URL. If `hasChildren` property key value is `false` then the returned value is `null`. **Note** Not applicable for **snapshots**.	No
`snapshots`	`String`	Specifies the dataspace's snapshots REST resource URL. **Note** Not applicable for **snapshots**.	Yes

```
{
  "rows": [
    {
      "label": "Master Data - Reference",
      "description": "Reference dataspace in EBX.",
      "details": "http://.../data/v1/BReference",
      "information": "http://.../data/v1/BReference:information",
      "key": "BReference",
      "closed": false,
      "isSelectAllowed": true,
      "hasChildren": true,
      "children": "http://.../data/v1/BReference:children",
      "snapshots": "http://.../data/v1/BReference:snapshots"
    },
    {
```

```
      // An other dataspace
    }
  ],
  "pagination": {
    "firstPage": null,
    "previousPage": null,
    "nextPage": null,
    "lastPage": null
  }
}
```

See also

Pagination *[p 1236]*

Beta feature: Select dataspaces or snapshots *[p 1174]*

Beta feature: Dataspace information

The response body contains a `content` JSON `Object` property, holding the following properties:

JSON property	JSON format	Description	Required
`content`	`Object`	Corresponds to the dataspace or the snapshot information.	Yes
`key`	`String`	Specifies the dataspace or snapshot formatted key value. Format: `[BV][a-zA-Z0-9_:.\-|]+` limited to 33 characters maximum.	Yes
`documentation`	`Array` of JSON `Object`	Corresponds to the localized documentation with a JSON `Object` by locale.	No
`locale`	`String`	Documentation locale (nested under the `documentation` property).	Yes
`label`	`String`	Documentation label (nested under the `documentation` property).	No
`description`	`String`	Documentation description (nested under the `documentation` property).	No
`closed`	`Boolean`	Specifies if the dataspace is closed.	Yes
`locked`	`Boolean`	Specifies if the dataspace is locked. **Note** Not applicable for **snapshots**.	Yes
`parent`	`String`	Specifies the parent dataspace or snapshot formatted key value.	No
`administration`	`Boolean`	Specifies if the dataspace or the snapshot is an administration one.	Yes
`owner`	`String`	Specifies the owner profile.	No
`creator`	`String`	Specifies the creator profile.	Yes
`creationDate`	`Date`	Specifies the creation date.	Yes

```
{
  "content": {
    "key": "BReference",
    "documentation": [
      {
        "locale": "en-US",
        "label": "Master Data - Reference",
        "description": "Reference dataspace in EBX."
      },
      {
        "locale": "fr-FR",
        "label": "Données - Référence",
        "description": "Espace de données référence de EBX."
      }
```

```
    ],
    "closed": false,
    "locked": false,
    "parent": null,
    "administration": false,
    "relationalMode": false,
    "owner": "Badministrator",
    "creator": "Badministrator",
    "creationDate": "2019-04-28T19:49:04.838"
  }
}
```

See also *Beta feature: Select dataspaces or snapshots* *[p 1174]*

Beta feature: Dataspace child or snapshot creation

The response body contains a `content` JSON `Object` property, holding the following properties:

JSON property	JSON format	Description	Required
`key`	`String`	Specifies the dataspace or snapshot formatted key value. Format: `[BV][a-zA-Z0-9_:.\-|]+` limited to 33 characters maximum. The default value is a timestamp.	No
`owner`	`String`	Specifies the owner profile. Default value is `null`.	No
`documentation`	`Array` of JSON `Object`	Corresponds to the localized documentation with a JSON `Object` by locale. Default value is `null`.	No
`locale`	`String`	Documentation locale (nested under the `documentation` property).	Yes
`label`	`String`	Documentation label (nested under the `documentation` property). Default value is `null`.	No
`description`	`String`	Documentation description (nested under the `documentation` property). Default value is `null`.	No
`dataspaceKeyToCopyPermissionsFrom`	`String`	Specifies the dataspace's formatted key value from which to copy permissions. **Note** Not applicable for **snapshots** creation.	No

```
{
  "content": {
    "key": "BMyData",
    "owner": "Beveryone",
    "documentation": [
      {
        "locale": "en-US",
        "label": "My dataspace",
        "description": "This space contains my data"
```

```
      }
    ],
    "dataspaceKeyToCopyPermissionsFrom": "BReference"
  }
}
```

See also *Beta feature: Create a child dataspace or a snapshot [p 1176]*

Health category

Health started and check responses

The response body contains a JSON String value corresponding to an `OK` value.

Health stopped response

The response body contains a JSON String value corresponding to an enumeration value. Possible values: `OK` or `NO`

SQL in EBX®

CHAPTER **180**

Introduction

This chapter contains the following topics:

1. Overview
2. Mapping data model entities
3. Mapping data types
4. SQL syntax
5. Limitations and performance guidelines

180.1 Overview

This documentation covers Structured Query Language (SQL) queries and expressions in EBX®. EBX® supports standard SQL queries to retrieve rows selected from one or more tables. Some EBX® SQL language features are extensions to the standard. Supported EBX® SQL syntax includes: table expressions (SELECT, FROM, WHERE, GROUP BY and HAVING clauses), DISTINCT, ORDER BY, LIMIT and OFFSET, combining queries (UNION [ALL]), and WITH (including RECURSIVE modifier).

The goal of this API is to provide to developers the ability to retrieve data from EBX® using a well-known standard.

EBX® SQL is accessible through Java APIs, especially from the class `Query`[API]. The queries also support parameters. See `Query.setParameter`[API].

Queries can involve a single dataset or multiple datasets, see `QueryBuilder`[API] to know more about how to create a query.

180.2 Mapping data model entities

The following section provides a detailed explanation about the mapping of the EBX® concepts into SQL.

Table (in data model)

EBX® tables are mapped naturally to SQL tables. In the data model, there can be more than one EBX® table with the same name. This ambiguity can occur when tables are in groups. To remove the ambiguity, use the full path of the table surrounded by double quotes (for example, `"my_group/my_table"` no longer conflicts with `"other_group/my_table"`). You can also use the entity name of

the table, which is unique inside the data model. You can use the table name only if it does not collide with an entity name or another table name.

See also *Entity mappings using the Data Model Assistant* *[p 92]*

Fields

In SQL Standard, the structure of a table consists of one or more columns. Every element (including fields) whose parent is an EBX® table, is mapped to a column.

Groups

In SQL Standard, the structure of a table consists of one or more columns. Every element whose parent is an EBX® table is mapped to a column. This includes groups that are mapped to SQL columns as *SQL structure types*.

Associations

In SQL Standard, querying data among multiple tables is based on foreign keys and primary keys. These concepts in EBX® are similar to those in SQL. Therefore, joins between tables in SQL can also be done using EBX® foreign and primary keys.

180.3 Mapping data types

Handling data through SQL is highly dependent on its data type. For example, in predicates, columns can be compared only if they have the same SQL data type. The SQL data types are types according to the type in the data model.

Simple data types

Supported standard SQL data types

This table lists all of the simple types defined in the XML Schema that are supported by EBX®, along with their corresponding standard SQL types.

XML Schema type	SQL type	Java type	Notes
xs:string	VARCHAR	java.lang.String	
xs:boolean	BOOLEAN	java.lang.Boolean	Values: TRUE, FALSE, UNKNOWN
xs:decimal	DECIMAL	java.math.BigDecimal	
xs:dateTime	TIMESTAMP	java.util.Date	
xs:time	TIME	java.util.Date	The date portion of the returned Date is always set to '1970/01/01'.
xs:date	DATE	java.util.Date	The time portion of the returned Date is always set to the beginning of the day (that is, '00:00:00').
xs:anyURI	VARCHAR	java.net.URI	
xs:Name (xs:string restriction)	VARCHAR	java.lang.String	
xs:int	INT	java.lang.Integer	

Extended simple types defined by EBX®

EBX® provides pre-defined simple data types. These types are defined by the internal schema `common-1.0.xsd`. Their definition is detailed in the section Extended simple types defined by EBX® [p 902]

XML Schema type	SQL type	Java class
`osd:text` (`xs:string` restriction)	`VARCHAR`	`java.lang.String`
`osd:html` (`xs:string` restriction)	`VARCHAR`	`java.lang.String`
`osd:email` (`xs:string` restriction)	`VARCHAR`	`java.lang.String`
`osd:password` (`xs:string` restriction)	`VARCHAR`	`java.lang.String`
`osd:color` (`xs:string` restriction)	`VARCHAR`	`java.lang.String`
`osd:resource` (`xs:anyURI` restriction)	VARCHAR	internal class
`osd:locale` (`xs:string` restriction)	`VARCHAR`	`java.util.Locale`
`osd:dataspaceKey` (`xs:string` restriction)	`VARCHAR`	`java.lang.String`
`osd:datasetName` (`xs:string` restriction)	`VARCHAR`	`java.lang.String`

List (multi-valued) types

Lists are handled as SQL Arrays. Their corresponding Java class is `java.util.List`.

Note: in order to filter on a multivalued field, first it has to be "normalized" via the `UNNEST` SQL operator (be aware that this is not recommended in terms of performance and can be quite inefficient on large tables):

- `SELECT T.s FROM myTable T, UNNEST(T.list) as L(item) WHERE L.item = 'test'`
- `SELECT p.name FROM Person p, UNNEST(p.addresses) AS ad(street, city) WHERE ad.city = 'Paris'`

Complex types

Complex types are handled as SQL Structured types. Their corresponding Java class is `Object[]`. This applies to foreign keys (see below) and group [p 35]s, because they are defined through complex types. Use the *dot operator* to access fields inside the SQL Structure types. For example, use `address.street` to access the field `street` of the field `address`, if it is a complex type. When you reference a sub-field of a complex type in a query, you must always use the table name or alias:

- `SELECT customer.address.street FROM customer`
- `SELECT c.address.street FROM customer c`

TableRef types

In EBX®, a table can have a primary key composed of multiple fields. Foreign keys are defined by a single field with the `osd:tableRef` [p 919] declaration. The standard SQL syntax has been extended to extract the value of any targeted primary key field. In the Extraction of foreign keys example [p 447], the following SQL expressions are valid:

- `tableA.fkb.id = 123`
- `YEAR(tableA.fkb.date) > 2018`

> **Note**
>
> Even if the primary key is composed of only one field, the name of the field must be specified to access the value. For example, if the primary key is composed of a single `id`, `fkb.id` must be used to access the value, as in `tableA.fkb.id = 123`

A typical example of tableRef usage would be SQL joins:

```
SELECT * FROM employee JOIN department ON employee.fkDept.id = department.id
```

Or in case the referenced table has a composite primary key:

```
SELECT * FROM tableA JOIN tableB ON tableA.fkB.id1 = tableB.id1 AND tableA.fkB.id2 =
tableB.id2
```

Analogously, we could perform the same JOIN via the string representation of the key, using `FK_AS_STRING` built-in function [p 1283] and `$pk` system column [p 1260]:

```
SELECT * FROM tableA JOIN tableB ON FK_AS_STRING(tableA.fkB) = tableB."$pk"
```

Metadata fields

EBX® SQL provides also access to some metadata information. These fields are not returned by a SQL statement, unless they are explicitly referenced. For example, "`SELECT * FROM ...`" does not return metadata fields, but "`SELECT metadataField FROM ...`" does.

Name	Description	SQL type	Java class	Examples

`creator`	The **`creator`** `Adaptation.getCreator`[API] of the record.	`VARCHAR`	`java.lang.String`	`SELECT t."ebx-metadata"."system"."creator" FROM myTable t`
				`SELECT t.* FROM myTable t WHERE t."ebx-metadata"."system"."creator" = 'Uuser1'`
				`SELECT t.* FROM myTable t ORDER BY t."ebx-metadata"."system"."creator"`

`creation_time`	The **`creation time`** `Adaptation.getTimeOfCreation`[API] of the record.	`TIMESTAMP`	`java.util.Date`	`SELECT t."ebx-metadata"."system"."creation_time" FROM myTable t`
				`SELECT t.* FROM myTable t WHERE t."ebx-metadata"."system"."creation_time" > TIMESTAMP'2001-06-10 14:57:20'`
				`SELECT t.* FROM myTable t ORDER BY t."ebx-metadata"."system"."creation_time"`

`updater`	The **`last user`** `Adaptation.getLastUser`[API] who modified the record.	`VARCHAR`	`java.lang.String`	`SELECT t."ebx-metadata"."system"."updater" FROM myTable t`
				`SELECT t.* FROM myTable t WHERE t."ebx-metadata"."system"."updater" = 'Uuser1'`
				`SELECT t.* FROM myTable t ORDER BY t."ebx-metadata"."system"."updater"`

`update_time`	The time of the **`last modification`** `Adaptation.getTimeOfLastModification`[API] of the record.	`TIMESTAMP`	`java.util.Date`	`SELECT t."ebx-metadata"."system"."update_time" FROM myTable t`
				`SELECT t.* FROM myTable t WHERE t."ebx-metadata"."system"."update_time" > TIMESTAMP'2001-06-10 14:57:20'`
				`SELECT t.* FROM myTable t ORDER BY t."ebx-metadata"."system"."update_time"`

System columns

Apart from the fields present in a table, EBX® SQL provides some extra system columns. These columns are not returned by a SQL statement, unless they are explicitly referenced. For example, "`SELECT * FROM ...`" does not return systems columns, but "`SELECT systemColumnName FROM ...`" does.

Name	Description	SQL type	Java class	Examples
`$adaptation`	The `Adaptation`[API] representing the table record.	internal type	`com.onwbp.adaptation.Adaptation`	`SELECT t."$adaptation" FROM myTable t WHERE t.value>100`

`$pk`	String representation of the primary key of the record.
	See also *`PrimaryKey.syntax`*[API]
		`VARCHAR`	`java.lang.String`	`SELECT t.* FROM myTable1 t WHERE t."$pk"='123'`
				`SELECT t.* FROM myTable2 t WHERE t."$pk"='123|abc'`
				`SELECT t."$pk" FROM myTable3 t WHERE t.value>100`
				`SELECT t.value FROM myTable3 t ORDER BY t."$pk"`

180.4 SQL syntax

Supported standard operators and functions

An operator is a reserved word or a character used primarily in a SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in a SQL statement, and to serve as conjunctions for multiple conditions in a statement. EBX® supports most of the SQL standard operators and functions. Some functions and operators can have optional parameters: they are surrounded by square brackets in the documentation. Generally, there are five types of operators and functions:

- Comparison operators and functions [p 1264]
- Arithmetic operators and functions [p 1268]
- Logical operators [p 1272]
- String operators and functions [p 1276]
- Date and time functions [p 1280]

The following table lists all of the operators' associativity and precedence, highest to lowest.

Operator	Associativity
.	left
[ ] (array element)	left
+ - (unary plus, minus)	right
* / %	left
+ -	left
BETWEEN, IN, LIKE, CONTAINS, and so on	-
< > = <= >= <> !=	left
IS NULL, IS FALSE, IS NOT TRUE, and so on	-
NOT	right
AND	left
OR	left

Escaping identifiers

In the following cases, the identifier must be escaped by using double quotes:

- when using the absolute path to identify a table (for example, "/root/myTable").
- when the field to identify is a reserved word (for example, "user", "order").
- when referring to a system column with a table alias (for example, t."$adaptation", t."$pk").
- when referring to a metadata field (for example, t."ebx-metadata"."system"."creation_time").

The following example shows a query to illustrate all cases:

```
SELECT t."user", t."$pk" FROM "/root/myTable" t WHERE t."order" = 1
```

Explain plan

EBX® SQL supports `EXPLAIN PLAN FOR ...` syntax to get the plan information of a query.

The result is similar to `Query.explain`[API].

Example: `EXPLAIN PLAN FOR SELECT id FROM myTable`

180.5 Limitations and performance guidelines

- Certain internal join optimizations do not support RIGHT and FULL joins, so avoid these join types if possible.

- The maximum precision and scale for numeric or decimal values is 1000.
- Queries using `GROUP BY` and/or aggregate functions (`MIN`, `MAX`, and so on) are not optimized, except for `COUNT`, which can be optimized in some circumstances.
- Currently, `MIN` and `MAX` operators do not exploit internal indices. Instead, use the following equivalent queries, which are probably more efficient:

  `SELECT val FROM myTable ORDER BY val DESC NULLS LAST LIMIT 1` instead of `SELECT MAX(val) FROM myTable`

  `SELECT val FROM myTable ORDER BY val LIMIT 1` instead of `SELECT MIN(val) FROM myTable`

CHAPTER **181**

Comparison operators

The table below lists all the SQL arithmetic operators and functions supported by EBX®, along with their standard SQL syntax. Some functions may have optional parameters: they are surrounded by square brackets.

Operator syntax	**Description and example(s)**

value1 = value2	Equals
	`SELECT 1 = 0 :  false`
	`SELECT 4 = 4 :  true`

value1 <> value2	Not equals
	`SELECT 1 <> 0 :  true`
	`SELECT 4 <> 4 :  false`

value1 > value2	Greater than
	`SELECT 1 > 0 :  true`
	`SELECT 4 > 4 :  false`

value1 >= value2	Greater than or equal
	`SELECT 1 >= 0 :  true`
	`SELECT 4 >= 4 :  true`

value1 < value2	Lower than
	`SELECT 1 < 0 :  false`
	`SELECT 4 < 4 :  false`

value1 <= value2	Less than or equal
	`SELECT 1 <= 0 :  false`
	`SELECT 4 <= 4 :  true`

value IS NULL	Whether value is null
	`SELECT 1 IS NULL :  false`
	`SELECT NULL IS NULL :  true`

value IS NOT NULL	Whether value is not null

Operator syntax	Description and example(s)
	`SELECT 1 IS NOT NULL :  true` `SELECT NULL IS NOT NULL :  false`

value1 IS DISTINCT FROM value2	Whether two values are not equal, treating null values as the same
	`SELECT 1 IS DISTINCT FROM 1 :  false`
	`SELECT 1 IS DISTINCT FROM 4 :  true`
	`SELECT 1 IS DISTINCT FROM NULL :  true`
	`SELECT NULL IS DISTINCT FROM NULL :  false`

value1 IS NOT DISTINCT FROM value2	Whether two values are equal, treating null values as the same
	`SELECT 1 IS NOT DISTINCT FROM 1 :  true`
	`SELECT 1 IS NOT DISTINCT FROM 4 :  false`
	`SELECT 1 IS NOT DISTINCT FROM NULL :  false`
	`SELECT NULL IS NOT DISTINCT FROM NULL :  true`

value1 BETWEEN value2 AND value3	Whether value1 is greater than or equal to value2 and less than or equal to value3
	`SELECT 4 BETWEEN 3 AND 10 :  true`
	`SELECT 1 BETWEEN 3 AND 10 :  false`

value1 NOT BETWEEN value2 AND value3	Whether value1 is greater than or equal to value2 and less than or equal to value3
	`SELECT 4 NOT BETWEEN 3 AND 10 :  false`
	`SELECT 1 NOT BETWEEN 3 AND 10 :  true`

string1 LIKE string2	Whether string1 matches pattern string2. The wildcard '%' represents zero, one or multiple characters. The wildcard '_' represents one single character. Use 'ESCAPE' to define a escape character (if you want to escape a wildcard). To find if string1 starts with a sequence, use pattern 'sequence %' (analogously use '%sequence' for ends with). The matching is case sensitive. To perform a case insensitive matching, use UPPER (or LOWER) on string1 and pattern.
	`SELECT name FROM employee WHERE name LIKE 'S%' : Smith, Saunders`
	`SELECT name FROM employee WHERE UPPER(name) LIKE 'SM %' :  Smith`
	`SELECT name FROM employee WHERE name LIKE '_m%' : Smith`
	`SELECT 'abcd%' LIKE '%!%' ESCAPE '!' :  true`

string1 NOT LIKE string2	Whether string1 does not match pattern string2. The wildcard '%' represents zero, one or multiple characters. The wildcard '_' represents one single character. Use 'ESCAPE' to define a escape character (if you want to escape a wildcard). To find if string1 does not start with a sequence, use pattern 'sequence %' (analogously use '%sequence' for ends with). The matching is case sensitive. To perform a case insensitive matching, use UPPER (or LOWER) on string1 and pattern. `SELECT name FROM employee WHERE name NOT LIKE 'S%' : Hamilton`

Operator syntax	Description and example(s)

	`SELECT name FROM employee WHERE UPPER(name) NOT LIKE 'SM%' :  Hamilton, Saunders`
	`SELECT name FROM employee WHERE name NOT LIKE '_m %' :  Hamilton, Saunders`
	`SELECT 'abcd%' NOT LIKE '%!_' ESCAPE '!' :  true`

value IN (value [, value]*)	Whether value is equal to a value in a list `SELECT firstname FROM employee WHERE name IN ('Smith', 'Hamilton') :  John, Maria`
value NOT IN (value [, value]*)	Whether value is not equal to every value in a list `SELECT firstname FROM employee WHERE name NOT IN ('Smith', 'Hamilton') :  Jennifer`
value IN (sub-query)	Whether value is equal to a row returned by sub-query `SELECT e.firstname FROM employee e WHERE e.department.id IN (SELECT d.id FROM department d WHERE d.name='IT') :  John`

CHAPTER **182**

Arithmetic operators and functions

The following table lists all of the SQL arithmetic operators and functions supported by EBX®, along with their standard SQL syntax. Some functions can have optional parameters: they are surrounded by square brackets.

Operator syntax	**Description and example(s)**
+ numeric	Returns numeric `SELECT +4 : 4`
- numeric	Returns numeric `SELECT -4 : -4`
numeric1 + numeric2	Returns numeric1 plus numeric2 `SELECT 18 + 4 : 22`
numeric1 - numeric2	Returns numeric1 minus numeric2 `SELECT 18 - 4 : 14`
numeric1 * numeric2	Returns numeric1 multiplied by numeric2 `SELECT 18 * 4 : 72`
numeric1 / numeric2	Returns numeric1 divided by numeric2 `SELECT 18 / 4 : 4`
ABS(numeric)	Returns the absolute value of numeric `SELECT ABS(-243.5) : 243.5`
ACOS(numeric)	Returns the arc cosine of numeric `SELECT ACOS(-1) : 3.141592653589793`
ASIN(numeric)	Returns the arc sine of numeric `SELECT ABS(0.25) : 0.25`
ATAN(numeric)	Returns the arc tangent of numeric `SELECT ATAN(2.5) : 1.1902899496825317`

Operator syntax	Description and example(s)
ATAN2(numeric, numeric)	Returns the arc tangent of the numeric coordinates `SELECT ATAN2(0.50, 1) :  0.4636476090008061`
CEIL(numeric)	Rounds numeric up, and returns the smallest number that is greater than or equal to numeric `SELECT CEIL(25.75) :  26`
CEILING(numeric)	Rounds numeric up, and returns the smallest number that is greater than or equal to numeric `SELECT CEILING(25.75) :  26`
COS(numeric)	Returns the cosine of numeric `SELECT COS(2) :  -0.4161468365471424`
COT(numeric)	Returns the cotangent of numeric `SELECT COT(6) :  -3.436353004180128`
DEGREES(numeric)	Converts numeric from radians to degrees `SELECT DEGREES(1.5) :  85.94366926962348`
EXP(numeric)	Returns e raised to the power of numeric `SELECT EXP(1) :  2.718281828459045`
FLOOR(numeric)	Rounds numeric down, and returns the largest number that is less than or equal to numeric `SELECT FLOOR(25.75) :  25`
LN(numeric)	Returns the natural logarithm (base e) of numeric `SELECT LN(2) :  0.6931471805599453`
LOG10(numeric)	Returns the base-10 logarithm of numeric `SELECT LOG10(2) :  0.3010299956639812`
MOD(numeric1, numeric2)	Returns the remainder (modulus) of numeric1 divided by numeric2. The result is negative only if numeric1 is negative `SELECT MOD(18, 4) :  2`
POWER(numeric1, numeric)	Returns numeric1 raised to the power of numeric2 `SELECT POWER(4, 2) :  16.0`
RADIANS(numeric)	Converts numeric from degrees to radians `SELECT RADIANS(180) :  3.141592653589793`
RAND([seed])	Returns a random double using numeric as the seed value if specified `SELECT RAND(6) :  0.9891840064573959`

Operator syntax	Description and example(s)

RAND_INTEGER([seed,] numeric)	Generates a random integer between 0 and numeric - 1 inclusive, initializing the random number generator with seed if specified
	`SELECT RAND_INTEGER(100, 5) :  0`
	`SELECT RAND_INTEGER(6, 100) :  11`

ROUND(numeric1, numeric2)	Rounds numeric1 to numeric2 places right to the decimal point `SELECT ROUND(135.375, 2) :  135.38`
SIGN(numeric)	Returns the signum of numeric `SELECT SIGN(255.5) :  1`
SIN(numeric)	Returns the sine of numeric `SELECT SIN(2) :  0.9092974268256817`
SQRT(numeric)	Returns the square root of numeric `SELECT SQRT(64) :  8.0`
TAN(numeric)	Returns the tangent of numeric `SELECT TAN(1.75) :  -5.52037992250933`

TRUNCATE(numeric1[, numeric2])	Truncates numeric1 to 0 (if no numeric2 specified) places right to the decimal point
	`SELECT TRUNCATE(135.375) :  135`
	`SELECT TRUNCATE(135.375, 2) :  135.37`

CHAPTER **183**

Logical operators

The following table lists all of the SQL logical operators supported by EBX®, along with their standard SQL syntax.

Operator syntax	Description and example(s)

boolean1 OR boolean2	Whether boolean1 is TRUE or boolean2 is TRUE
	`SELECT TRUE OR TRUE : true`
	`SELECT TRUE OR FALSE : true`
	`SELECT FALSE OR TRUE : true`
	`SELECT FALSE OR FALSE : false`

boolean1 AND boolean2	Whether boolean1 is TRUE and boolean2 is TRUE
	`SELECT TRUE AND TRUE : true`
	`SELECT TRUE AND FALSE : false`
	`SELECT FALSE AND TRUE : false`
	`SELECT FALSE AND FALSE : false`

NOT boolean	Whether boolean is not TRUE; returns UNKNOWN if boolean is UNKNOWN
	`SELECT NOT TRUE : false`
	`SELECT NOT FALSE : true`
	`SELECT NOT UNKNOWN : null`

boolean IS FALSE	Whether boolean is FALSE; returns FALSE if boolean is UNKNOWN
	`SELECT TRUE IS FALSE : false`
	`SELECT FALSE IS FALSE : true`

boolean IS NOT FALSE	Whether boolean is not FALSE; returns TRUE if boolean is UNKNOWN
	`SELECT TRUE IS NOT FALSE : true`
	`SELECT FALSE IS NOT FALSE : false`

boolean IS TRUE	Whether boolean is TRUE; returns TRUE if boolean is UNKNOWN `SELECT TRUE IS TRUE : true`

Operator syntax	Description and example(s)
	`SELECT FALSE IS TRUE :  false`

boolean IS NOT TRUE	Whether boolean is not TRUE; returns FALSE if boolean is UNKNOWN
	`SELECT TRUE IS NOT TRUE :  false`
	`SELECT FALSE IS NOT TRUE :  true`

boolean IS UNKNOWN	Whether boolean is UNKNOWN
	`SELECT TRUE IS UNKNOWN :  false`
	`SELECT FALSE IS UNKNOWN :  false`

boolean IS NOT UNKNOWN	Whether boolean is not UNKNOWN
	`SELECT TRUE IS NOT UNKNOWN :  true`
	`SELECT FALSE IS NOT UNKNOWN :  true`

CHAPTER **184**

String operators and functions

The following table lists all of the SQL string operators and functions supported by EBX®, along with their standard SQL syntax. Some functions can have optional parameters: they are surrounded by square brackets.

Operator syntax	**Description and example(s)**
string || string	Concatenates two character strings `SELECT 'Hello ' || 'world !' :  Hello world !`
CHAR_LENGTH(string)	Returns the number of characters in a character string `SELECT CHAR_LENGTH('Alfreds Futterkiste') :  19`
CHARACTER_LENGTH(string)	As CHAR_LENGTH(string) `SELECT CHARACTER_LENGTH('Alfreds Futterkiste') :  19`
UPPER(string)	Returns a character string converted to upper case `SELECT UPPER('SQL Tutorial is FUN!') :  SQL TUTORIAL IS FUN!`

POSITION(string1 IN string2 [FROM integer])	Returns the position of the first occurrence of string1 in string2 starting at a given point if specified
	`SELECT POSITION('is fun' IN 'Tutorial is FUN!') :  0`
	`SELECT POSITION('fun' IN 'Tutorial is FUN, very FUN!' FROM 17) :  0`

TRIM({ BOTH | LEADING | TRAILING } character FROM string)	Removes the longest string containing only the character from the start/end/both ends of string
	`SELECT TRIM(' ' FROM ' #SQL Tutorial! ') :  #SQL Tutorial!`
	`SELECT TRIM(LEADING ' ' FROM ' #SQL Tutorial! ') : #SQL Tutorial!`
	`SELECT TRIM(TRAILING ' ' FROM ' #SQL Tutorial! ') : #SQL Tutorial!`
	`SELECT TRIM(BOTH ' ' FROM ' #SQL Tutorial! ') : #SQL Tutorial!`

OVERLAY(string1 PLACING string2 FROM integer [FOR integer2])	Replaces a substring of string1 with string2

Operator syntax	Description and example(s)
	`SELECT OVERLAY('Tutorial is very FUN!' PLACING 'VERY' FROM 13) :  Tutorial is VERY FUN!` `SELECT OVERLAY('Tutorial is very FUN!' PLACING 'VERY' FROM 13 FOR 4) :  Tutorial is VERY FUN!`

SUBSTRING(string FROM integer [FOR integer])	Returns a substring of a character string starting at a given point
	`SELECT SUBSTRING('Tutorial is very FUN!' FROM 13) : very FUN!`
	`SELECT SUBSTRING('Tutorial is very FUN!' FROM 13 FOR 4) :  very`

CHAPTER **185**

Date and time functions

The following table lists all of the SQL date and time functions supported by EBX®, along with their standard SQL syntax. Some functions can have optional parameters: they are surrounded by square brackets.

Operator syntax	**Description and example(s)**
CURRENT_TIME	Returns the current time in the session time zone, in a value of datatype TIMESTAMP WITH TIME ZONE
CURRENT_DATE	Returns the current date in the session time zone, in a value of datatype DATE
CURRENT_TIMESTAMP	Returns the current date and time in the session time zone, in a value of datatype TIMESTAMP WITH TIME ZONE

Cast strings to timestamps	Note that datetime values displayed in EBX (UI and export files) are in local time zone, whereas the timestamp literals created by the SQL parser are in UTC; therefore if the server is not in UTC, you might need to create the literals on the local time zone to avoid this mismatch and get the expected results (see third example). You should also consider if the datetime value has millisecond precision (which is not displayed by EBX UI) in order to write your filter (see last example).
	`SELECT COUNT(*) FROM employee WHERE lastProfileUpdateTime=TIMESTAMP'2017-06-28 18:06:12' : 0`
	`SELECT COUNT(*) FROM employee WHERE lastProfileUpdateTime=CAST('2017-06-28 18:06:12' AS TIMESTAMP) : 0`
	`SELECT COUNT(*) FROM employee WHERE lastProfileUpdateTime=CAST('2017-06-28 18:06:12 Europe/Paris' AS TIMESTAMP WITH LOCAL TIME ZONE) : 1`
	`SELECT COUNT(*) FROM employee e WHERE e."ebx-metadata"."system"."update_time">=TIMESTAMP'2023-08-28 09:56:55' AND e."ebx-metadata"."system"."update_time"<TIMESTAMP'2023-08-28 09:56:56' : 1`

YEAR(date)	Extracts and returns the value of the year from a datetime value expression. Returns an integer. `SELECT YEAR(TIMESTAMP '1971-07-20 09:34:21') : 1971`

Operator syntax	Description and example(s)

	`SELECT YEAR(DATE '1968-07-20') : 1968`
	`SELECT YEAR(hiringDate) FROM employee WHERE name='Smith' : 2015`
	`SELECT YEAR(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 2017`

QUARTER(date)	Extracts and returns the value of the quarter from a datetime value expression. Returns an integer between 1 and 4.
	`SELECT QUARTER(TIMESTAMP '1971-07-20 09:34:21') : 3`
	`SELECT QUARTER(hiringDate) FROM employee WHERE name='Smith' : 4`
	`SELECT QUARTER(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 2`

QUARTER(date)	Extracts and returns the value of the month from a datetime value expression. Returns an integer between 1 and 12.
	`SELECT MONTH(TIMESTAMP '1971-07-20 09:34:21') : 7`
	`SELECT MONTH(hiringDate) FROM employee WHERE name='Smith' : 10`
	`SELECT MONTH(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 6`

WEEK(date)	Extracts and returns the value of the week from a datetime value expression. Returns an integer between 1 and 53.
	`SELECT WEEK(TIMESTAMP '1971-07-20 09:34:21') : 29`
	`SELECT WEEK(hiringDate) FROM employee WHERE name='Smith' : 42`
	`SELECT WEEK(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 26`

DAYOFYEAR(date)	Extracts and returns the value of the day of year from a datetime value expression. Returns an integer between 1 and 366.
	`SELECT DAYOFYEAR(TIMESTAMP '1971-07-20 09:34:21') : 201`
	`SELECT DAYOFYEAR(hiringDate) FROM employee WHERE name='Smith' : 287`
	`SELECT DAYOFYEAR(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 179`

DAYOFMONTH(date)	Extracts and returns the value of the day of month from a datetime value expression. Returns an integer between 1 and 31.
	`SELECT DAYOFMONTH(TIMESTAMP '1971-07-20 09:34:21') : 20`
	`SELECT DAYOFMONTH(hiringDate) FROM employee WHERE name='Smith' : 14`
	`SELECT DAYOFMONTH(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 28`

DAYOFWEEK(date)	Extracts and returns the value of the day of week from a datetime value expression. Returns an integer between 1 and 7.

Operator syntax	Description and example(s)

	`SELECT DAYOFWEEK(TIMESTAMP '1971-07-20 09:34:21') : 3`
	`SELECT DAYOFWEEK(hiringDate) FROM employee WHERE name='Smith' : 4`
	`SELECT DAYOFWEEK(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 4`

HOUR(date)	Extracts and returns the value of the hours from a datetime value expression. Returns an integer between 0 and 23.
	`SELECT HOUR(TIMESTAMP '1971-07-20 09:34:21') : 9`
	`SELECT HOUR(TIME '10:34:21') : 10`
	`SELECT HOUR(workStart) FROM employee WHERE name='Smith' : 8`
	`SELECT HOUR(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 16`

MINUTE(date)	Extracts and returns the value of the minutes from a datetime value expression. Returns an integer between 0 and 59.
	`SELECT MINUTE(TIMESTAMP '1971-07-20 09:34:21') : 34`
	`SELECT MINUTE(TIME '10:35:21') : 35`
	`SELECT MINUTE(workStart) FROM employee WHERE name='Smith' : 0`
	`SELECT MINUTE(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 6`

SECOND(date)	Extracts and returns the value of the seconds from a datetime value expression. Returns an integer between 0 and 59.
	`SELECT HOUR(TIMESTAMP '1969-07-20 09:34:21') : 9`
	`SELECT SECOND(TIME '10:34:22') : 22`
	`SELECT SECOND(workStart) FROM employee WHERE name='Smith' : 0`
	`SELECT SECOND(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 12`

CHAPTER **186**

EBX® SQL functions

The following table lists all of the EBX® built-in SQL functions, along with their syntax. Some functions can have optional parameters: they are surrounded by square brackets.

Operator syntax	Description and example(s)

FK_AS_STRING(tableRefValue)	Returns the string representation of a tableRef value. This function cannot be used on aggregated lists. This 1-parameter version of FK_AS_STRING can only be used on unambiguous contexts; in other scenarios, e.g. in certain queries after a UNION or UNNEST, the 3-parameter version must be used (see below).
	`SELECT e.name FROM employee e WHERE FK_AS_STRING(e.department) = '1' :  Smith`
	`SELECT FK_AS_STRING(e.department) FROM employee e WHERE e.name = 'Smith' :  1`

FK_AS_STRING('referencedDatasetName', 'referencedTableName', tableRefValue)	Returns the string representation of a tableRef value. This function cannot be used on aggregated lists. See QueryBuilder JavaDoc to learn more about the referenced dataset names.
	`SELECT e.name FROM employee e WHERE FK_AS_STRING('_public', '/root/department', e.department) = '1' :  Smith`
	`SELECT FK_AS_STRING('_public', '/root/department', e.department) FROM employee e WHERE e.name = 'Smith' :  1`

CHAPTER 187

REST Toolkit

This chapter contains the following topics:

1. Introduction
2. Application definitions
3. Service and operation definitions
4. Serialization of a table record
5. Authentication and lookup mechanism
6. REST authentication and permissions
7. URI builders
8. Exception handling
9. Monitoring
10. Packaging and registration
11. OpenAPI
12. Limitations

187.1 Introduction

TIBCO EBX® offers the possibility to develop custom REST services using the REST Toolkit. The REST Toolkit supports Jakarta RESTful Web Services 3.1 and JSR-367: JSON-B.

A REST service is implemented by a Java class and its operations are implemented by Java methods. The response can be generated by serializing POJO objects. The request input can be unserialized to POJOs. Various input and output formats, including JSON, are supported. For more details on supported formats, see media types [p 1287].

Rest Toolkit supports the following:

- Injectable objects

  EBX® provides injectable objects useful to authenticate the request's user, to access the EBX® repository or to built URIs without worrying about the configuration (for example reverse-proxy [p 1115] or REST forward [p 1042] modes);

  JAX-RS injectable objects are also supported.

- Annotations

EBX® provides annotations to describe resources, grant anonymous access or add restrictions to a method.

JAX-RS ans JSON-B annotations are also supported.

- logging and debugging utilities.

See also *Jakarta RESTful Web Services 3.1 API Specification*

187.2 Application definitions

An EBX® module, that includes custom REST services, must provide at least one REST Toolkit application class. A REST Toolkit application class extends the EBX® `RESTApplicationAbstract`[API] class. The minimum requirement is to define the base URL, using the `@ApplicationPath` annotation and the set of packages to scan for REST service classes.

> **Note**
>
> Only packages accessible from the web application's classloader can be scanned.

> **Note**
>
> It is possible to register REST resource classes or singletons, packaged inside or outside the web application archive, through the **ApplicationConfigurator.register(java.lang.Class)** `ApplicationConfigurator.register`[API] or **ApplicationConfigurator.register(java.lang.Object)** `ApplicationConfigurator.register`[API] methods.

> **Note**
>
> If no packages scope is defined, then every class reachable from the web application's classloader will be scanned.

The application path cannot be `"/"` and must not collide with an existing resource from the module. It is recommended to use `"/rest"` (the value of the `RESTApplicationAbstract.REST_DEFAULT_APPLICATION_PATH` constant).

EBX® `Documentation`[API] annotation is optional. It is displayed to administrators in 'Technical configuration' > 'Modules and data models' or when logging and debugging.

```
import jakarta.ws.rs.*;

import com.orchestranetworks.rest.*;
import com.orchestranetworks.rest.annotation.*;

@ApplicationPath(RESTApplicationAbstract.REST_DEFAULT_APPLICATION_PATH)
@Documentation("My REST sample application")
public final class RESTApplication extends RESTApplicationAbstract
{
   public RESTApplication()
   {
      // Adds one or more package names which will be used to scan for components.
      super((cfg) -> cfg.addPackages(RESTApplication.class.getPackage()));
   }
}
```

187.3 Service and operation definitions

A REST Toolkit service is implemented by a Java class and its operations are implemented by its methods.

Class and methods can be annotated by `@Path` to specify the access path. The `@Path` annotation value defined at the class level will prepend the ones defined on methods. Warning, only one `@Path` annotation is allowed per class or method.

Media types accepted and produced by a resource are respectively defined by the `@Consumes` and `@Produces` JAX-RS annotations. The supported media types are:

- `application/json` (MediaType.APPLICATION_JSON_TYPE)
- `application/octet-stream` (MediaType.APPLICATION_OCTET_STREAM_TYPE)
- `application/x-www-form-urlencoded` (MediaType.APPLICATION_FORM_URLENCODED_TYPE)
- `multipart/form-data` (MediaType.MULTIPART_FORM_DATA_TYPE)
- `text/css`
- `text/html` (MediaType.TEXT_HTML_TYPE)
- `text/plain` (MediaType.TEXT_PLAIN_TYPE)

Valid HTTP(S) methods are specified by JAX-RS annotations `@GET`, `@POST`, `@PUT`, etc. Only one of these annotations can be set on each Java method (this means that a Java method can support only one HTTP method).

Warning: URL parameters with a name prefixed with `ebx-` are reserved by REST Toolkit and should not be defined by custom REST services, unless explicitly authorized by the EBX® documentation.

URL and sample

The REST URL to access the description service for the sample is defined below:

`http[s]://<host>[:<port>]/<path to webapp>/rest/track/v1/description`

Where:

- `<path to webapp>` corresponds to the web application's path holding the REST Toolkit application, itself serving the sample service. The path is composed by multiple, or none, URI segments followed by the web application's name.

> **Note**
>
> Please note that `/rest/track/v1/description` corresponds to the concatenation of the application's `@ApplicationPath` and service's `@Path` annotations.

The following REST Toolkit service sample provides methods to query and manage track data:

```
import java.net.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.regex.*;
import java.util.stream.*;

import jakarta.servlet.http.*;
import jakarta.ws.rs.*;
import jakarta.ws.rs.container.*;
import jakarta.ws.rs.core.*;

import com.orchestranetworks.rest.annotation.*;
import com.orchestranetworks.rest.inject.*;

/**
 * The REST Toolkit Track service v1.
 */
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
@Path("/track/v1")
@Documentation("Track service")
```

```
public final class TrackService
{
 @Context
 private ResourceInfo resourceInfo;

 @Context
 private SessionContext sessionContext;

 private static final Map<Integer, TrackDTO> TRACKS = new ConcurrentHashMap<>();

 /**
  * Gets service description
  */
 @GET
 @Path("/description")
 @Documentation("Gets service description")
 @Produces({ MediaType.TEXT_PLAIN, MediaType.APPLICATION_JSON })
 @AnonymousAccessEnabled
 public String handleServiceDescription()
 {
  return this.resourceInfo.getResourceMethod().getAnnotation(Documentation.class).value();
 }

 /**
  * Selects tracks.
  */
 @GET
 @Path("/tracks")
 @Documentation("Selects tracks")
 public Collection<TrackDTO> handleSelectTracks(
   @QueryParam("singerFilter") final String singerFilter, // a URL parameter holding a Java regular expression
   @QueryParam("titleFilter") final String titleFilter) // a URL parameter holding a Java regular expression
 {
  final Pattern singerPattern = TrackService.compilePattern(singerFilter);
  final Pattern titlePattern = TrackService.compilePattern(titleFilter);

  return TRACKS.values()
    .stream()
    .filter(Objects::nonNull)
    .filter(track -> singerPattern == null || singerPattern.matcher(track.singer).matches())
    .filter(track -> titlePattern == null || titlePattern.matcher(track.title).matches())
    .collect(Collectors.toList());
 }

 private static Pattern compilePattern(final String aPattern)
 {
  if (aPattern == null || aPattern.isEmpty())
   return null;

  try
  {
   return Pattern.compile(aPattern);
  }
  catch (final PatternSyntaxException ignore)
  {
   // ignore invalid pattern
   return null;
  }
 }

 /**
  * Counts all tracks.
  */
 @GET
 @Path("/tracks:count")
 @Documentation("Counts all tracks")
 public int handleCountTracks()
 {
  return TRACKS.size();
 }

 /**
  * Selects a track by id.
  */
 @GET
 @Path("/tracks/{id}")
 @Documentation("Selects a track by id")
 public TrackDTO handleSelectTrackById(@PathParam("id") Integer id)
 {
  final TrackDTO track = TRACKS.get(id);
  if (track == null)
   throw new NotFoundException("Track id [" + id + "] does not found.");
  return track;
 }
```

```
/**
 * Deletes a track by id.
 */
@DELETE
@Path("/tracks/{id}")
@Documentation("Deletes a track by id")
public void handleDeleteTrackById(@PathParam("id") Integer id)
{
 if (!TRACKS.containsKey(id))
  throw new NotFoundException("Track id [" + id + "] does not found.");
 TRACKS.remove(id);
}

/**
 * Inserts or updates one or several tracks.
 * <p>
 * The complex response structure corresponds to one of:
 * <ul>
 *  <li>An empty content with the <code>location<code> HTTP header defined
 *   to the access URI.</li>
 *  <li>A JSON array of {@link ResultDetailsDTO} objects.</li>
 * </ul>
 */
@POST
@Path("/tracks")
@Documentation("Inserts or updates one or several tracks")
public Response handleInsertOrUpdateTracks(List<TrackDTO> tracks)
{
 int inserted = 0;
 int updated = 0;

 final ResultDetailsDTO[] resultDetails = new ResultDetailsDTO[tracks.size()];
 int resultIndex = 0;

 final URI base = this.sessionContext.getURIInfoUtility()
   .createBuilderForRESTApplication()
   .path(this.getClass())
   .segment("tracks")
   .build();

 for (final TrackDTO track : tracks)
 {
  final String id = String.valueOf(track.id);
  final URI uri = UriBuilder.fromUri(base).segment(id).build();

  final int code;
  if (TRACKS.containsKey(track.id))
  {
   code = HttpServletResponse.SC_NO_CONTENT;
   updated++;
  }
  else
  {
   code = HttpServletResponse.SC_CREATED;
   inserted++;
  }

  TRACKS.put(track.id, track);

  resultDetails[resultIndex++] = ResultDetailsDTO.create(
   code,
   null,
   String.valueOf(track.id),
   uri);
 }

 if (inserted == 1 && updated == 0)
  return Response.created(resultDetails[0].details).build();

 return Response.ok().entity(resultDetails).build();
}

/**
 * Updates one track.
 */
@PUT
@Path("/tracks/{id}")
@Documentation("Update one track")
public void handleUpdateOneTrack(@PathParam("id") Integer id, TrackDTO aTrack)
{
 final TrackDTO track = TRACKS.get(id);
 if (track == null)
  throw new NotFoundException("Track id [" + id + "] does not found.");

 if (aTrack.id != null && !aTrack.id.equals(track.id))
```

```
   throw new BadRequestException("Selected track id [" + id
    + "] is not equals to body track id.");

  TRACKS.put(aTrack.id, aTrack);
 }
}
```

This REST service uses the following Java classes, which represent a Data Transfer Objects (DTO), to serialize and deserialize data:

```
/**
 * DTO for a track.
 */
public final class TrackDTO
{
 public Integer id;
 public String singer;
 public String title;
}

import java.net.*;

/**
 * DTO for result details.
 */
@JsonbPropertyOrder({ "code", "label", "foreignKey", "details" })
public final class ResultDetailsDTO
{
 public int code;
 public String label;
 public String foreignKey;
 public URI details;

 public static ResultDetailsDTO create(
  final int aCode,
  final String aForeignKey,
  final URI aDetails)
 {
  return ResultDetailsDTO.create(aCode, null, aForeignKey, aDetails);
 }

 public static ResultDetailsDTO create(
  final int aCode,
  final String aLabel,
  final String aForeignKey,
  final URI aDetails)
 {
  final ResultDetailsDTO result = new ResultDetailsDTO();
  result.code = aCode;
  result.label = aLabel;
  result.foreignKey = aForeignKey;
  result.details = aDetails;
  return result;
 }
}
```

187.4 Serialization of a table record

Built-in serializers

Default JSON serializers and deserializers are provided to handle table records when declared in DTOs as `ContentHolder`[API]s. Both extended [p 1194] and compact [p 1224] JSON formats of record are supported.

```
/**
 * DTO for a singer.
 */
public final class SingerDTO
{
 @Table(
  dataModel = "urn:ebx:module:tracks-module:/WEB-INF/ebx/schemas/tracks.xsd",
  tablePath = "/root/Singers")
 public ContentHolder content;
}
```

A same DTO can be used for serialization and deserialization. In case of serialization, a `ContentHolderForInput`[API] instance will be automatically created and filled with the proper data. Afterwards, this instance will be able to copy its data into a `ValueContextForUpdate`[API]. To deserialize a table records, a `ContentHolderForOutput`[API] must be created in the REST operation JAVA method and returned. The provided `Adaptation`[API] data will then be transformed into a valid peace of JSON and placed into the HTTP response body.

```
/**
 * The REST Toolkit Singer service v1.
 */
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
@Path("/singer/v1")
@Documentation("Singer service")
public final class SingerService
{
 ...

 /**
  * Selects a singer by id.
  */
 @GET
 @Path("/singers/{id}")
 @Documentation("Selects a singer by id")
 public SingerDTO handleSelectSingerById(final @PathParam("id") Integer id)
 {
  // find the singer adaptation by id
  final Adaptation singerRecord = ... ;

  final SingerDTO singerDTO = new SingerDTO();
  singerDTO.content = ContentHolderForOutput.createForRecord(singerRecord);
  return singerDTO;
 }


 /**
  * Inserts one singer.
  */
 @POST
 @Path("/singers")
 @Documentation("Inserts one singer")
 public void handleInsertOneSinger(final SingerDTO aSingerDTO)
 {
  final ProgrammaticService svc = ... ;
  final AdaptationTable singersTable = ... ;
  final ProcedureResult procedureResult = svc.execute(
    (aContext) -> {
     final ValueContextForUpdate createContext = aContext.getContextForNewOccurrence(singersTable); ;
     aSingerDTO.content.asContentHolderForInput().copyTo(createContext);
     aContext.doCreateOccurrence(createContext, singersTable);
    });

  if (procedureResult.hasFailed())
   throw new UnprocessableEntityException(
    procedureResult.getException().getLocalizedMessage());
 }

 /**
  * updates one singer.
  */
 @PUT
 @Path("/singers/{id}")
 @Documentation("updates one singer")
 public void handleUpdateOneSinger(@PathParam("id") final Integer id, final SingerDTO aSingerDTO)
 {
  final ProgrammaticService svc = ... ;
  final AdaptationTable singersTable = ... ;
  final ProcedureResult procedureResult = svc.execute(
    (aContext) -> {
     // find the singer adaptation by id
     final Adaptation singerRecord = ... ;

     if (singerRecord == null)
      throw new NotFoundException("Singer with the id ["+ id + "] has not been found.");

     final ValueContextForUpdate createContext = aContext.getContext(singerRecord.getAdaptationName()); ;
     aSingerDTO.content.asContentHolderForInput().copyTo(createContext);
     aContext.doModifyContent(singerRecord, createContext);
    });

  if (procedureResult.hasFailed()){
```

```
   final Exception ex = procedureResult.getException();
   final Throwable cause = ex.getCause();
   if(cause instanceof NotFoundException)
    throw (NotFoundException) cause;

   throw new UnprocessableEntityException(ex.getLocalizedMessage());
  }
 }
}
```

The default JSON format of the responses, only composed of business data fields, is called *compact*.

```
{
  "singer":{
   "firstname":"Frank",
   "lastname":"Sinatra"
  }
}
```

To add technical fields or metadata, the `ExtendedOutput`[API] annotation must be placed over the `ContentHolder`[API] field. The annotation must declare every wished options or the *ALL* one.

```
/**
 * DTO for a singer with technical fields.
 */
public final class SingerWithTechnicalsDTO
{
 @Table(
  dataModel = "urn:ebx:module:tracks-module:/WEB-INF/ebx/schemas/tracks.xsd",
  tablePath = "/root/Singers")
 @ExtendedOutput({Include.LABEL,Include.TECHNICALS,Include.CONTENT})
 public ContentHolder content;
}

{
  "singer":{
    "label": "23",
    "creationDate": "2018-09-04T15:35:10.706",
    "creationUser": "user",
    "lastUpdateDate": "2018-10-02T17:05:47.090",
    "lastUpdateUser": "user",
   "content":{
    "firstname":{
     "content":"Frank",
     "label":"First Name"
    },
    "lastname":{
     "content":"Sinatra",
     "label":"Last Name"
    }
    }
  }
}
```

To use the extended JSON format for input HTTP requests the `ExtendedInput`[API] annotation must be placed over the `ContentHolder`[API] field.

```
/**
 * DTO for a singer using the extended JSON format.
 */
public final class SingerExtendedInputDTO
{
 @Table(
  dataModel = "urn:ebx:module:tracks-module:/WEB-INF/ebx/schemas/tracks.xsd",
  tablePath = "/root/Singers")
 @ExtendedInput
 public ContentHolder content;
}
```

Custom serializers

Since TIBCO EBX® is based on JSON-B (JSR-367), custom serializers and deserializers can be defined through JsonbTypeSerializer and JsonbTypeDeserializer annotations.

By default, `java.math.BigDecimal` is serialized into a JSON `String`. To override this behaviour, implement a custom serializer to a JSON `Number`.

```
/**
 * Serializes decimal values in numbers.
 */
public final class BigDecimalSerializerInJsonNumber implements JsonbSerializer<BigDecimal>
{
 @Override
 public void serialize(
  final BigDecimal aBigDecimal,
  final JsonGenerator aJsonGenerator,
  final SerializationContext aSerializationContext)
 {
  if (aBigDecimal == null)
   aJsonGenerator.writeNull();
  else
   aJsonGenerator.write(aBigDecimal);
 }
}

/**
 * DTO for a vynil.
 */
public final class VinylDTO
{
 @JsonbTypeSerializer(BigDecimalSerializerInJsonNumber.class)
 public BigDecimal price;

 @JsonbTypeSerializer(CustomTrackDtoSerializer.class)
 @JsonbTypeDeserializer(CustomTrackDtoDeserializer.class)
 public TrackDTO track;
}
```

187.5 Authentication and lookup mechanism

A custom REST service developed with REST Toolkit supports the same authentication methods and lookup mechanism as the built-in REST data services. However, there is a slight difference concerning the 'Anonymous authentication Scheme' since its scope can be wider by using the `AnonymousAccessEnabled`[API]. See REST authentication and permissions [p 1293] for more information.

See also

Authentication [p 1119]

Lookup mechanism [p 1120]

187.6 REST authentication and permissions

By default, every REST resource Java method requires users to be authenticated.

However, some methods may need to be accessible anonymously. These methods must be annotated by `AnonymousAccessEnabled`[API].

Some methods may need to be restricted to given profiles. These methods may be annotated by `Authorization`[API] to specify an authorization rule. An authorization rule is a Java class that implements the `AuthorizationRule`[API] interface.

```
import jakarta.ws.rs.*;

import com.orchestranetworks.rest.annotation.*;

/**
 * The REST Toolkit service v1.
 */
@Path("/service/v1")
@Documentation("Service")
public final class Service
{
 ...
```

```
 /**
  * Gets service description
  */
 @GET
 @AnonymousAccessEnabled
 public String handleServiceDescription()
 {
  ...
 }

 /**
  * Gets restricted service
  */
 @GET
 @Authorization(IsUserAuthorized.class)
 public RestrictedServiceDTO handleRestrictedService()
 {
  ...
 }
}
```

187.7 URI builders

REST Toolkit provides an utility interface `URIInfoUtility`[API] to generate URIs. An instance of this interface is accessible through the injectable built-in object `SessionContext`[API].

URI builders for built-in

Several URI builders interfaces have been designed to ease built-in RESTful services URI build. Each interface constitute an aggregation of methods related to the same functional concept. This division allows development of modular and generic algorithms. Some of these interfaces are themselves aggregation of other ones, leading to intuitive use of the builders since only consistent combination of method calls are allowed. For example, a URI builder configured for the REST `hierarchy` category will not allow calls to record URI build methods, since record access are part of the REST `data` category concept.

Moreover, these URI builders are preconfigured according to:

- the data from the incoming HTTP request which constitute the base configuration,
- the Module public path prefix [p 599] which prepends the EBX® module segment, if it exists,
- the data services lineage setting [p 590] which overrides the whole previous configuration, if it exists,
- The REST forward [p 1042] mode which overrides partially the previous configuration since only a peace of the URI's path is kept, if it is activated.

See `URIBuilderForBuiltin`[API] and `CategoryURIBuilder`[API] for more information.

URI builders for resources

URI builders to generate resource access URI are also available. Since resources are seen as only one functional concept, every methods have been defined in the same interface.

Like the URI builder for built-in data services, these ones are already preconfigured according to:

- the data from the incoming HTTP request which constitute the base configuration,
- the external resources settings [p 590] which overrides all or part of the previous configuration, if they exist,
- the Module public path prefix [p 599] which prepends the EBX® module segment, if it exists.

URI builders for server and application

When URIs to services or resources not covered by the previous builders must be build, the default preconfigured URI builders should be used. There are two default URI builders which only differ in their ending path segment. The first one end its path at the server's segment (just before the EBX® module segment) and the second at the REST application's last segment (defined in the `@ApplicationPath` annotation).

These default URI builders are already preconfigured according to:

- the data from the incoming HTTP request which constitute the base configuration,
- the Module public path prefix [p 599] which prepends the EBX® module segment, if it exists.

187.8 Exception handling

A REST Toolkit Java method can produce a standard HTTP error response by throwing a Java exception that extends the JAX-RS class `jakarta.ws.rs.WebApplicationException`. JAX-RS defines exceptions for various HTTP status codes. EBX® defines `UnprocessableEntityException`[API] that adds support for the HTTP `422` *(Unprocessable entity)* code.

Plain Java exceptions are mapped to the HTTP status code `500` *(Internal server error)*.

If a custom REST service declares an `"application/json"` media type in the `@Produces` Java annotation, then the exception content is returned as shown in the example below. For any other supported media type, no response body is returned and the REST services logging category generates an associated warning message.

```
{
  "code": 999,                      // JSON Number, HTTP error code or EBX® error code
                                    // in case of HTTP error 422 (optional)
  "errors": [                       // Additional messages to give details (optional).
    {
      "message": "Message 1"        // JSON String
    },
    {
      "message": "Message 2"
    }
  ]
}
```

See also *Media Types [p 1287]*

The REST Toolkit provides an ExceptionMapper implementation, but you can replace it with your own implementation as shown in the example below.

```
import jakarta.annotation.*;
import jakarta.ws.rs.core.*;
import jakarta.ws.rs.ext.*;

@Provider
@Priority(4999)
public class CustomExceptionMapper implements ExceptionMapper<Throwable>
{
 @Override
 public Response toResponse(final Throwable anException)
 {
  ...
 }
}
```

You can manage a particular method by directly returning the JAX-RS Response.

See also *ApplicationConfigurator.register*[API]

187.9 Monitoring

REST Toolkit events monitoring is similar to the data services log configuration. The difference is the property key which must be `ebx.log4j.category.log.restServices`.

See also

Monitoring *[p 1121]*

Configuring the EBX® logs *[p 582]*

Some additional properties are available to configure the log messages. See Configuring REST toolkit services [p 588] for further information.

187.10 Packaging and registration

All applications and components are required to be packaged into the module's Web Application (war file).

The JAX-RS libraries, except the JAX-RS client API, are already included in `ebx.jar` and must not be packaged in the war file.

See Jakarta EE deployment [p 533] for more information.

The registration of a REST Toolkit application is integrated into the EBX® module registration process. The registration class must extend `ModuleRegistrationListener`[API], declare the Servlet 3.0 annotation `@WebListener` and override the `handleContextInitialized` method.

See Module registration [p 880] for more information.

```
import jakarta.servlet.annotation.*;

import com.orchestranetworks.module.*;

@WebListener
public final class RegistrationModule extends ModuleRegistrationListener
{
 @Override
 public void handleContextInitialized(final ModuleInitializedContext aContext)
 {
  // Registers dynamically a REST Toolkit application.
  aContext.registerRESTApplication(RESTApplication.class);
 }
}
```

187.11 OpenAPI

Overview

The OpenAPI documents, generated through the TIBCO EBX® native integration, comply with the OpenAPI V3 specification . By structuring and describing the available REST resources of a REST Toolkit application and associated the operations, These documents facilitate their development and consumption. However, only the JSON format is currently supported.

Activation

To activate the OpenAPI documents generation:

- The `OpenApiApplicationPath`[API] annotation must be set on the REST Toolkit application class to document. This annotation requires the definition of a unique path used to configure the OpenAPI operations' base URI.

  > **Note**
  >
  > The OpenAPI application path will be followed by the generator version (i.e. `/{open-api-application-path}/v1`). It is recommended to not use paths like `/open-api` as there will be confusion between the generator version and the open-api specification version.

- The `OpenApiResource`[API] annotation must be set on the REST Toolkit class resources to consider during the documentation process.

Configuration

The parameter `ebx.restservices.openApi.unpublished.applications` can be used to not publish the OpenAPI endpoints on a specific environment.

```
################################################################
# Comma-separated list of canonical application class names
# whose OpenAPI descriptions will not be published.
################################################################
ebx.restservices.openApi.unpublished.applications=com.example.RestApplication
```

The parameter `ebx.restservices.openapi.style.short.name` can be used to simplify the names of the generated schemas.

```
################################################################
# If true, only the simple name of the DTO classes will be used to name the OpenAPI schemas.
# It also removes the DTO and Dto suffixes at the end of the class name.
# Default value is false.
# Note that this may cause conflicts between DTOs with the same name under different packages.
# It is not recommended, when set to true, to use the underscore character '_' in @Schema
# annotations since it cause calculation errors.
################################################################
ebx.restservices.openapi.style.short.name=true
```

Operations

The OpenAPI operations use the GET or POST HTTP method to generate the JSON document of an entire application or a single service.

The URL format for application is:

`http[s]://<host>[:<port>]/<open-api-path>/v1`

The URL format for a single service is:

`http[s]://<host>[:<port>]/<open-api-application-path>/v1/{resourcePath: [^:]*}`

Where:

- `<open-api-application-path>` corresponds to the unique path mentioned in the `OpenApiApplicationPath`[API] annotation.
- `{resourcePath: [^:]*}` corresponds to the REST resource's path to document.

HTTP codes

HTTP code	Description
`200`*(OK)*	The document has been successfully generated.
`401`*(Unauthorized)*	Authentication has failed.
`403`*(Forbidden)*	The specified resource read permission has been denied to the current user.
`404`*(Not found)*	The resource, specified in the URL, cannot be found.

Enhance documentation

MicroProfile OpenAPI

In addition to the JAX-RS annotations that will be processed, by default, to generate the OpenAPI documents, TIBCO EBX® supports the Microprofile OpenAPI annotations. These annotations are used to enhance the description without needing to rewrite portions of the OpenAPI document that are already covered by the REST Toolkit framework. The annotations can be used on both application and resource classes depending on their type.

> **Note**
>
> If not explicitly defined on a resource class, the OpenAPIDefinition annotation will be inherited from its REST Toolkit application class. This annotation describes the root document object of the OpenAPI document.

Serialization of a table record

There are further actions to take when a `ContentHolder`[API] is used in a DTO to serialize a table record [p 1290].

- The `Table`[API] annotation must be set on the field.
- The Schema annotation must tag the field for processing.

> **Note**
>
> The `ref` value of the Schema annotation must be non empty and unique in the whole application scope.

```
/**
 * DTO for a singer.
 */
public final class SingerDTO
{
 @Table(
   dataModel = "urn:ebx:module:tracks-module:/WEB-INF/ebx/schemas/tracks.xsd",
   tablePath = "/root/Singers")
 @Schema(ref = "singer")
 public ContentHolder content;
```

```
}
```

> **Note**
>
> Note that due to a limitation the metadata field is not added in this case.

Permissions

In development run mode, anonymous access to OpenAPI endpoints is granted to any user. In production and integration run modes, the OpenAPI access permissions are inherited from the ones of the underlying resources [p 1293].

See also *EBX® run mode [p 599]*

- The resource's documentation will only contain the operations accessible by the current user.
- A permission denied will be returned if the current user can not access any operation.
- The REST Toolkit application's documentation will only contain the allowed scopes of resources.
- A permission denied will be returned if the current user can not access any resource.

Visual documentation

In the development run mode [p 599], the Swagger UI is available through the UI tab Administration > Technical configuration > Modules and data models. It allows to view and interact with the documented resources of the corresponding REST Toolkit application. Visual documentation facilitates back-end implementation and client-side consumption.

The Swagger UI is also directly available through the following URLs:

URL format for application OpenAPI visualisation:

```
http[s]://<host>[:<port>]/<open-api-application-path>/v1/ui
```

URL format for a single service OpenAPI visualisation:

```
http[s]://<host>[:<port>]/<open-api-application-path>/v1/ui/{resourcePath: [^:]*}
```

Where:

- `<open-api-application-path>` corresponds to the unique path mentioned in the `OpenApiApplicationPath`[API] annotation.
- `{resourcePath: [^:]*}` corresponds to the REST resource's path to document.

> **Note**
>
> When using a reverse proxy or HTTPS configuration, the OpenAPI UI is considered an external resource. This means that external resource URLs [p 590] must be configured correctly.

187.12 **Limitations**

OpenAPI	The metadata field is not added on table record serialization. See Serialization of a table record [p 1298] for more information. DTO classes using Java generics that are not lists or maps are not supported.

CHAPTER 188

Record permission

This chapter contains the following topics:

1. Introduction
2. Lexical structure
3. Identifiers
4. Types
5. Field access
6. Operators
7. Null value management
8. If statement
9. Return statement
10. Context variables
11. Context functions
12. Other functions

188.1 Introduction

Use the record permission DSL (Domain Specific Language) to specify access rules on records of a given table.

The main goals of the record permission DSL are to not require Java coding, and to be easy to use by people without deep programming knowledge.

You can specify permission on any table using a script. The script consists of a sequence of **if then else** and **return** statements that indicate the permission for a record.

You can edit the script using the Data Model Assistant (DMA) [p 44] .

188.2 Lexical structure

Introduction

The script has following structure:

```
begin
  <statement 1>
```

```
  <statement 2>
  ...
  <last statement>
end
```

All statements except the last one must be an "if then" [p 1309] or an "if then else" [p 1310] statement.

The last statement can be an "if then" [p 1309] , "if then else" [p 1310] or "return" [p 1311] statement.

Example:

```
if isMember(administrator) then
  // Administrator has no restrictions.
  return readWrite;

if isMember('french-team') and record.Country='F' then
  //Members of 'french-team' can view and modify data for France.
  return readWrite;

if isMember('us-team') and record.Country='US' then
  //Members of 'us-team' can view and modify data for US.
  return readWrite;

// This statement is not actually needed as 'hidden' is the default permission.
return hidden;
```

Character set

The Unicode character set is supported.

Character case sensitivity

The DSL is case-sensitive.

Comments

A single line comment extends from // to the end of the current line:

```
// This is a comment
if record.LastName = 'Doe' then // This is another comment.
  return readOnly;
```

A multi-line comment extends from **/*** and ends with ***/** :

```
/* This is a sample of a multi-line
   comment */
if record.isActive then
  return readWrite;
```

Keywords

There are two types of keyword:

- **Reserved keywords** are: **if** , **then** , **else** , **begin** , **end** , **return** , **null** , **and** , **or** , **not** , **true** , **false** , **_ebx** .
- **Unreserved keywords** are all other keywords defined by the DSL.

The main difference between the two types of keywords is that unreserved ones, but not reserved ones, can be used as plain (unquoted) identifiers.

188.3 **Identifiers**

Unquoted identifier

An **unquoted identifier** is an unlimited-length sequence of letters, digits or underscore (**_**). The first character must be a letter.

Valid letters are **a to z** and **A to Z** . Valid digits are **0 to 9** .

An unquoted identifier cannot be equal to a reserved keyword.

Quoted identifiers

A quoted identifier is an unlimited length of Unicode characters

Quoted identifiers **must** be used surrounded by double quotes.

An unquoted identifier can be used surrounded by double quotes. This means that identifier "a_name" is equal to a_name.

Quoted identifiers can be reserved keywords except **"_ebx"** .

188.4 **Types**

Supported types

The following types are supported:

Type	**EBX® corresponding types**
Boolean	xs:boolean
Decimal	xs:decimal xs:int xs:integer
String	xs:string xs:anyURI xs:Name osd:text osd:html osd:email osd:password osd:color osd:resource osd:locale osd:dataspaceKey osd:datasetName
Timestamp	xs:dateTime
Date	xs:date
Time	xs:time

Literals

String literal

String literals can be any sequence of Unicode characters surrounded by single quotes. The following table displays characters that need to be replaced by an escape sequence:

Character	Escape sequence
Tab	\t
Backspace	\b
New line	\n
Carriage return	\r
Form feed	\f
Single quote	\'
Backlash	\\

A character can be specified by a Unicode escape sequence that has the format \uXXXX with XXXX the hexadecimal code of the Unicode character.

Examples

Value	Syntax
O’Harra	**'O\'Harra'**
Noël	**'No\u00EBl'**
été	**'\u00e9\u00E9'**

Note

An invalid escape or Unicode sequence generates an error at compile time.

Decimal literal

The following decimal formats are supported:

Format	Examples
Integer	546 -67
Floating point	54.987 -433.876 0.00054 -0.0032
Exponent notation	34.654e-5 -45E+65 1.543e23

Timestamp literal

Timestamp literals have the format **dt(yyyy-MM-dd hh:mm:ss.sss)** .

Seconds are optional. When seconds are not specified, 0 is assumed. Seconds can have fractions up to millisecond precision.

The dates that are not valid in the Gregorian calendar generate an error at compile time.

Examples:

```
dt(2010-01-02 00:00:00.000)
dt(2019-2-3 12:56:7)
dt(2019-2-3 12:56:7.5)
dt(2019-5-7 1:6)
```

Date literal

Date literals have format **d(yyyy-MM-dd)** .

The dates that are not valid in the Gregorian calendar generate an error at compile time.

Examples:

```
d(2010-01-02)
d(2019-2-3)
dt(2019-5-7)
```

Time literal

Time literals have the format **t(hh:mm:ss.sss)** .

Seconds are optional. When seconds arenot specified, 0 is assumed. Seconds can have fractions up to millisecond precision.

Invalid times generate an error at compile time.

Examples:

```
t(00:00:00)
t(12:56:7)
t(12:56:7.5)
t(1:6)
```

Boolean literal

Boolean literals are **true** and **false** .

188.5 Field access

Simple fields

Only access to optimized/indexed table field is supported. Dot notation is used to access tables fields. For example, a condition of current table fields whose path is **./OfficeAddress/City** would be:

```
if record.OfficeAddress.City = 'Paris' then
  return readWrite;
```

The alias **record** always refers to the current record. Depending on context, other aliases might be available.

Each step (parts separated by a dot) is an identifier [p 1303] . This means that the following quoted notation can be used for any step:

```
if record."OfficeAddress".City = 'Paris' then
  return readWrite;
```

This is useful for steps equal to a reserved keyword or using characters, such as the minus character (**-**) or dot (**.**), which are not compatible with unquoted identifiers.

At runtime, any step can evaluate to **null** . In this case, the full field expression evaluates to **null** .

Foreign table fields

You can access foreign tables by "following" foreign keys using dot notation.

In the following example, the field **Supervisor** is a foreign key:

```
if record.Supervisor.Name = 'John Doe' then
  return readWrite;
```

There can be multiple levels of foreign keys, such as in the following example:

```
if record.Supervisor.Supervisor.Supervisor.Name = 'John Doe' then
  return readOnly;
```

List (multi-valued) field access

Multi-valued fields are not supported.

Associations

Aggregate functions can take as input expressions based on an association. In the following example, the field **ManagedUsers** is an association:

```
// All users that manage at least 2 persons have read write access.
if count(record.ManagedUsers[]) >= 2 then
  return readWrite;
```

You can apply a filter on an association. You must use an alias to access fields from the association. An association alias is declared using **:** . In the following example, the alias is **u1** :

```
// All users that manage at least one person whose office is in Briton has read only access.
if exists(record.ManagedUsers:u1[u1.OfficeAddress.City='Briton']) then
  return readOnly;
```

A filter on an association can reference a field of the current record:

```
// All users that manage at least one person whose office is in the same city as user has read only access.
if exists(record.ManagedUsers:t1[t1.OfficeAddress.City=record.OfficeAddress.City]) then
  return readOnly;
```

A filter on an association can include embedded aggregate functions:

```
// All users that manage at least one person who manages at least one person has read only access.
if exists(record.ManagedUsers:t1[exists(t1.ManagedUsers[])]) then
  return readOnly;
```

A filter on an association on an embedded aggregate function can reference fields of the current record and fields of parent associations:

```
// All users that manage at least one person who manages at least one person whose office is in the same city
// as his/her two first managers in the hierarchy has read only access.
if exists(record.ManagedUsers:t1[exits(t1.ManagedUsers:t2[t2.OfficeAddress.City=record.OfficeAddress.City and
 t2.OfficeAddress.City=t1.OfficeAddress.City])]) then
  return readOnly;
```

188.6 **Operators**

By default, operation evaluation order is based on precedence and associativity. The order of evaluation can be indicated explicitly by using parentheses.

The following table shows all operators from highest to lowest precedence and their associativity:

Precedence Level	Operator	Operand type	Result type	Associativity
8	[] (access to an element of a list) . (access to fields) () (parenthesis)	List index must be a decimal.	Can be any type.	Left to right.
7	not	Boolean.	Boolean.	
6	* /	Decimal.	Decimal.	Left to right.
5	+ -	Decimal.	Decimal.	Left to right.
4	< <= > >=	String, Decimal, timestamp, date, time (3).	Boolean.	Not associative.
3	= <>	String, decimal, timestamp, date, time, boolean (3).	Boolean.	
2	and	Boolean.	Boolean.	Left to right.
1	or	Boolean.	Boolean.	Left to right.

188.7 Null value management

Arithmetic operators

An arithmetic operator (***** , / , **+** and **-**) returns **null** if any operand is **null** .

Comparison operators

A Comparison operator (< , <= , > , => , = and <>) returns **null** if any operand is **null** .

Boolean operators

Boolean operators use thread-value logic.

Truth table for **and** is:

And	**true**	**false**	**null**
true	true	false	null
false	false	false	false
null	null	false	null

Truth table for **or** is:

Or	**true**	**false**	**null**
true	true	true	true
false	true	false	null
null	true	null	null

Index expressions

An indexed expression with an index that evaluates **null** or is out of range returns **null** .

Functions

Functions usually return **null** if a parameter is **null** . An exception is the function **isNull(value)** , which never returns **null** .

188.8 If statement

"If then" statement

An "if then" statement has the following syntax:

```
if condition-expression then
```

```
  then-body-statement
```

The condition expression must evaluate to a boolean type.

If the expression evaluates to **true** , the "then" statement is executed. If the expression evaluates to **false** or **null** , the "then" statement is ignored.

The 'then' statement is a body statement [p 1310] .

"If then else" statement

An "if then else" statement has the following syntax:

```
if condition-expression then
  then-body-statement
else
  else-body-statement
```

The condition expression must be of boolean type.

If the expression evaluates to **true** , then the "then" statement is executed. If the expression evaluates to **false** or **null** , then the "else" statement is executed.

A "then" or "else" statement is a body statement [p 1310] .

Note

The expression:

```
if condition-expression then
  statement-a;
else
  statement-b;
```

might not be equivalent to:

```
if not condition-expression then
  statement-b;
else
  statement-a;
```

Indeed, if the expression is null, then the "else" statement is executed in both cases.

"Then" or "else" body statement

A body statement can be:

- A return statement [p 1311] ,
- An "if then" [p 1309] or "if then else" [p 1310] statement,
- A statement block.

A **statement block** is used to group multiple statements. It starts with the keyword **begin** and ends with the keyword **end** .

```
begin
  <statement 1>
  <statement 2>
  ...
  <last statement>
end
```

All statements except last one must be an "if then" [p 1309] or an "if then else" [p 1310] statement.

The last statement can be a "if then" [p 1309] , "if then else" [p 1310] or a "return" [p 1311] statement.

```
if isMember('sales-team')then
begin
  if record.Country='F' then
     return readWrite;
```

```
  if record.Country='UK' then
     return readOnly;
end
else
begin
  if record.Country='D' then
     return readOnly;

  if record.Country='B' then
     return readWrite;

  return hidden;
end
```

188.9 Return statement

A return statement specifies access to records that meet given conditions.

The following table shows valid return statements.

Return statement	Description
return hidden;	The record is hidden (the user has no access).
return readOnly;	The record is read only for the current user.
return readWrite;	The record can be read and modified by the current user.

If no return statement applies to a given record, the value **hidden** is assumed.

188.10 Context variables

Introduction

Record permissions can depend on the current dataspace, dataset, or session.

Dataspace

The predefined alias **dataspace** provides access to information on the current dataspace.

This alias gives access to the following fields:

Name	Type	Description
name	**string**	The name of the dataspace. Because the dataspace namespace and the snapshot namespace are independent, the returned string is only an identifier in the context of one of the namespaces. For a global dataspace or snapshot identifier, use the field **id** .
id	**string**	The persistent identifier of a dataspace or snapshot. Compared to **name** , this identifier additionally specifies whether this id is for a dataspace or a snapshot.
isSnapshot	**boolean**	Is **true** if dataspace is a snapshot and **false** if dataspace is a branch.

Example:

```
if dataspace.name = 'branch-R' then
  return readOnly;
```

Dataset

The predefined alias **dataset** provides access to information on the current dataset.

This alias has the following fields:

Name	Type	Description
name	**string**	The name of the dataset.

Example:

```
if dataset.name = 'TEST' then
  return readWrite;
```

Session

The predefined alias **session** provides access to information on the current user session.

This alias has the following fields:

Name	Type	Description
userId	**string**	The current user’s id.
userEmail	**string**	The current user’s email.
trackingInfo	**string**	The current session’s tracking information.

To check current user’s roles, see Roles [p 1313] .

Example:

```
if session.userId = 'jdoe' then
  return readWrite;
```

188.11 Context functions

Roles

The following table describes the **isMember()** function:

Syntax	Description
isMember(roleExpression…)	Returns true if the current user has at least one of the roles evaluated from the specified expressions. A **roleExpression** can be a **role literal** or a **field path** . Regarding role literals, the following **built-in roles** must be specified without quotes: **administrator** , **readOnly** , **everyone** . Custom roles are specified as string literals. Specifying a role that does not exist is not an error. (It is considered as a role with no members.) **Note** The roles **administrator** and **'administrator'** can be two distinct roles. This means that **isMember(administrator)** might not return the same result as **isMember('administrator')** . The same rule applies for other built in roles.

Example:

```
// Role literals:
if isMember('SALES', 'SUPPORT') then
  return readWrite;

// Role administrator (not surrounded by quotes):
if isMember(administrator) then
  return readOnly;

// Field paths:
if isMember(record.AuthorizedRole) then
  return readWrite;

// Different role expressions can be used at the same time:
if isMember('SALES', 'SUPPORT', administrator, record.AuthorizedRole) then
  return readWrite;
```

In order to make the permission depend on roles configured in a table, it must be related to the record by one or many associations. Then the function **isMember** must be used as follows:

```
// All records that are assigned to a zone which is authorized for the current user
// (i.e there is at least one of user's role configured in the Roles.AuthorizedRole field).
if exists(record.Zones:Z[exists(Z.Roles:R[isMember(R.AuthorizedRole)])]) then
  return readOnly;
```

Session

The following table describes the session functions:

Syntax	Description
getSessionInputParameter(parameterKey, lookupInParentSessions)	Returns the session input parameter corresponding to the specified **parameterKey** . In the context of a workflow interaction, the function looks for the parameter in the input parameters of the workflow task. When **lookupInParentSessions** is **true** , the function looks for the parameter in the parent sessions. Otherwise, it looks for the parameter only in the current session.
isInWorkflowInteraction(lookupInParentSessions)	Returns **true** if the current session is in a workflow interaction context. When **lookupInParentSessions** is **true** , the function returns **true** if a parent session is in a workflow interaction context. Otherwise, the function evaluates only the context of the current session.

Example:

```
// Give read write access if the scripted rule is called in a workflow interaction context and the 'instance'
 input parameter for the workflow task is 'Library'.
if isInWorkflowInteraction(true) and getSessionInputParameter('instance', true) = 'Library' then
  return readWrite;
```

188.12 **Other functions**

String matching functions

A string matching function takes three parameters:

- A **stringExpression** that evaluates to a string.
- A **pattern** that must be a string literal.
- An optional boolean literal **isCaseSensitive** that indicates if matching is case-sensitive. If omitted, it is assumed that matching is case-insensitive.

Syntax	Description
matches(stringExpression, pattern, isCaseSensitive)	Returns **true** if **stringExpression** matches a java regular expression **pattern** .
startsWith(stringExpression, pattern, isCaseSensitive)	Returns **true** if **stringExpression** starts with string **pattern** .
endsWith(stringExpression, pattern, isCaseSensitive)	Returns **true** if **stringExpression** ends with string **pattern** .
contains(stringExpression, pattern, isCaseSensitive)	Returns **true** if **stringExpression** contains a string **pattern** .
containsWholeWord(stringExpression, pattern, isCaseSensitive)	Returns **true** if **stringExpression** contains a whole word **pattern** .

Examples:

```
// Give read write access if first name starts with 'a', 'b' or 'c' (case-sensitive).
if matches(record.FirstName, '[a-c].*', true) then
  return readWrite;

// Give read only access if first name starts with 'Lé' (case-insensitive).
if startsWith(record.FirstName, 'Lé') then
  return readOnly;

// Give read only access if first name ends with 'my' (case-insensitive).
if endsWith(record.FirstName, 'my') then
  return readOnly;

// Give read only access if email contains with 'BeauMont@' (case-sensitive).
if contains(record.Email, 'BeauMont@', true) then
  return readOnly;

// Give read write access if last name contains the whole word 'Michel' (case-insensitive).
if containsWholeWord(record.LastName, 'Michel', false) then
  return readWrite;
```

Aggregate functions

The following aggregate function can be used with associations:

Syntax	Description
count(expression)	Returns the number of rows of an expression.
exists(expression)	Returns true if the expression evaluates to at least one row.

Examples:

```
// Give read write access if managed users count is at least 2.
if count(record.ManagedUsers[]) >= 2 then
  return readWrite;

// Give read only access if count of managed users that are not in Paris is less than 4.
if count(record.ManagedUsers:m1[m1.OfficeAddress.City<>'Paris']) < 4 then
  return readOnly;

// Give read write access if managed users count is at least 1.
if exists(record.ManagedUsers[]) then
  return readWrite;

// Give read write access if count of managed users that are not in Briton is at least 1.
if exists(record.ManagedUsers:u1[u1.OfficeAddress.City<>'Briton']) then
  return readWrite;
```

Date and time functions

The following table lists the date and time functions.

Syntax	Description
datetimeNow()	Returns the current date and time
dateNow()	Returns the current date
timeNow()	Returns the current time

Examples:

```
// Give read write access if the last possible update date hasn't expired yet.
if record.LastPossibleUpdateDate >= dateNow() then
  return readWrite;
```

```
else
  return readOnly;
```

Miscellaneous functions

The following table describes built-in functions that return a boolean.

Syntax	Description
isNull(value)	Returns true if value is null. Value can be an expression be of any type.

Examples:

```
// Give read write access if no supervisor.
if isNull(record.supervisor) then
  return readWrite;
```

EBX® Script

CHAPTER **189**

Reference

This chapter contains the following topics:

1. Introduction
2. Lexical structure
3. Identifiers
4. References
5. Types
6. Keyword raise
7. Literals
8. Operators
9. Assignments
10. Statement blocks
11. Variables and constants
12. Predefined variables
13. Functions and procedures
14. Exported functions or procedures
15. If statement
16. Loops
17. Units
18. SQL queries
19. Logging and debugging
20. Initial script

189.1 Introduction

EBX® Script is a simple and quite powerful language intended to be used anywhere where Java is currently needed like EBX® customization (computed values, table trigger, script task…).

It is based on the DSL (Domain Specific Language). DSL is a procedural and strongly statically typed language.

To write a script, the EBX® Script IDE can be used. It is an integrated development environment that provides tools and features to assist in defining scripts for function fields, table triggers and script tasks. It includes an editor with various features like syntax highlighting, contextual code completion, source compiling and publishing.

When a new function field or a table trigger is created using the DMA [p 44] , a default script is created in the EBX® Script IDE.

When a script task is created using the EBX® Script IDE, it can be specified in the Workflow Model [p 199] .

189.2 Lexical structure

Introduction

A script has following structure:

```
<unit usage statement 1>
<unit usage statement 2>
...
<unit usage statement N>

<model usage statement 1>
<model usage statement 2>
...
<model usage statement M>

<function or procedure definition 1>
<function or procedure definition 2>
...
<function or procedure definition M>
```

For more information, see Types declared in a data model [p 1324] , Unit [p 1340] and function and procedure [p 1336] .

The following example is a script of a function field.

```
// This field returns the full address for current record.
export function getValue(): string
begin
  var address := _ebx.record.FirstName | ' ' | _ebx.record.LastName;

  for street in record.OfficeAddress.Street do
  begin
    address |= '\n' | street;
  end;

  address |= '\n' | _ebx.record.OfficeAddress.ZipCode | ' ' | _ebx.record.OfficeAddress.City;
  address |= '\n' | _ebx.record.OfficeAddress.Country;

  return address;
end
```

The following example is a script of a table trigger.

```
 //This trigger adds the full name of current record.
export procedure onBeforeCreate()
begin
_ebx.record.fullName := _ebx.record.firstName | ' ' | _ebx.record.lastName;
end
```

Character set

The Unicode character set is supported.

Character case sensitivity

The DSL is case-sensitive.

Comments

A single line comment extends from // to the end of the current line:

```
// This is a comment
if _ebx.record.LastName = 'Doe' then // This is another comment.
  return true;
```

A multi-line comment extends from **/*** and ends with ***/** :

```
/* This is an example of a multi-line
   comment */
if _ebx.record.isActive then
  return false;
```

Keywords

The reserved keywords are: **and** , **or** , **not** , **uses** , **as** , **frommodelexport** , **typeof** , **tableof** , **primarykeyof** , **mutable** , **immutable** , **modifiableunmodifiable** , **persistedfunction** , **procedure** , **const** , **var** , **if** , **then** , **else** , **for** , **while** , **in** , **do** , **begin** , **end** , **return** , **true** , **false** , **raise** , **null** .

Reserved keywords cannot be used as plain (unquoted) identifiers.

189.3 Identifiers

Unquoted identifier

An **unquoted identifier** is an unlimited-length sequence of letters, digits, or underscore (**_**). The first character must be a letter or an underscore.

Valid letters are **a** to **z** and **A** to **Z** . Valid digits are **0** to **9** .

An unquoted identifier may not be equal to a reserved keyword.

Quoted identifiers

A quoted identifier is an unlimited length of any Unicode character except double quote (**"**).

Quoted identifiers **must** be used surrounded by double quotes.

An unquoted identifier can be used surrounded by double quotes.This means that identifier "a_name" is equal to a_name.

Quoted identifiers can be reserved keywords.

189.4 References

References to data models

A script may declare references to data models.

For data models deployed as an XSD file, supported syntaxes are:

```
references model short_name from module module_name as alias;
references model short_name from module module_name;
references model file_path from module module_name as alias;
```

The short name is the name of the file without the path and extension. For exemple the short name of **/WEB-INF/ebx/schemas/Directory.xsd** is **Directory** .

If no alias is specified, the alias is assumed to be equal to the short name or the file path.

It is possible to declare a data model from current module by using one of the following syntaxes:

```
references model short_name from module . as alias;
references model short_name from module .;
references model file_path from module . as alias;
```

For embedded models, use one of the following syntaxes:

```
references model short_name as alias;
references model short_name;
```

Data model references samples:

```
references model Directory from module "ebx-test" as dic;
references model Directory from module "ebx-test";
references model "/WEB-INF/ebx/schemas/userservice/Directory.xsd" from module "ebx-test" as dic;

references model Directory from module . as dic;
references model Directory from module .;
references model "/WEB-INF/ebx/schemas/userservice/Directory.xsd" from module . as dic;

references model catalog as cat;
references model catalog;
```

189.5 **Types**

Predefined types

The following predefined types are supported:

Type	Keyword	Properties	EBX® corresponding types
Boolean	boolean		xs:boolean
Decimal (unlimited precision)	decimal		xs:decimal
Integer (32 bits)	int		xs:int xs:integer
String	string	length (int greater or equal to 0)	xs:string xs:Name osd:text osd:html osd:email osd:password [1] osd:color osd:dataspaceKey osd:datasetName
Timestamp (millisecond precision and without time-zone)	timestamp	year month (1 to 12) day (1 to 31) hour (0 to 23) minute (0 to 59) second (0.000 to 59.999)	xs:dateTime
Date (without time-zone)	date	year month (1 to 12) day (1 to 31)	xs:date
Time (millisecond precision)	time	hour (0 to 23) minute (0 to 59) second (0.000 to 59.999)	xs:time
Locale	locale		osd:locale
URI (Uniform Resource Identifier)	uri		xs:anyURI
Resource	resource		osd:resource
Dataspace	dataspace	name	

Type	Keyword	Properties	EBX® corresponding types
		identifier (the name prefixed by 'B' or 'V') isSnapshot (Is true if the dataspace is a snapshot)	
Dataset	dataset	name dataspace	
Transaction	transaction	dataspace isAllPrivileges (read/write boolean property) executionInformation (read/write string property) executionTimestamp executionUUIDString	

[1] It is not possible to define a function field for type **osd:password** .

Complex types

A complex type is the type of a group (complex) node defined in an EBX® schema.

For more information, see the chapter complex variables [p 1334]

List types

The DSL supports lists. Declare a list by using the following syntax:

```
list<item_type>
```

A list is unmodifiable by default. You can declare a modifiable list by using following syntax:

```
modifiable list<item_type>
```

An item cannot be added, removed, or replaced if the list is unmodifiable.

Use indexed notation to set or get a value. (The first index is 0):

```
// Set first value of cities.
cities[0] := 'Paris';

// Get the second item of cities.
return cities[1];
```

An index can be any expression of int or decimal type. If the index is a decimal, the fractional part is ignored.

A get expression returns null if the index is null, negative, or out of range.

A set expression reports an error at runtime if the index is null, negative, or out of range.

The property **size** returns the size of the list:

```
var size := cities.size;
```

You can iterate a list.For more information, see for loops [p 1339] .

Multi-value schema fields

Schema fields that are multi-valued are considered of list type.

A multi-valued schema field is a field that has its maximum number of values (maxOccurs) greater than one or set to "unbounded".

Indexed notation is used to access or set a value. The first index is 0. In the following example, record field 'OfficeAddress/street' is multi-valued:

```
// Return the second line of the street part of the address.
return OfficeAddress.street[1].
```

The **typeof** keyword can be used with multi-valued fields (see chapter keyword typeof [p 1324]).

Keyword typeof

Use the keyword **typeof** to specify a type depending on a field or variable.

For example, to reference the type of record field **OfficeAddress** , use the following syntax:

```
typeof record.OfficeAddress
```

If a field is multi-valued, it is considered a list. You can reference the type of an item of the list using following syntax:

```
typeof record.Addresses[*]
```

Types declared in a data model

To reference the type of a field in **current** data model that has path **/root/User/Address** , use the following syntax:

```
var address: .User.Address;
```

Note that '/' are replaced by '.' and that first step is omitted.

To get the type of a field in from a data model that has path **/root/Sales/Amount** in a data model declared with alias 'hist', use the following syntax:

```
var amount: hist.Sales.Amount
```

If a field is multi-valued, it is considered a list. You can reference the type of an item of the list using following syntax:

```
var products: hist.Sales.products[*]
```

Record, table, primary key

If path "/root/test/table" of a data model **sample** is for a model, use one of the following syntaxes for specifying a record:

```
var immutableRecord1: sample.test.table;
var immutableRecord2: immutable sample.test.table
var mutableRecord: mutable sample.test.table
```

When no modifier is specified, a type is immutable.

Use the following syntax for specifying a table:

```
var aTable: tableof sample.test.table;
aTable := db.findTable<.directory.Customer>(_ebx.dataset);
```

Use one of the following syntaxes for specifying a primary key:

```
var immutablePrimaryKey1: primarykeyof sample.test.table;
var immutablePrimaryKey: immutable primarykeyof sample.test.table
var mutablePrimaryKey: mutable primarykeyof sample.test.table
```

When no modifier is specified, the primary key is immutable.

Type modifiers

A type may be:

- **immutable** : it is not possible to modify an item of this type.
- **mutable** : it may be possible to modify an item of this type.

A simple type is always immutable.

You can assign a mutable value to a variable or a return value of a compatible immutable type, but assigning an immutable value to a mutable variable or return value generates an error at compile time.

You can specify an immutable type using one of following syntax:

```
immutable typeof record.OfficeAddress
immutable .User.Address
immutable hist.Sales.Amount
```

You can specify an mutable type using one of following syntax:

```
mutable typeof record.OfficeAddress
mutable .User.Address
mutable hist.Sales.Amount
```

Some types may not support **immutable** or **mutable** keywords and generate an error at compile time.

A list may be:

- **unmodifiable** : it is not possible to remove on insert an item. Items may be immutable or not.
- **modifiable** : it is possible to remove on insert an item. Items may be immutable or not.

You can assign a modifiable list to a variable or a return value of a compatible unmodifiable list type, but assigning an unmodifiable list to a modifiable variable or return list generates an error at compile time.

You can modify a mutable complex or list object, but other constraints may apply that can prevent modifying some attributes of an object.

You can specify an unmodifiable list type using following syntax:

```
unmodifiable typeof  _ebx.record.Addresses[*]
unmodifiable .User.children[*]
unmodifiable hist.Sales.products[*]
```

You can specify a modifiable list type using following syntax:

```
modifiable typeof  _ebx.record.Addresses[*]
modifiable .User.children[*]
modifiable hist.Sales.products[*]
```

189.6 **Keyword raise**

Use the keyword **raise** to reports an error during the execution of a trigger or a function field.

The following syntax is used to raise an error without a message:

```
raise code;
```

The following syntax is used to raise an error with a message:

```
raise code, message;
```

- code is the error identifier. It must be an integer 32 value.

- message specifies the error message to be displayed.It is an optional string value.

Example of using the keyword **raise** in a function field:

```
// this function raises an error when the name is empty.
uses core.string as string;
export function getValue(): string
begin
 var name := _ebx.record.name;
 if not string.isEmpty(name) then
  return name;

//value cannot be empty!
 raise 341i;
end
```

Example of using the keyword **raise** in a table trigger:

```
// this trigger checks sales amount and reports an error if the allowed amount is exceeded.
export procedure onBeforeCreate()
begin
if (_ebx.record.amount > 1000) then
raise 450 , 'Sales Amount '| _ebx.record.amount |' has exceeded the allowed amount of 1000';
end
```

189.7 **Literals**

String literal

String literals can be any sequence of Unicode characters surrounded by single quotes. The following table displays characters that need to be replaced by an escape sequence:

Character	**Escape sequence**
Tab	\t
Backspace	\b
New line	\n
Carriage return	\r
Form feed	\f
Single quote	\'
Backlash	\\

Specify a character by using a Unicode escape sequence that has format **\uXXXX** , where XXXX is the hexadecimal code of the Unicode character.

Examples

Value	Syntax
O’Harra	**'O\'Harra'**
Noël	**'No\u00EBl'**
été	**'\u00e9t\u00E9'**

Note

An invalid escape or Unicode sequence generates an error at compile time.

Decimal literal

The following decimal formats are supported:

Format	Examples
Without fraction	546 -67
Floating point	54.987 -433.876 0.00054 -0.0032
Exponent notation	34.654e-5 -45E+65 1.543e23

Integer (32 bits) literal

Integer 32 bits literals have a format similar to **90324i** or **-543i** .

The value must be between **-2147483648i** and **2147483647i** inclusive.

Attention: If the **i** letter is missing the value will be considered a decimal.

Timestamp literal

Timestamp literals have the format **dt(yyyy-MM-dd hh:mm:ss.sss)** .

Seconds are optional. When they are not specified, 0 is assumed. Seconds can have fractions up to millisecond precision.

Any dates that are not valid in the Gregorian calendar generate errors at compile time.

Examples:

```
dt(2010-01-02 00:00:00.000)
dt(2019-2-3 12:56:7)
dt(2019-2-3 12:56:7.5)
dt(2019-5-7 1:6)
```

Date literal

Date literals have the format **d(yyyy-MM-dd)** .

Any dates that are not valid in the Gregorian calendar generate errors at compile time.

Examples:

```
d(2010-01-02)
d(2019-2-3)
dt(2019-5-7)
```

Time literal

Time literals have the format **t(hh:mm:ss.sss)** .

Seconds are optional. When they are not specified, 0 is assumed. Seconds can have fractions up to millisecond precision.

Invalid times generate an error at compile time.

Examples:

```
t(00:00:00)
t(12:56:7)
t(12:56:7.5)
t(1:6)
```

Boolean literal

A Boolean literal is either the keyword **true** or the keyword **false** .

Null literal

A null literal is the keyword **null** .

Use this literal only in the following situations:

- To set a variable or field,
- As a return value.

Note

It is not possible to use this keyword to test if a variable or a field is **null** . Instead, use the function **isNull()** .

189.8 **Operators**

Precedence

By default, operation evaluation order is based on precedence and associativity. Use parentheses to explicitly indicate the order of evaluation.

The following table shows all operators, from highest to lowest precedence, and their associativity:

Precedence Level	Operator	Operand type	Result type	Associativity
9	(access to an element of a list) . (access to fields) () (parenthesis)	List index must be an int or a decimal.	Can be any type.	Left to right.
8	not	boolean	boolean	
7	* /	decimal or int	decimal	Left to right.
6	+ -	decimal or int	decimal	Left to right.
5	| (string concatenation)	string	string	Left to right.
4	< <= > >=	string, decimal, int, timestamp, date, time (3).	boolean	Not associative.
3	= <>	string, decimal, int, timestamp, date, time, boolean (3).	boolean	
2	and	boolean	boolean	Left to right.
1	or	boolean	boolean	Left to right.

Arithmetic operators

All arithmetic operators (***** , / , **+** and **-**) are evaluated in **decimal** with 34 digits precision (IEEE 754R Decimal128 format). Values of **ìnt** type are always converted to **decimal** before comparison.

Result is **null** if any operand is **null** .

A **decimal** value is automatically converted when assigned to an **int** type variable. An error occurs if the value cannot be converted because it is not an integer, is greater than 2147483647, or is less than -2147483648.

String concatenation operator

The string concatenation operator (**|**) operands can be of any type and are automatically converted to a string before concatenation. The **null** value is replaced by the empty string.

The string concatenation operator (**|**) replaces all null operands by the empty string before executing the concatenation.

Boolean operators

Boolean operators use **three-valued logic** also called ternary logic. This logic is also used by SQL.

Boolean variables can have one of three possible truth values: **true** , **false** and **null** .

Just like binary logic, three-valued logic has operators such as **and** , **or** and **not** . However, the behavior of these operators is extended to handle the **null** value in the following way:

- **not** : Inverts **true** to **false** , **false** to **true** but leaves **null** unchanged.
- **and** : If all operands are **true** , the result is **true** . If one operand is **false** , the result is **false** . If at least one operand is **null** and none are **false** the result is **null** . This means that one or more **false** operands override all **null** operands.
- **or** : If one operand is **true** , the result is **true** . If all operands are **false** , the result is **false** . If at least one operand is **null** and none are **true** , the result is **null** . This means that that one or more **true** operands override all **null** operands.

Comparison operators

A comparison operator (< , <= , > , => , = and <>) compares two operands of same type. Values of **int** type are always converted to **decimal** before comparison.

The returned value of a comparator is of **boolean** type. This value is **true** or **false** only if all operands are not **null** . It is **null** if any of its operand is **null** .

Built-in and unit functions

Built-in and unit functions usually return **null** if a parameter is **null** .

There can be exceptions, so be sure to read the API documentation [p 1386] .

189.9 **Assignments**

The following table shows all assignment operators:

Operator	Operand type	Description
:=	Can be used with any type.	Standard assignment.
|=	string	a |= b is equivalent to a := a | b
+=	decimal, int	a += b is equivalent to a := a + b
-=	decimal, int	a -= b is equivalent to a := a - b
*=	decimal, int	a *= b is equivalent to a := a * b
/=	decimal, int	a /= b is equivalent to a := a / b

189.10 Statement blocks

A statement block is used to group multiple statements. It starts by the keyword **begin** and ends with the keyword **end** . Most statements need to be terminated by a **;** :

```
begin
  statement_1;
  statement_2;
end
```

A statement block can contain one or more statement blocks that can also contain statement blocks:

```
begin
  statement_1;
  begin
    statement_2_1;
    statement_2_3;
    begin
      statement_3_1;
      statement_3_2;
    end
  end
  statement_3;
end
```

189.11 Variables and constants

Introduction

Any statement in a block can declare a variable or a constant.

A variable or a constant always has a type that is fixed when the variable is declared.

A value can be assigned to a variable using the assignment operator **:=** .

Its is an error to assign a value of the wrong type to a variable, or to change the value of a constant after its declaration.

Assignment statements

Declaration with type detection

A variable or constant type can be detected automatically if initialized when declared. This feature is called type inference.

Syntax for a variable is:

```
var variable_name := a_value;
```

The syntax for a constant is:

```
const const_name := a_value;
```

Examples:

```
begin
  var firstName := 'John';        // Variable of type string.
  var age := 30i;                 // Variable of type int.
  var cost := 34.54;              // Variable of type decimal.
  var value := 43;                // Variable of type decimal not int
                                  // because the letter `i` is missing.
  var address := record.address;  // Variable of an immutable complex.

  const MAX_AGE := 100i;              // Constant of type int.
  const DEFAULT_COUNTRY := 'France'; //  Constant of type string.
end
```

Note

A constant is not necessarily immutable.

Explicit type definition

It is possible to explicitly specify the variable's type.

The syntax for a variable is:

```
// Following variable initial value is null.
var variable_1_name : variable_1_type;

// Value a_value must be compatible with type variable_1_type.
var variable_2_name : variable_1_type := a_value;
```

The syntax for a constant is:

```
const const_name : variable_1_type := a_value;
```

Examples:

```
begin
  var firstName : string;
  firstName := 'John';

  var age : int;
  age := 30;
  var address : typeof record.address;
  address := record.address;

  const MAX_AGE : int := 100;
  const DEFAULT_COUNTRY : string := 'France';
end
```

Variables that are not initialized have a null value:

```
begin
  var firstName : string;
  if isNull(fistName) then
     doSomething(); // Statement will be executed.
end
```

The following statements have errors:

```
begin
  // Following will not compile because variable is not initialized and no type
  // is defined.
  var firstName;
  // Following will not compile because variable type is decimal and value is
  // a string.
  var amount : decimal := 'test';
end
```

Reference variables (ref)

Reference variables store a mutable reference to a value of a component type. They behave like a boxed value:

- Reads auto-dereference to the referenced value.
- Assignments write to the referenced value.

Syntax

The **ref** declaration mirrors **var** in shape. The type is optional when an initializer is provided.

```
// With explicit type and initializer
ref counter: int := 0i;

// With explicit type only (initialized to null)
ref name: string;
```

```
// With initializer only (type inferred from initializer)
ref price := 12.50;
```

If no type is specified, an initializer with a non-null type is required. Otherwise, a compile-time error is reported (same rule as **var**).

Usage

Reads and writes on a **ref** variable target the referenced value directly; you do not need to access a **.value** field.

```
begin
  ref total: decimal := 0.0;
  total += 10;        // increments the referenced value
  var x := total;     // reads the referenced value

  ref title := 'Mr';  // type inferred as string
  title := title | 's';
end
```

You can also use **ref** with lists and indexes; assignments behave like regular list element stores.

```
uses core.list as list;
begin
  ref numbers: modifiable list<int> := list.of<int>();
  list.add(numbers, 1i);
  list.add(numbers, 2i);
end
```

Note: The word **ref** is not a globally reserved keyword (for backward compatibility). You can use **ref** as an identifier without quotes. In declaration context, when **ref** appears at the beginning of a statement, it is parsed as a reference variable declaration; if you truly need a variable literally named **ref** in that position, use a quoted identifier (for example, **"ref"**).

Using ref with transactions

ref is especially useful to capture a value computed inside an **execute transaction** block and return it afterwards.

```
uses core.data as db;

export function updateCounter(): int
begin
  ref result: int; // will be assigned inside the transaction block

  execute transaction tr on db.findDataspace('sample')
  begin
    // Compute under transaction context (read/update data as needed)
    result := 42i;
  end

  // `result` carries the value out of the transaction block
  return result;
end
```

Scope

A variable or constant scope spans from its declaration to the end of the statement block where it was declared:

```
begin
  var firstName := 'John';
  var lastName := 'Doe';

  begin
    // Following is OK because firstName and lastName are visible.
    var fullName := firstName | ' ' | lastName;
    ....
  end

  // Following will NOT compile because fullName is out of the scope.
  var message := 'Please contact ' | fullName;
```

```
end
```

A variable with same name cannot exist in same block or sub blocks:

```
begin
  var firstName := 'John';

  begin
    var firstName := 'Bob'; // Error! Variable already declared.
    ....
  end
end
```

The following is correct:

```
begin
  begin
    var firstName := 'Bob';
    ....
  end
  // The following is not an error, because block where previous variable was
  // declared is ended.
  var firstName := 'John';
end
```

Complex variables

Variable or constants of complex type can be declared using type detection or typeof keyword [p 1324] .

Dot notation is used to access fields of a variable of complex type.

Examples:

```
// Declaration using type detection.
var address1 := _ebx.record.OfficeAddress;

// Declaration using typeof notation.
var address2 : typeof _ebx.record.OfficeAddress;
address2 := address1;
```

Each step (parts separated by a dot) is an identifier [p 1320] . This means that following quoted notation can be used for any step:

```
var city := _ebx.record."OfficeAddress".City;
```

This is useful for steps equal to a reserved keyword or using characters, such as the minus character (**-**) or dot (**.**), that are not compatible with unquoted identifiers.

At runtime, any step can evaluate to **null** . In this case the full field expression evaluates to **null** .

Multi-valued fields

Multi-valued fields are treated as ordered lists. Index notation **[index]** is used to access an item of the list. Indexes are 0 based (first index is 0).

In the following example, the field "Address" is multivalued:

```
var city := _ebx.record.Address[1].City;
```

If an index is out of bound, the indexed expression will return **null** . In this case the full field expression evaluates to **null** .

An index may be a **decimal** or int **expression** . If **decimal`** the fractional part will be ignored.

189.12 Predefined variables

Introduction

Predefined variables allow access to information of the context. The following predefined variables described in this section are common to contexts (function field, table trigger, dataset trigger). They are constant and of an immutable complex type.

Note that there are some specific predefined variables for each contexts: function field, table trigger and script task.

For details on these specific variables, see the links Function field [p 1343] , Table trigger [p 1351] and Script Task [p 1345] .

Dataspace

The predefined variable **_ebx.dataspace** provides access to information on the current dataspace.

This variable has following fields:

Name	Type	Description
name	**string**	The name of the dataspace. Since the dataspace namespace and the snapshot namespace are independent, the returned string is only an identifier in the context of one of the namespaces. For a global dataspace or snapshot identifier, use field **id** .
identifier	**string**	The persistent identifier of a dataspace or snapshot. Compared to **name** , this identifier additionally specifies whether this id is for a dataspace or a snapshot.
isSnapshot	**boolean**	Is **true** if dataspace is a snapshot and **false** if dataspace is a branch.

Example:

```
// Returns details on the current dataspace.
export function getValue(): string
begin
  if _ebx.dataspace.isSnapshot then
    return 'Snapshot: ' |  dataspace.name;
  else
    return 'Branch: ' |  dataspace.name;
end
```

Dataset

The predefined variable **_ebx.dataset** provides access to information on the current dataset.

This variable has the following fields:

Name	Type	Description
name	**string**	The name of the dataset
dataspace	**dataspace**	The dataspace of the dataset

Example:

```
// Returns the current dataset name.
export function getValue(): string
begin
  return _ebx.dataset.name;
end
```

Node: The predefined variable **_ebx.dataset** is not available in the context of a dataset trigger when the trigger is of type **afterDelete** .

Root

The predefined variable **_ebx.root** is available when the current script is for a table trigger, dataset trigger or for a function field. It provides read-only access to the fields of current datasets.

Its fields are defined by the current EBX® schema.

Examples:

```
// this function field returns information on current dataset data.
export function getValue(): string
begin
 return _ebx.root.City | ' ' | _ebx.root.Region;
end

// this table trigger logs information of current dataset data.
uses core.log as log;
export procedure onNewContext()
begin
 log.info(_ebx.root.City | ' ' | _ebx.root.Region);
end

// this dataset trigger logs information of current dataset data.
uses core.log as log;
export procedure onAfterDuplicate()
begin
 log.info(_ebx.root.Country);
end
```

189.13 **Functions and procedures**

Functions

Functions are methods that return a value.

A function with no parameters has the following syntax:

```
function function_name(): return_value_type
begin
  ....
  return a_value;
end
```

A function with parameters has the following syntax:

```
function function_name(
  parameter_1 : parameter_1_type,
  parameter_2 : parameter_2_type...,): return_value_type
```

```
begin
  ....
  return a_value;
end
```

It’s possible to declare a parameter constant using the **const** keyword:

```
function function_name(const parameter_1 : parameter_1_type): return_value_type
begin
  ....
  return a_value;
end
```

The last statement of a function must always be a return statement. The function can include as many return statement as necessary.

Example:

```
function getFullName(firstName: string, lastName: string): string
begin
  if isNull(firstName) then
    return lastName;

  return firstName | ' ' | lastName;
end
```

It is illegal to return a value different from the one declared:

```
function getValue(): string
begin
    return 0; // Error! Return type is decimal, not string.
end
```

Procedures

Procedures are methods that do not return a value.

A procedure with no parameters has the following syntax:

```
procedure procedure_name()
begin
  ....
end
```

A procedure with parameters has the following syntax:

```
procedure procedure_name(
  parameter_1 : parameter_1_type,
  parameter_2 : parameter_2_type...)
begin
  ....
end
```

It’s possible to declare a parameter constant using the **const** keyword:

```
function procedure_name(const parameter_1 : parameter_1_type): return_value_type
begin
  ....
  return a_value;
end
```

A procedure cannot have a return statement, but the function can include as many return statements as necessary. A procedure statement cannot return a value.

189.14 **Exported functions or procedures**

An exported function can be called directly by EBX®. Scripts for REST services and constraint validation can include multiple exported functions. All other script types allow only one exported function or procedure per script. Its signature (name, return type) depends on the context.

For details on the exported function or procedure of each context, see the Function field [p 1343] and the Table trigger [p 1351] .

Parameters

Simple type parameters are passed by value. This means that a function or a procedure receives a copy of the original value.

Complex and list types are passed by reference. This means that a function or a procedure receives the original object. If the function or procedure modifies the object, the original one is modified.

189.15 If statement

"If then" statements

An "if then" statement has the following syntax:

```
if condition-expression then
  then-statement
```

The condition expression must evaluate to a boolean type.

The **then** statement can be a statement block [p 1331] . It is executed only if the condition expression is **true** .

"If then else" statements

An "if then else" statement has the following syntax:

```
if condition-expression then
  then-statement
else
  else-statement
```

The condition expression must be of boolean type.

A 'then' or 'else' statement can be a statement block [p 1331] . The **then** statement is executed only if the condition expression is **true** . The **else** statement is executed only if the condition expression is **false** or **null** .

If statement and three-valued logic

The following code:

```
if condition-expression then
  statements-a;
else
  statements-b;
```

Is not equivalent to:

```
if not condition-expression then
  statements-b;
else
  statements-a;
```

Indeed, if the expression is **null** , the **else** statement is executed in both cases.

In the following example, the **reportError()** statement is **not executed** if either a or b is **null** .

```
if a <> b then
begin
  reportError();
  return;
end
```

```
doSomething();
return;
```

This might not be the intended behavior. Instead, one may use the following code:

```
if a = b then
  doSomething();
else
  reportError();
return;
```

Or

```
if isTrueOrNull(a <> b) then
begin
  reportError();
  return;
end

doSomething();
return;
```

The previous code examples consider that having **null** values is always an error. If having both **a** and **b** as **null** is not an error, one may use code similar to the following:

```
if (isNull(a) and isNull(b)) or (a = b) then
  doSomething();
else
  reportError();
```

189.16 **Loops**

"For in do" loops

A "for in do" loop statement is used to select each item of a list and execute a block statement. It has following syntax:

```
for item_variable_name in list do
begin
  statement_a;
  statement_b;
  ...
end
```

The block statement is executed once for each value of the list. At each iteration, the read-only item variable takes a value of the list in the order of the list.

The name of the item variable must be unique in the current scope or an error will be generated at compile time.

If a single statement must be executed for each item, the following simpler syntax can be used:

```
for item_variable_name in list do statement;
```

The following example iterates a list of complex:

```
// Concatenate all city addresses in a single string.
var cities := '';
for address in _ebx.record.Addresses do
begin
  if cities <> '' then cities |= ', ';
  cities |= address.city;
end
```

"While do" loops

A "while do" loop statement is used to execute a statement block until a condition is **true** . It has following syntax:

```
while condition do
begin
  statement_a;
  statement_b;
  ...
end
```

If a single statement must be executed, the following simpler syntax can be used:

```
while condition do statement_a;
```

The following example calculates factorial of a value:

```
function factorial(value : decimal): decimal
begin
 var factorial := value;
 while(value > 1) do
 begin
   value -= 1;
   factorial *= value;
 end
 return factorial;
end
```

A **decimal** type is used instead of an **int** type to avoid out of range errors.

189.17 **Units**

EBX® provides an API that is packaged in "units".

A unit can define multiple function or procedures.

Except for the default one, a unit must be declared before usage. The declaration must be at the top of the script and must follow these types of syntaxes:

```
uses package_name.unit_name_a;
uses package_name.unit_name_b as alias_b;
```

Currently, **package_name** is always **core** . A unit alias must be unique in a script.

Methods can be referenced using following syntaxes:

```
value1 := package_name.unit_name_a.function_a();
value2 := package_name.unit_name_a.function_b(parameter1, parameter2);

value3 := alias_a.function_c();
value4 := alias_a.function_d(parameter1, parameter2);

package_name.unit_name_a.procedure_a();
package_name.unit_name_a.procedure_b(parameter1, parameter2);

alias_a.procedure_c();
alias_a.procedure_d(parameter1, parameter2);
```

The following example uses the **core.list** unit to create a list:

```
uses core.list as list;

export function getValue(): typeof _ebx.record.Cities
begin
  return list.of('Paris', 'Bruxelles', 'Berlin');
end
```

For details on the provided units, see the API Documentation [p 1386] .

189.18 **SQL queries**

An EBX® Script can execute SQL queries (select expression).

There are two types of SQL queries: **embedded** and **dynamic** .

Embedded queries

Whenever possible, use embedded queries because they are validated at compile time. This allows errors to be detected at compile time rather than at runtime.

Embedded queries must be between back-quote and prefixed by **sql** .

Sample:

```
uses core.data as d;
// References to data models used by the queries are declared at the beginning of the script.
references model myModel from module "ebx-test" as m;

// Syntax of the query is validated at compile time.
var query := sql `select u.f_name as firstName, u.l_name as lastName from m.users u where location = ?`;

// Adds the dataset corresponding to the model 'm' to the query.
d.addDataset(query, ‘m', d.findDataset(d.findDataspace('myDataspace'), 'myDataset'));

// Sets the parameter value for the query. Parameter existence and type are checked at compile time.
// The parameter name is parameter_<position>.
query.parameter_0 := 'France';

// Executes the query and fetches rows.
for tuple in d.execute(query) do
begin
   // Retrieves the values of the row. Existence and type are checked at compile time.
   // The name of a value is the alias set in the query or the column name of no alias are set.
   var firstName := tuple.firstName;
   var lastName := tuple.lastName;
end;
d.close(query); // Optional. Query is closed automatically if the last tuple is reached or when script execution
 ends.
```

Dynamic queries

Dynamic queries are built at runtime and therefore cannot not be validated at compile time. Illegal queries will generate an error at runtime.

Sample:

```
uses core.data as d;
// References to data models used by the queries are declared at the beginning of the script.
references model myModel from module "ebx-test" as m;

// Builds the query expression.
var queryExpression := 'select u.f_name as firstName, u.l_name as lastName from m.users u';
queryExpression += ' where location = ?';

// Creates the query.
var query := d.createQuery(queryExpression);

// Adds the dataset corresponding to the model 'm' to the query.
db.addDataset(query, ‘m', d.findDataset(d.findDataspace('myDataspace'), 'myDataset'));

// Sets the parameter value for the query. Parameter existence and type are checked at runtime.
d.setParameter(query, 0,  'France');

// Executes the query and fetches rows.
for tuple in d.execute(query) do
begin
   // Retrieves the values of the row. Existence and type are checked at runtime.
   // The name of a value is the alias set in the query or the column name of no alias are set.
   var firstName := d.field<string>(tuple, 'firstName');
   var lastName := d.field<string>(tuple, 'lastName');
end;
d.close(query); // Optional. Query is closed automatically if the last tuple is reached or when script execution
 ends.
```

Query session

A property **query.session** is provided to allow the script to explicitly set a session to the sql query. To disable access control of the sql query, a **null** value can be set for the query session.

```
// create a sql query
var query := sql `Select b.col from tableB b`;
// set a session to sql query
query.session:= ...;

// get userId and locale of the query session
var userId:=query.session.userId;
var locale:=query.session.locale;
```

Note: In the context of triggers and script tasks, the current user session is set by default to the SQL query. In the context of function values, no default user session is set to the SQL query.

189.19 Logging and debugging

The unit unit.log [p 1386] provides a function that can be used to log a message.

A message logged by scripts can be viewed using **Administration>Repository management>Scripting** EBX® menu.

Another useful feature is that if a runtime encounters an error while executing a script, it is usually logged with the line in the script where the error occurred.

189.20 Initial script

When a new function field or table trigger is created using the DMA [p 44] , an initial script is created.

The following example is a script created for a table trigger:

```
uses core.datetime as datetime;
export procedure onBeforeCreate()
begin
record.inscriptionDate := datetime.now();
end
```

The following example is a script created for a function field of type string:

```
// Returns the full name.
export function getValue(): string
begin
 return _ebx.record.FirstName | ' ' | _ebx.record.LastName;
end
```

For details on initial script, see the Function field [p 1343] and the Table trigger [p 1351] .

CHAPTER 190

Function field

This chapter contains the following topics:

1. Introduction
2. Initial script
3. Predefined variable

190.1 Introduction

The DSL (Domain Specific Language) is used to define a function field. For more information on DSL, see Reference [p 1318]

To create and modify a function field, use the Data Model Assistant (DMA) [p 44] .

The Data Model Assistant (DMA) includes a script editor that provides contextual code completion.

190.2 Initial script

Introduction

When a function field is first edited using use the Data Model Assistant (DMA) [p 44] , EBX® provides an initial script with the correct definition of the exported function.

The following example is a script created for a function field of type string:

```
// Returns the full name.
export function getValue(): string
begin
  if isNull(record.FirstName) then
    return _ebx.record.LastName;

 return _ebx.record.FirstName | ' ' | _ebx.record.LastName;
end
```

Exported function

An exported function can be called directly by EBX®. Only one exported function is allowed per script. Its signature (name, and return type) depends on the type of the function field and should not be changed..

For the field of a record, definition must be similar to:

```
export function getValue(): typeof _ebx.record.<path>
begin
 ...
```

```
end
```

For the field of a dataset, definition must be similar to:

```
export function getValue(): typeof _ebx.root.<path>
begin
 ...
end
```

190.3 Predefined variable

Introduction

Predefined variables allow access to the context of the function field. The following contextual variable is specific to function fields. It is constant and of an immutable complex type. For more details on common predefined variables, see the Reference [p 1318] .

Record

The predefined variable **_ebx.record** is available only if the current script is for a function field of a table. It allows read-only access to the current record.

Its fields are defined by the current EBX® schema.

Example:

```
// Returns the full name.
export function getValue(): string
begin
 return _ebx.record.FirstName | ' ' | _ebx.record.LastName;
end
```

CHAPTER 191

Script task

This chapter contains the following topics:

1. Introduction
2. Initial script
3. Predefined variables
4. Transaction

191.1 Introduction

A Workflow model is defined by a sequence of steps. A step may be a script task. The script task is an automatic task that is performed without human user involvement. It is based on the procedural language DSL (Domain Specific Language) to define custom operations. For more information on DSL, see Reference [p 1318] .

When the task is ready to start, the engine will execute the script. Once the script is completed, the task is also be completed.

To create and modify a script task, use the EBX® Script IDE editor that provides contextual code completion.

191.2 Initial script

Introduction

When a script task is first edited using use the script IDE, EBX® provides an initial script with the correct definition of the exported procedure.

An initial script task has following structure:

```
<unit usage statement 1>
<unit usage statement 2>
...
<unit usage statement N>

export procedure executeScriptTask()
begin
  ...
  <statement 1>
  <statement 2>
  ...
end
```

The following example is a script of a workflow task that logs a message.

```
uses core.log as log;
export procedure executeScriptTask()
begin
 log.info('A new script task is executed');
end
```

Exported procedure

An exported procedure can be called directly by EBX®. Only one exported procedure is allowed per script task.

The signature of the script task procedure is the following:

```
export procedure executeScriptTask()
```

This signature can not be changed.

191.3 Predefined variables

Introduction

Predefined variables allow access to the context of a script task. The following contextual variable **_ebx.parameters** is specific to script task. It is constant and of an immutable complex type. Note that the common predefined variables of contexts (function field, table trigger) described in Reference [p 1318] can not used in the context of script task.

_ebx.parameters

The predefined variable **_ebx.parameters** provides access to the parameters of the script task context. The parameters of the script task can be **input** and/or **output** . They are defined by using the script IDE. The predefined variable **_ebx.workflow** provides access to information about the workflow model like the identifier, the creator, the last modification date, etc. These parameter are read-only.

The parameters of the variable **_ebx.parameters** have the following properties:

Property	Description
name	the name of the parameter.
type	The type of the parameter can be string, boolean, locale…
isInput	When this property is checked, the parameter is defined as an input. The parameter is not modifiable.
isOutput	When this property is checked, the parameter is defined as an output. The parameter is readable and modifiable by the script. If the **output** parameter is not also **input** , its initial value is null.

The syntax to get the value of an **input** parameter is :

```
var a_value := _ebx.parameters.input_parameter_name;
```

where **input_parameter_name** is the name of the **input** parameter.

The syntax to set the value of an **output** parameter is :

```
_ebx.parameters.output_parameter_name := a_value;
```

where **output_parameter_name** is the name of the **output** parameter.

Examples:

```
export procedure executeScriptTask()
begin
  var id:=_ebx.parameters.id; // input parameter of type string.
  var active:=_ebx.parameters.isActive; // input parameter of type boolean.

  _ebx.parameters.country:='USA'; // output parameter of type string.
  _ebx.parameters.locale:=locale.of('fr','FR'); // output parameter of type locale.
  _ebx.parameters.isActive:= false; // output parameter of type boolean.

  _ebx.parameters.message='Task A :' | _ebx.parameters.message; // input output parameter of type string.
end

// This task saves information of a person in the table
export procedure executeScriptTask()
begin
  var id:=_ebx.parameters.id;
  var firstName:=_ebx.parameters.firstName;
  var lastName:=_ebx.parameters.lastName;
  _ebx.parameters.fullName := firstName | ' ' | lastName;
  savePerson(id,firstName, lastName,_ebx.task.fullName);
end
```

_ebx.workflow

The predefined variable **_ebx.workflow** provides a read-only access to information on the current workflow model. This contextual variable is constant and of an immutable complex type.

It has the following fields:

Name	Type	Description
id	**string**	The identifier of the workflow model
publicationId	**string**	The publication identifier of the workflow model
label	**message**	The label of the workflow model
description	**message**	The description of the workflow model
creationDate	**date**	The creation date of the workflow model
lastModificationDate	**date**	The last modification date of the workflow model

Example:

```
// This script task logs information of the current workflow model.
uses core.log as log;
export procedure onNewContext()
begin
var message := ' Workflow identifier : ' |_ebx.workflow.id;
message |= '\n Workflow creation date : ' | _ebx.workflow.creationDate;
message |= '\n Workflow last modification date: ' | _ebx.workflow.lastModificationDate;
log.info(message);
end
```

191.4 Transaction

Introduction

In the context of script tasks, users may need to perform some actions on a dataspace under a transaction (i.e. update, modify, delete records, etc). Hence, if one or more actions fail, all other actions must back out leaving the state of the dataspace unchanged. The transaction ensures that the integrity state of data is never compromised.

Execute statement

The syntax to declare a transaction is the following:

```
execute transaction transaction_variable_name on dataspace_variable
begin
  // Transaction block.
...
  <statement_1>
  <statement_2>
...
end
```

- **transaction_variable_name** is the name of the transaction variable.
- **dataspace_variable** is the dataspace on which the transaction is executed. It is of type 'dataspaceType'.

Constants defined in encompassing scopes can be accessed by the transaction block, but non-constant variables cannot.

Return statements are not allowed in the transaction block.

Example:

```
references model "model.xsd" from module "ebx-test" as schema;
uses core.data as db;

export procedure executeScriptTask()
begin
    var myDataspace := db.lookupDataspace('Bdataspace');

    const id := 'xxx';

    // executes a transaction that creates an new  record in the table
    execute transaction tr on myDataspace
    begin
        var dataset := db.findDataset(tr.dataspace, 'dataset');
        var tableA:=db.findTable<schema.tableA>(dataset);
        var recordA:= db.recordOf(tr, tableA);
        recordA.id:=id; // the 'id' value is declared outside the transaction block. It must be const.
        db.saveRecord(recordA);
    end
end
```

Notes

Several transaction statements can be defined per script task. The name of transaction variable must be unique in the current scope otherwise an error will be generated at compile time.

```
export procedure executeScriptTask()
begin
    execute transaction tr_a on myDataspace
    begin
      <statement_a>;
    end

    execute transaction tr_b on myDataspace
    begin
      <statement_b>;
    end
```

```
end
```

The **nested** transactions are not **allowed** in the script task procedure. An error will be raised at compile time.

```
export procedure executeScriptTask()
begin
    execute transaction tr on myDataspace
    begin
      <statement_a>;
      // nested transaction is not allowed here !
      execute transaction tr on myDataspace
      begin
      <statement_b>;
       end
    end
end
```

CHAPTER 192

Table trigger

This chapter contains the following topics:

1. Introduction
2. Initial script
3. Predefined variables

192.1 Introduction

The trigger defines procedures that are implicitly and automatically executed when some operations are performed on the table, such as creations, updates, or deletions.

The trigger can be executed either just before the event (insert, delete, update) or just after.

The DSL (Domain Specific Language) is used to define a table trigger. For more details on DSL, see the Reference [p 1318] .

To create a table trigger, use the Data Model Assistant (DMA) [p 44] .

To edit and modify a script of the table trigger, use the EBX IDE that provides contextual code completion.

192.2 Initial script

Introduction

When a table trigger is first edited using the DMA [p 44] , EBX® provides an initial script with the correct definition of the exported procedure.

An initial script of a table trigger has following structure:

```
<unit usage statement 1>
<unit usage statement 2>
...
<unit usage statement N>

export procedure trigger_signature_name()
begin
  ...
  <statement 1>
  <statement 2>
  ...
end
```

The following script is a an example of **before create** trigger:

```
// this trigger calculates automatically the total and the percentage of students marks.
```

```
export procedure onBeforeCreate()
Begin
_ebx.record.total := _ebx.record.subj1 + _ebx.record.subj2 + _ebx.record.subj3;
_ebx.record.per :=(_ebx.record.total/60)*100;
end
```

The following script is a an example of **before modify** trigger:

```
// this trigger updates automatically the last modification date.
uses core.datetime as datetime;
export procedure onBeforeModify()
begin
_ebx.record.lastModificationDate := datetime.now();
end
```

Exported procedures

An exported procedure can be called directly by EBX®. Only one exported procedure is allowed per script in a table trigger.

The signatures of table trigger procedures are the following:

- **onNewContext()** : is called when a new context of record is created.
- **onBeforeCreate()** : is called before the creation of a record.
- **onAfterCreate()** : is called after the creation of a record.
- **onBeforeModify()** : is called before the modification of a record.
- **onAfterModify()** : is called after the modification of a record.
- **onBeforeDelete()** : is called before deleting a record.
- **onAfterDelete()** : is called after deleting a record.
- **onBeforeTransactionCancel()** : is called before a cancel.
- **onBeforeTransactionCommit()** : is called before a commit.

Note: The exported procedure signature (name) depends on the trigger's type and should not be changed.

192.3 Predefined variables

Introduction

Predefined variables allow access to the context of a table trigger. The following contextual variables are specific to the table trigger. For more details on common predefined variables, see the Reference [p 1318] .

Session

The predefined variable **_ebx.session** provides access to information on the current session. This contextual variable is constant and of an immutable complex type.

It has the following fields:

Name	Type	Description
userId	**string**	The identifier of the current user
userEmail	**string**	The email of the current user
locale	**string**	The locale of the current session

Example:

```
// this triggers logs information of the current session.
uses core.log as log;
export procedure onNewContext()
begin
var message := ' Session locale : ' | _ebx.session.locale;
message |= '\n User id : ' | _ebx.session.userId;
message |= '\n User email: ' | _ebx.session.userEmail;
log.info(message);
end
```

Record

The predefined variable **_ebx.record** allows read/write access to the current record to be created, updated or deleted. Its fields are defined by the current EBX® schema.

Example:

```
// This trigger adds the full name and the modification date of a new created person.
uses core.datetime as datetime;
export procedure onBeforeCreate()
begin
  _ebx.record.inscriptionDate := datetime.now();
  _ebx.record.fullName := _ebx.record.firstName | ' ' | _ebx.record.lastName;
end
```

NextRecord

The predefined variable **_ebx.nextRecord** allows read-write access to the next record to be updated. It is available only for **onBeforeModify** triggers. It is defined by the current EBX® schema.

Example:

```
uses core.datetime as datetime;
export procedure onBeforeModify()
Begin

   if (_ebx.nextRecord.phone <> _ebx.record.phone) then
   begin
   _ebx.nextRecord.phoneChangeDate := datetime.now();
   _ebx.nextRecord.description := 'old phone number is  ' | _ebx.record.phone | '
   and New Phone number is ' | _ebx.nextRecord.phone ;
   end
 end
```

PreviousRecord

The predefined variable **_ebx.previousRecord** allows read-only access to the previous record after an update. It is available only for **OnAfterModify** triggers. It is defined by the current EBX® schema.

Example:

```
// This trigger logs all data modifications of the table sales.
uses core.log as log;
uses core.datetime as datetime;
export procedure OnAfterModify( )
```

```
begin
var message := ' Sales id : ' | _ebx.record.sales_id;
message |= '\n Previous amount : ' | _ebx.previousRecord.amount;
message |= '\n Current amount : ' |  _ebx.record.amount;
message |= '\n Updated on : ' | datetime.now( );
log.info(message);
end
```

Transaction

The predefined variable **_ebx.transaction** provides a read-only access to the current transaction of the trigger. It is available only in the context of a table trigger and dataset trigger. This variable is constant and of an immutable complex type. The transaction is needed to perform some operations on the dataspace (update, modify, delete records, etc). It ensure that the integrity state of data is never compromised when some actions fail.

Example:

```
uses core.data as d;
// The trigger creates a new record in a table.
export procedure onAfterCreate()
begin
var tableA := d.findTable<.tableName>(_ebx.dataset)
var recordA:= d.recordOf(_ebx.transaction, tableA);
// Update the record.
...
d.saveRecord(recordA);
end
```

CHAPTER 193

Dataset trigger

This chapter contains the following topics:

1. Introduction
2. Initial script
3. Predefined variables

193.1 Introduction

The trigger defines procedures that are implicitly and automatically executed when some operations are performed on the dataset. The trigger can be executed either just before or just after the event (creations, updates, or duplications) .

The DSL (Domain Specific Language) is used to define a dataset trigger. For more details on DSL, see the Reference [p 1318] .

To create a dataset trigger, use the Data Model Assistant (DMA) [p 44] .

To edit, modify and publish a script of the dataset trigger, use the EBX® IDE that provides contextual code completion.

193.2 Initial script

Introduction

When a dataset trigger is first edited in EBX® IDE, an initial script with the correct definition of the exported procedure is provided.

An initial script of a dataset trigger has following structure:

```
<unit usage statement 1>
<unit usage statement 2>
...
<unit usage statement N>

export procedure trigger_signature_name()
begin
  ...
  <statement 1>
  <statement 2>
  ...
end
```

The following script is a an example of **before modify** trigger:

```
// this trigger updates root fields before modifying the dataset.
export procedure onBeforeModify()
```

```
begin
_ebx.root.fieldA:='xx';
_ebx.root.fieldB:='yy';
end
```

Exported procedures

An exported procedure can be called directly by EBX®. Only one exported procedure is allowed per script in a dataset trigger.

The signatures of dataset trigger procedures are the following:

- **onBeforeCreate()** : is called before the creation of a dataset.
- **onAfterCreate()** : is called after the creation of a dataset.
- **onBeforeModify()** : is called before the modification of a dataset.
- **onAfterModify()** : is called after the modification of a dataset.
- **onBeforeDelete()** : is called before deleting a dataset.
- **onAfterDelete()** : is called after deleting a dataset.
- **onAfterDuplicate()** : is called after duplicating a dataset.

Note: The exported procedure signature (name) depends on the trigger's type and should not be changed.

193.3 Predefined variables

Introduction

Predefined variables allow access to the context of a dataset trigger. The following contextual variables are specific to the dataset trigger. For more details on common predefined variables, see the Reference [p 1318] .

Session

The predefined variable **_ebx.session** provides access to information on the current session. This contextual variable is constant and of an immutable complex type. This variable is also available in the context of a table trigger.

It has the following fields:

Name	Type	Description
userId	**string**	The identifier of the current user
userEmail	**string**	The email of the current user
locale	**string**	The locale of the current session

Example:

```
// this triggers logs user identifier of the current session.
uses core.log as log;
export procedure onBeforeCreate()
begin
log.info(' User id : ' | _ebx.session.userId);
end
```

originalDataset

The predefined variable **_ebx.originalDataset** provides access to information of the original dataset. It is available only for the trigger after duplicate.

This variable has the following fields:

Name	Type	Description
name	**string**	The name of the original dataset
dataspace	**dataspace**	The dataspace of the dataset

Example:

```
// this triggers logs the original and the current of dataset after the duplicate operation.
uses core.log as log;
export procedure onAfterDuplicate()
begin
var message := \n Original dataset : ' | _ebx.originalDataset.name;
message |= '\n Current dataset : ' | _ebx.dataset.name;
log.info(message);
end
```

Transaction

The transaction is needed to perform operations on the dataset such as creation, delete and duplicate. The predefined variable **_ebx.transaction** provides a read-only access to the current transaction of the dataset trigger. It is available only for dataset trigger **afterCreate** , **afterModify** , **afterDelete** and **afterDuplicate** . This variable is constant and of an immutable complex type.

Example:

The following script is a an example of **after duplicate** trigger:

```
uses core.data as d;
uses core.complex as c;
// The trigger deletes a record from a table after duplicating the dataset.
export procedure onAfterDuplicate()
begin
var table := d.findTable<.table>(_ebx.dataset);

// Creates the primary key.
var pk := c.primaryKeyOf<.table>();
pk.id := '1';

// Deletes the record using the primary key.
d.deleteRecordByPrimaryKey(_ebx.transaction, table, pk);

end
```

The following script is a an example of **after create** trigger:

```
// this trigger updates fields values after creating the dataset.
references model "sample-model" as data;
uses core.data as d;
export procedure onAfterCreate()
begin
var root:=d.lookupDatasetRootForUpdate<data>(_ebx.transaction,_ebx.dataset);
root.fieldA:='A';
root.fieldB:='B';
d.saveDatasetRoot(root);
end
```

CHAPTER **194**

Scripted REST service

This chapter contains the following topics:

1. Introduction
2. Endpoint mapping and operationId
3. Exported functions
4. Parameters: parsing and validation
5. Producing responses
6. Schemas, types, and OpenAPI
7. Supported OpenAPI subset
8. Authentication and session
9. Transactions and data access
10. Error handling
11. See also

194.1 Introduction

A scripted REST service defines a REST endpoint in EBX® and its related script. You create and maintain these services in the Script IDE. After you compile and publish the service from the IDE, it becomes available to REST clients.

Each operation in a scripted REST service is implemented as an EBX® Script exported function that runs when the operation is called via an HTTP request. Exported functions return a **rest_response** value. The **core.rest** unit provides utilities to construct responses. Requests and responses only use JSON content.

194.2 Endpoint mapping and operationId

A scripted REST service defines one or more endpoints. Each endpoint has a URL path and one or more HTTP methods. For each method, you assign an **Operation id** which is interpreted as an OpenAPI **operationId** . This value identifies the exported EBX® Script function that runs when a client calls the endpoint with that method. The **Operation id** value must match the name of corresponding exported function exactly.

The URL path can include path parameters (appearing in braces) and query parameters. Parameters are used by the related function as follows:

- Parameter types are inferred from the OpenAPI schema. See Schemas, types, and OpenAPI [p 1362] for details.
- Each path parameter becomes a function parameter.
  - The parameter name matches the path parameter name.
  - The parameter type is inferred from the OpenAPI schema.
  - The order of path parameters in the function matches their order in the path template.
- Query parameters defined for the operation also become function parameters, following the path parameters in the order they are declared in the OpenAPI document.

Examples (names reflect **operationId**):

OpenAPI (excerpt):

```
{
  "openapi": "3.0.1",
  "paths": {
    "/users": {
      "get":    { "operationId": "getUsers",    "responses": { "200": { "content": { "application/json":
 { "schema": { "type": "array", "items": { "$ref": "#/components/schemas/User" } } } } } } },
      "post":   { "operationId": "createUser",  "requestBody": { "required": true, "content": { "application/
json": { "schema": { "$ref": "#/components/schemas/User" } } } },
                  "responses": { "201": { "content": { "application/json": { "schema": { "$ref": "#/components/
schemas/User" } } } } } }
    },
    "/users/{id}": {
      "parameters": [ { "name": "id", "in": "path", "required": true, "schema": { "type": "number" } } ],
      "get":    { "operationId": "getUser",     "responses": { "200": { "content": { "application/json":
 { "schema": { "$ref": "#/components/schemas/User" } } } } } },
      "delete": { "operationId": "deleteUser",  "responses": { "204": {} } }
    },
    "/display/{p1}/{p2}": {
      "parameters": [
        { "name": "p1", "in": "path", "required": true, "schema": { "type": "string" } },
        { "name": "p2", "in": "path", "required": true, "schema": { "type": "number" } }
      ],
      "get": { "operationId": "displayParameters",
        "parameters": [
          { "name": "message",  "in": "query", "required": true,  "schema": { "type": "string" } },
          { "name": "count",    "in": "query", "required": false, "schema": { "type": "number" } },
          { "name": "datetime", "in": "query", "required": false, "schema": { "type": "string", "format": "date-
time" } },
          { "name": "date",     "in": "query", "required": false, "schema": { "type": "string", "format":
 "date" } },
          { "name": "time",     "in": "query", "required": false, "schema": { "type": "string", "format":
 "time" } }
        ],
        "responses": { "200": { "content": { "application/json": { "schema": { "$ref": "#/components/schemas/
Query" } } } } }
      }
    }
  },
  "components": {
    "schemas": {
      "User": { "type": "object", "properties": { "id": { "type": "number" }, "firstName": { "type": "string" },
 "lastName": { "type": "string" } } },
      "Query": { "type": "object", "properties": { "p1": { "type": "string" }, "p2": { "type": "number" },
 "message": { "type": "string" }, "count": { "type": "number" }, "datetime": { "type": "string", "format": "date-
time" }, "date": { "type": "string", "format": "date" }, "time": { "type": "string", "format": "time" } } },
      "Error": { "type": "object", "properties": { "message": { "type": "string" } } }
    }
  }
}
```

Equivalent exported methods:

```
uses core.rest as rest;
uses core.list as list;

export function getUsers(): rest_response
begin
  // Return a list of users
```

```
  return rest.responseOf(200, list.of<User>());
end

export function createUser(const user: User): rest_response
begin
  return rest.responseOf(201, user);
end

export function getUser(id: decimal): rest_response
begin
  // Return typed object or a typed error
  var err := instanceOf<Error>();
  err.message := 'Not found';
  return rest.responseOf(404, err);
end

export function deleteUser(const id: decimal): rest_response
begin
  return rest.emptyResponse(204);
end

export function displayParameters(
  p1: string,
  p2: decimal,
  message: string,
  count: decimal,
  datetime: datetime,
  date: date,
  time: time): rest_response
begin
  var q := instanceOf<Query>();
  q.p1 := p1;
  q.p2 := p2;
  q.message := message;
  q.count := count;
  q.datetime := datetime;
  q.date := date;
  q.time := time;
  return rest.responseOf(200, q);
end
```

194.3 **Exported functions**

In the context of a scripted REST service, exported functions are REST operation handlers. Each exported function must return a **rest_response** and can accept parameters for path, query, or body as defined by the service contract.

- Declare path and query parameters using scalar types such as **string** , **int** , **decimal** , **boolean** , **date** , **time** , or **datetime** .
- The request body, when present, is typically declared as a typed object generated from the **components.schemas** API, or as **json_value** for unstructured JSON.
- The function name must match the OpenAPI **operationId** .
- Parameter order: path parameters first (path order), then query parameters (OpenAPI document order).

For a full example, see the combined OpenAPI + exported methods above.

194.4 **Parameters: parsing and validation**

Scripted REST services handle parameter parsing and validation automatically:

- Path and query parameters are parsed according to their declared types.
- Invalid numbers and dates return 400 with a standard error payload, for example: "Path parameter 'year' is not valid." or "Query parameter 'date' is not valid." as applicable.
- For boolean path parameters, non-"true" values are coerced to false rather than rejected.

- Empty path segments are rejected with 400 by the container before reaching the script.

194.5 Producing responses

To produce responses from exported functions, use **core.rest** utilities:

- **rest.responseOf(status: int, content: any)** creates a JSON response with an optional status code.
- If **status** is null, 200 is assumed.
- **content** can be a JSON object type, any **json_value** , or lists of these.
- **rest.emptyResponse(status: int)** creates a response without a body.
- If **status** is null, 204 is assumed.

194.6 Schemas, types, and OpenAPI

OpenAPI 3 describes operations and data shapes. In scripted REST services:

- Each **components.schemas** entry becomes an EBX® Script type with the same name (for example: **User** , **Error** , **Meeting** , **Query**). Fields map 1:1 to schema properties.
- Scalars map as follows:
  - integer (format: int32) → **int**
  - number → **decimal**
  - boolean → **boolean**
  - string → **string**
  - string (format: date-time) → **datetime**
  - string (format: date) → **date**
  - string (format: time) → **time**
- Responses/bodies can be typed objects, lists of typed objects, or generic JSON via **json_value** .

Examples:

```
// Typed body
export function updateUser(const user: User): rest_response
begin
  return rest.responseOf(200, user);
end

// List of strings
uses core.list as list;
uses core.json as json;
export function getTags(): rest_response
begin
  var tags := list.of('one', 'two');
  return rest.responseOf(200, json.copyOfStrings(tags));
end
```

When using OpenAPI-driven templates, object types and field names are derived from the component schemas. Keep the returned structure aligned with the declared types for clear compile-time validation.

194.7 Supported OpenAPI subset

Scripted REST services support a subset of OpenAPI 3.0 features:

- Paths: keys are validated as path templates. Keep them free of any base root (the script name defines the base path).
- Methods: **GET** , **POST** , **PUT** , **DELETE** supported. Response keys must be integer HTTP codes in a valid range.
- Security: **security: []** allows anonymous access. Any other **security** object is not supported.
- Parameters:
  - At path level: only **in: path** are processed; names must exist in the path template.
  - At method level: only **in: query** are processed; **path** must be declared at path level.
  - Each parameter must declare a **schema** with a supported primitive type (**string** , **integer** , **number** , **boolean**) and optional format (**date** , **time** , **date-time**). Parameters cannot be arrays or objects and cannot use **$ref** .
- Request bodies: only **application/json** is considered. The schema may be a **$ref** to a component schema or an inline scalar/array/object as applicable.
- Responses: only **application/json** is considered. Each status can optionally declare a JSON schema (scalar, array, or **$ref**).
- Components schemas: only **object** types are supported with properties; additionalProperties must be absent or false. **$ref** targets must be under **#/components/schemas** .

194.8 Authentication and session

Configure scripted REST services to allow anonymous access or require authentication. Implement your handlers so they do not assume a user context unless required by the endpoint design, and return appropriate status codes when access should be restricted.

- At runtime, **_ebx.anonymous** indicates whether the request is anonymous.
- For anonymous-friendly endpoints, the OpenAPI operation typically sets **security: []** .
- Use **_ebx.session.userId** when a user session exists.

Example:

```
uses core.rest as rest;
uses core.list as list;
uses core.json as json;

export function getSessionDetails(): rest_response
begin
  var userId := '';
  if _ebx.anonymous then
    userId := 'anonymous';
  else
    userId := _ebx.session.userId;

  var content := list.of(json.ofString(userId));
  return rest.responseOf(200, content);
end
```

194.9 Transactions and data access

Use regular EBX® Script transaction and data APIs to read and modify EBX® data. The following example includes a typical pattern with a transaction block and a **ref** to carry the response across the block:

```
uses core.rest as rest;
uses core.data as db;
```

```
uses core.complex as cp;

// Model reference for dataset types used in generics below (replace with your own module/name)
references model "directory" from module "my-module" as dir;

export function updateUser(const user: User): rest_response
begin
  ref response: rest_response;

  execute transaction tr on db.findDataspace('rest-test') as system
  begin
    const directory := db.findDataset(tr.dataspace, 'directory');
    const table := db.findTable<dir.User>(directory);

    var pk := cp.primaryKeyOf<dir.User>();
    pk.id := user.id;

    var record := db.lookupRecordForUpdate(tr, table, pk);
    if isNull(record) then
    begin
      var error := instanceOf<Error>();
      error.message := 'User "' | user.id | '" not found';
      response := rest.responseOf(404, error);
    end
    else
    begin
      record.firstName := user.firstName;
      record.lastName := user.lastName;
      db.saveRecord(record);
      response := rest.responseOf(200, user);
    end
  end

  return response;
end
```

When the request should run with elevated privileges, use **execute transaction … as system** . Otherwise, omit **as system** to use the authenticated user context.

194.10 Error handling

Scripted REST services handle errors as follows:

- Invalid path or query parameters are rejected automatically with **400** and a standard error payload.
- For business errors (not found, already exists, validation), return typed error content with an appropriate status code (for example, **404** , **400**).

Example:

```
var err := instanceOf<Error>();
err.message := 'User not found';
return rest.responseOf(404, err);
```

194.11 See also

- REST service [p 1365] (calling external services from DSL)
- Reference [p 1318] (language reference and types)
- REST Service tutorial [p 1569] (detailed walkthrough of creating a scripted REST service)

CHAPTER 195

REST service

This chapter contains the following topics:

1. Introduction
2. Configuration and authentication
3. Calling a REST service

195.1 Introduction

EBX® Script can call external REST services using the **HTTP API** . This API provides methods to:

- Create HTTP requests.
- Add parameters to headers.
- Read responses (message body, HTTP code, etc.)

Use the following HTTP methods to perform operations on server resources:

- **GET** : retrieves a resource or a collection from the server.
- **POST** : sends data to create a new resource on the server.
- **PUT** : sends data to update an existing resource on the server.
- **DELETE** : deletes a resource on the server.

REST services can be called from table triggers [p 1351] and script tasks [p 1345] by using the EBX® Script IDE.

195.2 Configuration and authentication

Configure each REST service called in a script in the following table [p 715] : *Administration* > *Integration* > *External services* > *Configuration* where the following properties are required:

- **Configuration id** : The unique identifier for the REST configuration. It is required to create an HTTP request in the API.
- **Base URL** : The URL of the server to call. Use HTTPS to encrypt the communication between the client and the server. The URL has the following format:

```
http[s]://<host>[:<port>]/{path}{?<queryParameters>}

where

- host and port: defines the server address where the API resides.
- path: is the path of the specific resource within the API.
```

```
- queryParameters: are optional parameters used to filter or refine the resource.
```

- **Usage** : Defines how the service is used: AI Assistant and/or the REST service.
- **Authentication type** : Specifies the authentication type used to access resources. The following types are supported:

Type	Description
Basic	is a simple and fast method of HTTP authentication that requires a valid username and password. Credentials are encoded in base 64 and included in the HTTP request header. Note that there is no need to add the authentication code in the header of the request. It is automatically included before each access to the server.
EBX	enables access to the EBX Built-in RESTful [p 1125] service. A valid username and password must be provided to obtain an authentication token. The token has a timeout period. A new token is requested and included in the HTTP request header for each access to the EBX® server. Note that there is no need to add the EBX token in the header from the script. It is automatically included before each access to the EBX® server.
Custom	enables custom authentication by providing valid key names and key values. The values of the keys can be secret and are encrypted based on the keystore. This custom type can be used to call a service with authentication like API keys, tokens, or OAuth.
None	no authentication is needed.

195.3 Calling a REST service

To call a REST service from an EBX® script:

- Navigate to *Administration* > *Integration* > *External services* > *Configuration* and configure the server: set a configuration ID, URL, authentication type, and credentials.
- Access table triggers [p 1351] or script tasks [p 1345] within the EBX® Script IDE.
- Call the REST service using the specified configuration ID.

Create a request

A new HTTP request can be created using the core.http [p 1386] API. It provides methods to configure the request, add paths, parameters and header fields.

The following examples show some scripts calling a REST service.

Example : create a request to get all products of a specific category

GET https://api.example.com/v1/products/?sort=desc&category=jewelery

The following configuration is used:

- configuration id: ebx-1
- URL: https://api.example.com
- Usage: Rest

- authentication type: basic
- credentials: username, password

```
uses core.http as client;
...
procedure getJeweleryProducts()
begin

  // create a request with basic authentication
  var request := client.createRequest('ebx-1');

  // include content type in the header
  client.setHeader(request,'Content-Type','application/json');

  // add paths
  client.addPath(request,'v1');
  client.addPath(request,'products');

  // add query parameters
  client.addParameter(request,'category','jewelery');
  client.addParameter(request,'sort','desc');
  ...
  // sends a request to get products of a specific category
  ...
end
```

Example : create a request to get all products with a limited result.

GET https://api.example.com/v1/products?limit=15

The following configuration is used:

- configuration id: ebx-2
- URL: https://api.example.com
- Usage: Rest
- authentication type: custom
- key name: api_key
- key value: access_key

```
uses core.http as client;
...
procedure getProducts()
begin
  // create a request with custom authentication
  var request := client.createRequest('ebx-2');
  client.addPath(request,'v1');
  client.addPath(request,'products');
  client.addParameter(request,'limit','15');

  // get api_key value saved during the configuration of the REST service
  var accessKey := client.getConfigValue(request,'api_key');
  if not isNull(accessKey) then
  begin
    // include the authorization api_key in the header
    client.setHeader(request,'Authorization','api_key '| accessKey);
    ...
    // sends a request to get all products
    ...
  end
end
```

Example : create a request to call the EBX Built-in RESTful [p 1125] service and update the user email.

PUT https://ebx-example.io/test/ebx-dataservices/rest/data/v1/Bdataspace/dataset/root/user/ID6766

The following configuration is used:

- configuration id: ebx-3
- URL: https://ebx-example.io/test/ebx-dataservices
- Usage: Rest

- authentication type: ebx
- credentials: username, password

```
uses core.http as client;
...
procedure updateUserEmail()
begin

var request:= client.createRequest('ebx-3');

// add paths
client.addPath(request,'data/v1/Bdataspace/dataset/root/user/ID6766');

// include the content type in the header
client.setHeader(request,'Content-Type','application/json');

// sends a put request to update the user email
...
end
```

Example : create a request to get information about a city

GET https://geo.api.gouv.fr/communes?codePostal=92160&fields=nom,departement,region

The following configuration is used:

- configuration id: ebx-4
- URL: https://geo.api.gouv.fr/communes
- Usage: Rest
- authentication type: none

```
uses core.http as client;
...
procedure getCityInfo()
begin

   var request:= client.createRequest('ebx-4);

   // add query parameters
   client.addParameter(request,'codePostal','92160');
   client.addParameter(request,'fields','nom,departement,region');

   // get the url of the request.
   var message:='Request URL : '|client.getRequestUrl(request);

   // sends a request to get information about a city specified by its zipcode 92160
   ...
end
```

Send a request

The core.http [p 1386] API provides several HTTP methods (GET, PUT, POST, DELETE) to call a REST service. The following examples show some scripts using HTTP method to perform operations on the products.

GET

Example : retrieve a single product from the server

GET https://api.example.com/v1/products/P87RZ63

The following script sends a GET request to retrieve a single product by its identifier.

```
uses core.http as client;

procedure getSingleProduct()
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'v1/products/P87RZ63');

  // get response
```

```
  var response := client.get(request);
  ...
end
```

POST

Example : create a new product on the server

POST https://api.example.com/v1/products

The following script sends a POST request to add new product.

```
uses core.http as client;

procedure addProduct()
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'v1/products');
  client.setHeader(request,'Content-Type','application/json');
  var body:='{
                "id": "L32LQ",
                "title": "TV LED Sony",
                "price": 1543,
                "image": "https://tv.pravatar.sony",
                "category": "electronic"
            }';

 // sends a POST request
  var response := client.post(request,body);
  ...
end
```

PUT

Example : update an existing product on the server.

PUT https://api.example.com/v1/products/L32LQ

The following script sends a PUT request to update a specific product.

```
uses core.http as client;
...
procedure updateProduct()
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'v1/products/L32LQ');

  var body:='{
                "price": 789
            }';

 // sends a PUT request
  var response := client.put(request,body);
  ...
end
```

DELETE

Example : deletes an existing product on the server.

DELETE https://api.example.com/v1/products/L32LQ

The following script sends a DELETE request to delete a specific product.

```
uses core.http as client;
...
procedure deleteProduct()
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'v1/products/L32LQ');
  var response := client.delete(request);
  ...
end
```

Read a response

Use the core.http [p 1386] API to read status codes, headers, and bodies of the response. Use json.http [p 1386] to parse the response body and manage JSON data.

Example : read the body response of the request to get information about a specific product.

```
uses core.http as client;
uses core.json as json;
uses core.log as log;
...
procedure getProductInfo()
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'v1/products/P87RZ63');

  // send a get request
  var response := client.get(request);

  // get status code of the response
  var status:=client.getStatusCode(response);
  if (status = 200) then
  begin

     if (client.hasBody(response)) then
     begin
       // get the first content encoding from the response header
       var encoding := client.getHeaderFirstValue(response,'content-encoding');

       // get the list of access allowed methods from the response header
       var allowedMethod := client.getHeaderValues(response,'access-control-allow-methods');

       // get the body of the response as object
       var body:=client.getJsonBody(response).asObject;

       // retrieve information about the product
       log.info('id :' | json.getString(body ,'id'));
       log.info('title :' | json.getString(body ,'title'));
       log.info('price :' | json.getString(body ,'price'));
     end
    ...
  end
end
```

Handle response errors

When an error occurs, check the status code and the additional information reported in the response body. The core.http [p 1386] API provides methods to fetch the request URL, to get the status code, and to read error messages in the response body.

Example : log response errors when sending a request to the EBX Built-in RESTful [p 1125] service to update the user email.

```
uses core.http as client;
uses core.json as json;
uses core.log as log;
...
procedure updateEmail()
begin

   var request:= client.createRequest('ebx-3');
   client.addPath(request,'data/v1/Bdataspace/dataset/root/user/ID66Z7');

   // set a new timeout for the request
   client.setTimeout(request,800);

   var json:='{
                   "content": {

                     "email": {
                       "content": "loana-provost@gmail.com"
                     }
                   }
              }';

    // update user email
```

```
    var response := client.put(request,json);
    var status:=client.getStatusCode(response);

   // read the response
   if (status>=200 and status <=300) then
    begin
       log.info('The request was successful '|status);
      if (client.hasBody(response)) then
      begin
        log.info('Response body: ' | client.getStringBody(response));
      end
    end
   else
      begin
        log.info('Error when sending the request');
        // log the url of the request.
        log.info('Request URL : '|client.getRequestUrl(request));

        // log the url of the response.
        log.info('Response URL : '|client.getResponseUrl(response));

        var body:=client.getJsonBody(response).asObject;
        // log error status code
        log.info('Code : ' | json.getDecimal(body,'code'));

        // log error message
        var errors:=json.getArray(body,'errors');
        for error in errors do
         begin
         log.info('Message : ' |  json.getString(error.asObject,'message'));
         end
      end
end
```

CHAPTER **196**

Role mapper

This chapter contains the following topics:

1. Introduction
2. Configuration
3. Initial script
4. Function signature
5. Available functions
6. Examples
7. Troubleshooting

196.1 Introduction

Role mapper scripts provide a flexible approach to mapping external roles from identity providers to internal EBX® roles as part of the synchronized directory [p 706] functionality.

To create, modify and publish a role mapper script, use the EBX® Script IDE.

196.2 Configuration

Role mapper scripts are configured in the synchronized directory section of **ebx.properties** :

```
# Enable synchronized directory
ebx.directory.synchronized.source=LDAP
...

# Set role mapper to use scripts
ebx.directory.synchronized.role.mapper=SCRIPT
```

196.3 Initial script

When a role mapper script is first edited using the EBX® Script IDE, EBX® provides an initial script with the correct definition of the exported function.

An initial role mapper script has the following structure:

```
<unit usage statement 1>
<unit usage statement 2>
...
<unit usage statement N>

export function mapRoles(userAttributes: unmodifiable dictionary<set<string>>,
                         externalRoles: unmodifiable set<string>): set<profile>
```

```
begin
  ...
  <statement 1>
  <statement 2>
  ...
end
```

The following example is a basic role mapper script:

```
uses core.set as set;
uses core.profile as profile;

export function mapRoles(userAttributes: unmodifiable dictionary<set<string>>,
                         externalRoles: unmodifiable set<string>): set<profile>
begin
    var result := set.of<profile>();

    // Map external admin role to built-in administrator
    if set.contains(externalRoles, 'admin') then
    begin
        set.add(result, profile.ofAdministrator());
    end

    return result;
end
```

196.4 Function signature

The **mapRoles** function must have the following signature:

```
export function mapRoles(userAttributes: unmodifiable dictionary<set<string>>,
                         externalRoles: unmodifiable set<string>): set<profile>
```

Parameters

- **userAttributes** : a dictionary containing user attributes from the identity provider. Each key is an attribute name and each value is a set of strings representing the attribute values.

  Example: **{'department': {'IT', 'Engineering'}, 'location': {'Paris'}, 'title': {'Senior Developer'}}**

- **externalRoles** : a set of external roles assigned to the user by the identity provider.

  Example: **{'admin', 'developer', 'manager'}**

Return value

The function must return a set of **profile** objects representing the internal roles to be assigned to the user.

Note: If the function returns an empty set, no roles will be assigned to the user.

196.5 Available functions

The following profile functions are available using the core.profile [p 1503] unit:

- **profile.ofAdministrator()** : returns the built-in administrator profile.
- **profile.ofReadOnly()** : returns the built-in read-only profile.
- **profile.ofOwner()** : returns the built-in owner profile.
- **profile.ofEveryone()** : returns the built-in everyone profile.
- **profile.forSpecificRole(roleName)** : returns a specific role profile with the given name.

196.6 Examples

Basic role mapping

This example shows how to map external roles directly to internal EBX® profiles.

```
uses core.set as set;
uses core.profile as profile;

export function mapRoles(userAttributes: unmodifiable dictionary<set < string >>,
externalRoles: unmodifiable set<string>): set<profile>
begin
 var result := set.of<profile>();

 // Map external admin role to built-in administrator
 if set.contains(externalRoles, 'external_admin') then
  set.add(result, profile.ofAdministrator());

 // Map external read role to built-in read-only
 if set.contains(externalRoles, 'external_reader') then
  set.add(result, profile.ofReadOnly());

 return result;
end
```

Attribute-based role mapping

This example demonstrates how to use user attributes (like department) to determine role assignments.

```
uses core.dictionary as dict;
uses core.set as set;
uses core.profile as profile;

export function mapRoles(userAttributes: unmodifiable dictionary<set < string >>, externalRoles: unmodifiable
 set<string>): set<profile>
begin
 var result := set.of<profile>();

 // Get user department from attributes
 var departments := dict.get(userAttributes, 'department');

 if isNull(departments) then
  return result;

 if set.contains(departments, 'IT') then
  set.add(result, profile.ofAdministrator());

 if set.contains(departments, 'HR') then
  set.add(result, profile.forSpecificRole('HR_Manager'));

 return result;
end
```

196.7 Troubleshooting

Script not executing:

- Verify that **ebx.directory.synchronized.role.mapper=SCRIPT** is set in **ebx.properties**
- Check that the script is published and active in the EBX® Script IDE
- Review script logs for runtime errors

CHAPTER 197

Constraint validation

This chapter contains the following topics:

1. Introduction
2. Initial script
3. Predefined variables
4. Constraint types
5. Constraint dependencies

197.1 Introduction

EBX® Script lets you define validation constraints on data models. Constraints are rules on a data model that ensure users enter only valid, accurate, and logically consistent data. You can define constraints at different levels: table, record, table field and dataset field.

When a constraint detects invalid data, it can add a custom message to the validation report. You choose the message text and severity level.

You create constraints in the Advanced controls [p 49] view of the Data Model Assistant (DMA) [p 44] . EBX® generates an initial script that you can edit in the EBX® Script IDE, which provides contextual code completion.

Use the EBX® script API core.message [p 1386] to manage localized user messages in constraints. This API provides several methods to create and add user-friendly messages to the validation report.

197.2 Initial script

When you create a validation constraint using the Advanced controls [p 49] in the DMA [p 44] , EBX® generates an initial script. The initial script contains the correct structure and the necessary exported procedures for the constraint type.

Structure

A constraint's initial script has the following structure:

```
<unit usage statement 1>
<unit usage statement 2>
...
<unit usage statement N>

export procedure constraint_signature_name()
begin
  ...
```

```
  <statement 1>
  <statement 2>
  ...
end

export procedure setup()
begin
  ...
  <statement 1>
  ...
end
```

The following script example shows a constraint on an employee in a department. It checks whether the employee number is less than 100. If it is, a warning message is added to the context of the validation report.

```
export procedure checkRecord(record : .Department)
begin

if (record.Number < 100i) then
 begin
  var message:='The employee number of the department '|record.Name |' is under 100';
  msg.addWarning(message);
 end
end

export procedure setup()
begin
    // Initialization can be added here
  ...
end
```

Exported procedures

EBX® calls exported procedures directly when evaluating a constraint. Each constraint script can declare only the exported procedures that are relevant to its type (two or three procedures, depending on the constraint type). The exported procedure **setup( )** is mandatory. EBX® calls it to initialize the constraint context. Use it to perform initialization tasks, such as declaring the constraint's dependencies.

EBX® includes the following constraint types:

- **Constraint on table** : defines a table-level check to ensure that the entire table satisfy a specific condition. It also allows definition of a record-level check to ensure that the record satisfies a specific condition.
- **Constraint on field** : defines a field-level check to ensure that the field satisfies a specific condition.
- **Constraint on null** : defines a field-level check to ensure that the field is mandatory and can not be null.

The signatures of constraint procedures are the following:

- **checkTable( )** and **checkRecord( )** : are called when the constraint type is *Constraint on table* .
- **check( )** : is called when the constraint type is *Constraint on field* .
- **checkNull( )** : is called when the constraint type is *Constraint on null* .
- **setup( )** : is called to initialize the constraint. This procedure is mandatory for all constraint types.

Note: The exported procedure signature (name) depends on the constraint's type and should not be changed. The exported procedure **setup( )** is mandatory and should not be changed.

197.3 Predefined variables

Predefined variables give you access to information that is available in the constraint context. The following predefined variables can be used depending on the constraint type.

Dataset

The predefined variable **_ebx.dataset** provides access to information on the current dataset. It is a common predefined variable for all EBX® Script usages (trigger, function field, etc.). For more details, see the Reference [p 1318] section. It is also available for all constraint types.

Example:

```
uses core.message as msg;
// This constraint adds information about the name of the current dataset.
export procedure checkTable()
begin

msg.adInfo('The dataset name is: ' | _ebx.dataset.name);

end
export procedure setup()
begin
  ...
end
```

Dataspace

The predefined variable **_ebx.dataspace** allows read access to the current dataspace. It is common predefined variable for all script usages (trigger, function field, …). For more details, see the Reference [p 1318] . It is also available for all constraint types.

Example:

```
uses core.message as msg;
// This constraint adds information about the name of the current dataspace.
export procedure check()
begin

msg.addInfo('The dataspace name is: ' | _ebx.dataspace.name);

end
export procedure setup()
begin
  ...
end
```

Root

The predefined variable **_ebx.root** provides read-only access to the current root of the dataset. It is available for all constraint types. The fields under the root are defined by the current EBX® schema.

Example:

```
uses core.message as msg;
// This constraint adds information about the reference code.
export procedure check()
begin

msg.addInfo('Reference code '|_ebx.root.code);

end
export procedure setup()
begin
  ...
end
```

197.4 Constraint types

A validation constraint defines specific rules to ensure data consistency within a data model. It can be applied at different levels of the data model (table, record, or field). When a validation constraint is violated, a customized error message with a user-defined severity level can be added to the validation report. Below are examples of validation constraints at different levels.

Constraint on table

Table-level check

This constraint type checks the entire table and adds messages when errors are detected. The whole table is checked at once. You can select specific records to be checked using SQL queries and XPath expressions.

Example:

```
uses core.message as msg;

export procedure checkTable()
begin

// This constraint checks the customers with an empty address and adds an error message.
// A localized user message ('The address is mandatory.') is created from a resource bundle.
var userMessage:= msg.ofBundleError('bundle1/path1','address.mandatory');

// Adds an error message for each customer with an empty address.
msg.addXPathMessage('osd:is-empty(Address)',userMessage);
end

export procedure checkRecord(record :.Employee)
begin
end

export procedure setup()
begin
  ...
end
```

Record-level check

This constraint supports record-level checks, in addition to the table-level check. It allows you to define custom validations for specific records. This helps detect errors when users input data or when modifying records.

The **record** parameter in the **checkRecord** method provides read-only access to the current record. It is available at record-level check for the constraint type **Constraint on Table** . The fields of the record are defined by the current EBX® schema.

Example 1:

```
uses core.string as str;
uses core.message as msg;

export procedure checkRecord(record :.Employee)
begin

// This constraint checks if the email ends with tibco.com and adds a warning message if not.
if (not(str.endsWith(record.Email,'tibco.com'))) then
  begin
    msg.addWarning('The email '|record.Email |' should end with tibco.com');
  end
end
end

export procedure setup()
begin
  ...
```

```
end
```

Example 2:

```
uses core.message as msg;
export procedure checkTable()
begin
end

export procedure checkRecord(record : .Product)
begin

// This constraint checks if the price of the product is under a threshold.
if (record.Price < _ebx.root.PriceThreshold) then
 begin
   var message:='The price of the product '|record.Price |' is under the threshold '|_ebx.root.PriceThreshold;
   msg.addWarning(message);
 end

end
export procedure setup()
begin
  ...
end
```

Constraint on field

This constraint type defines a field-level check. It ensures that the field satisfies a specific condition. A localized message can be added if the field is invalid. You can define this constraint type on a dataset field, or at a table field.

Dataset field-level check

This constraint type defines a dataset field-level check. Use the predefined variable **_ebx.root** to access other dataset fields.

Example:

```
uses core.message as msg;
export procedure check()
begin

// This constraint checks if the image height is between 150 and 500.
if (_ebx.root.image.height > 150 and _ebx.root.image.height < 500) then
  begin
    // add a warning message
    var message:='Image height should be between 150 and 500. Current height is '|_ebx.root.image.height;
    msg.addWarning(message);
  end

end
export procedure setup()
begin
  ...
end
```

Table field-level check

This constraint type defines a table field-level check. It ensures that the field satisfies a specific condition for each record in the table. A localized message can be added if the field is invalid. The variable **record** is passed by parameter in the **check()** . It provides read-only access to the current field’s containing record.

Example:

```
uses core.message as msg;
// This constraint checks if the project's completed tasks are under 10 and adds an info message
export procedure check(record : .Project)
begin
if (record.Number < 10) then
 begin
  msg.addInfo('Completed tasks is under 10'|record.ProjectId);
 end
```

```
end

export procedure setup()
begin
end
```

Constraint on null

This constraint type defines a dataset or table field-level check on whether a field contains a value. You can add a message if the field is null.

Dataset field-level check

Example:

```
uses core.message as msg;
// This constraint adds an error message if the project code is missing.
export procedure checkNull()
begin

//Add a message from a bundle with one parameter: 'The code of the project {0} is mandatory'
var message:= msg.ofBundleError('bundle1/path1','project.code.mandatory',_ebx.root.project.name);
msg.addMessage(message);

end
export procedure setup()
begin
  ...
end
```

Table field-level check

Example:

```
uses core.message as msg;
// This constraint checks that the publication year is present for the Book.
// If the publication year is missing, an error message is added to the validation report.
export procedure checkNull(record : .Book)
begin
  msg.addError('Publication year is mandatory');
end

export procedure setup()
begin
end
```

197.5 Constraint dependencies

The constraint validation result can depend on updates, insertions or deletions of specific fields. In these cases, you can explicitly declare the field's dependencies during the initialization of the constraint. Declaring constraint dependencies helps to minimize the number of calls to constraints during the validation process.

You do not have to specify dependencies. However, explicit dependencies improve performance when you validate large volumes of data or when the validation logic is expensive.

The EBX® Script API core.validation [p 1386] provides several methods for declaring a constraint's dependencies.

Warning: you cannot specify dependencies on computed values or inherited fields.

Insertion, deletion and modification dependency

The validation result of the constraint depends on insertions, modifications and deletions of the specified field.

Example:

```
uses core.validation as v;
export procedure checkTable()
begin
  ...
end
export procedure checkRecord(record :.Product)
begin
  ...
end

export procedure setup()
begin
  // Declare that this constraint depends on the current record's 'Price' and 'Quantity' fields.
  v.addDependencyToInsertDeleteAndModify('/root/Product/Price');
  v.addDependencyToInsertDeleteAndModify('/root/Product/Quantity');
end
```

Insertion and deletion dependency

The validation result of the constraint depends on insertions and deletions of the specified table.

Example:

```
uses core.validation as v;
export procedure checkTable()
begin
  ...
end
export procedure checkRecord(record : .Customers)
begin
  ...
end
export procedure setup()
begin
  // Declares that this constraint depends on the 'Orders' table.
  v.addDependencyToInsertAndDelete('/root/Orders');
end
```

Modification dependency

The validation result of the constraint depends on the value of the specified field.

Example:

```
uses core.message as msg;
uses core.validation as v;
export procedure check()
begin
    ...
end

export procedure setup()
begin
 // Declares that this constraint depends on modification of the code field.
 v.addDependencyToModify('/root/Code');
end
```

Dependency on another dataset

The validation result of the constraint depends on the insertions, modifications and deletions of the specified field or table in specified dataset and dataspace.

Example:

```
uses core.validation as v;
export procedure checkTable()
begin
  ...
end
export procedure checkRecord(record : .Customers)
begin
  ...
end
```

```
export procedure setup()
begin
  // Declare that this constraint depends on the 'Status' field of another dataset and dataspace.
  v.addDependencyToInsertAndDeleteAndModifyInOtherDataset("/root/Orders/Status", 'otherDataset',
 'otherDataspace');
  v.addDependencyToInsertAndDeleteInOtherDataset('/root/Items','otherDataset','otherDataspace');
end
```

CHAPTER 198

Unit summary

Units	Description
default [p 1389]	The default unit.
core.complex [p 1393]	Script unit that provides methods to create and update complex values.
core.data [p 1398]	Script unit that provides database operations (insert, modify, update, lookup).
core.date [p 1420]	Script unit that provides date functions.
core.datetime [p 1426]	Script unit that provides datetime functions.
core.dictionary [p 1434]	Script unit that provides methods to create and manipulate dictionaries.
core.http [p 1442]	Script unit that provides methods to call an external REST service.
core.json [p 1455]	Script unit that provides methods to manage json data.
core.list [p 1470]	Script unit that provides methods to create and update lists.
core.locale [p 1477]	Script unit that provides functions for managing locales.
core.log [p 1479]	Script unit that provides methods to log messages.
core.math [p 1482]	Script unit that provides mathematical functions and constants.
core.message [p 1496]	Script unit that provides methods to create and manage user messages
core.number	Script unit that provides number functions and constants.
core.profile [p 1503]	Script unit that provides methods useful when managing profiles.
core.resource [p 1505]	This script unit methods that generates resource references compatible with the schema type ods:resource.

Units	Description
core.rest [p 1545]	Script unit that provides methods useful when writing REST services.
core.set [p 1510]	Script unit that provides methods to create and manipulate sets.
core.string [p 1518]	Script unit that provides string manipulation functions.
core.time [p 1533]	Script unit that provides time functions.
core.uri [p 1537]	Script unit that provides functions for managing Uniform Resource Identifier (URI) references.
core.validation [p 1539]	Script unit that provides methods for setting dependencies in the context of a validation constraint.

CHAPTER 199

Unit default

The default unit. This unit is automatically included in a script.

Methods from this script can be called directly and do not require a uses statement.

Methods
instanceOf<type>(): type [p 1389] Creates an object instance for a given type.
function isFalseOrNull(value: boolean): boolean [p 1390] Returns true if value is false of null.
function isNotNull(value: any_type): boolean [p 1390] Returns true if value is not null.
function isNull(value: any_type): boolean [p 1390] Returns true if value is null.
function isTrueOrNull(value: boolean): boolean [p 1391] Returns true if value is true or null.

This chapter contains the following topics:

1. instanceOf
2. isFalseOrNull
3. isNotNull
4. isNull
5. isTrueOrNull

199.1 instanceOf

instanceOf<type>(): type

Creates an object instance for a given type.

Example:

```
export function getValue(): typeof _ebx.record.OfficeAddress
begin
  var value := create<typeof _ebx.record.OfficeAddress>();
  value.Street := '4323 Broadway';
  value.City := 'New York';
  value.State := 'NY';
  value.Zip := '10019';
  value.Country := 'USA';
  return value;
end
```

Function Types:

type: the object type. Is mandatory. Some types are not supported such as those that can be created using literals (string, numbers...) or that can only be instantiated by EBX.

Return:

the new object. Is always the mutable version of the specified type.

Can be used in:

199.2 isFalseOrNull

function isFalseOrNull(value: boolean): boolean

Returns true if value is false of null.

Parameters:

value: a boolean value.

Return:

true if value is null or false, or else false. Never returns null.

Can be used in:

199.3 isNotNull

function isNotNull(value: any_type): boolean

Returns true if value is not null.

Parameters:

value a value of any type.

Return:

true if value is not null or else false. Never returns null.

Can be used in:

199.4 isNull

function isNull(value: any_type): boolean

Returns true if value is null.

Parameters:

value a value of any type.

Return:

true if value is null or else false. Never returns null.

Can be used in:

199.5 isTrueOrNull

function isTrueOrNull(value: boolean): boolean

Returns true if value is true or null.

Parameters:

value: a boolean value.

Return:

true if value is null or true, or else false. Never returns null.

Can be used in:

CHAPTER 200

Unit core.complex

Script unit that provides methods to create and update complex values.

Methods
foreignKeyOf<foreignKeyType>(): foreignKeyType [p 1393] Create an instance of a foreign key type.
formatPrimaryKey(primaryKey: primaryKeyType): string [p 1394] Returns the internal tableRef String representation of a primary key.
of<complexType>(): complexType [p 1394] Create an instance of a complex type.
parsePrimaryKey<primaryKeyType>(value: string): primaryKeyType [p 1395] Parses a string to a primary key.
primaryKeyOf<primaryKeyType>(): primaryKeyType [p 1395] Create an instance of a primary key type.

This chapter contains the following topics:

1. foreignKeyOf
2. formatPrimaryKey
3. of
4. parsePrimaryKey
5. primaryKeyOf

200.1 foreignKeyOf

foreignKeyOf<foreignKeyType>(): foreignKeyType

Create an instance of a foreign key type.

Note: A foreign key field contains a primary key on the referenced table.

This method is equivalent to the method complex.primaryKeyOf. The method complex.primaryKeyOf should be preferred

Example:

```
uses core.complex as complex;

export function getValue(): typeof _ebx.record.Supervisor
begin
  var foreignKey := complex.foreignKeyOf<typeof _ebx.record.Supervisor>();
  foreignKey.Id := 435;
  return foreignKey;
end
```

Function Types:

primaryKeyType: the primary key type. Can be the type of a foreign key or a an expression foreignKeyOf. Is mandatory.

Return:

the new foreign key object. Is always mutable.

Can be used in:

200.2 formatPrimaryKey

formatPrimaryKey(primaryKey: primaryKeyType): string

Returns the internal tableRef String representation of a primary key.

Function Types:

string: the primary key internal representation.

Parameters:

primaryKey: the primary key

Return:

the new mutable primary key object if the string is parsable, otherwise null.

Can be used in:

200.3 of

of<complexType>(): complexType

Create an instance of a complex type.

Example:

```
uses core.complex as complex;

export function getValue(): typeof _ebx.record.OfficeAddress
begin
  var value := complex.of<typeof _ebx.record.OfficeAddress>();
  value.Street := '4323 Broadway';
  value.City := 'New York';
  value.State := 'NY';
  value.Zip := '10019';
  value.Country := 'USA';
  return value;
end
```

Function Types:

complexType: the object type. Is mandatory. Should be an expression typeof identifier.

Return:

the new object. Is always the mutable version of the specified type.

Can be used in:

200.4 parsePrimaryKey

parsePrimaryKey<primaryKeyType>(value: string): primaryKeyType

Parses a string to a primary key.

Example:

```
uses core.complex as complex;

export function getValue(): typeof _ebx.record.Supervisor
begin
  var foreignKey := complex.parsePrimaryKey<typeof _ebx.record.Supervisor>('567');
  return foreignKey;
end
```

Function Types:

primaryKeyType: the primary key type. Can be the type of a foreign key or an expression primaryKeyOf. Is mandatory.

Parameters:

value: a Stringcontaining the primary key internal representation to be parsed

Return:

the new mutable primary key object if the string is parsable, otherwise null.

Can be used in:

200.5 primaryKeyOf

primaryKeyOf<primaryKeyType>(): primaryKeyType

Create an instance of a primary key type.

Note: A foreign key field contains a primary key on the referenced table.

Example:

```
uses core.complex as complex;

export function getValue(): typeof _ebx.record.Supervisor
begin
  var foreignKey := complex.primaryKeyOf<typeof _ebx.record.Supervisor>();
  foreignKey.Id := 435;
  return foreignKey;
end
```

Function Types:

primaryKeyType: the primary key type. Can be the type of a foreign key or an expression primaryKeyOf. Is mandatory.

Return:

the new primary key object. Is always mutable.

Can be used in:

CHAPTER 201

Unit core.data

Script unit that provides database operations (insert, modify, update, lookup).

Methods
procedure addDataset(sql: queryType, alias: string, dataset: datasetType) [p 1402] Adds a dataset to a SQL query.
function createQuery(sql: string, datasets: string...): [p 1402] Creates a SQL query.
procedure deleteRecord(transaction: transactionType, record: recordType) [p 1402] Deletes a record.
procedure deleteRecordByPrimaryKey(transaction: transactionType, table: tableType, primaryKey: primaryKeyType) [p 1403] Deletes a record specified by its primary key.
function executeQuery(query: query): list of tupleType. [p 1403] Executes the SQL query.
function explain(query: query): string. [p 1404] Returns a meaningful string representation of the SQL query plan.
function fetchFirst(query: query): tuple. [p 1404] Executes the SQL query and returns the first tuple.
function fetchOne(query: query): tuple. [p 1405] Executes the SQL query and returns a single tuple or null if the number of tuple is not exactly one.
function field<fieldType>(tuple: tupleType, alias: string): fieldType. [p 1405] Returns the field of the SQL tuple for the specified alias.
function fieldByIndex<fieldType>(tuple: tupleType, index: int): fieldType. [p 1406] Returns the field of the SQL tuple for the specified position.
function fieldNames(result: query_result): list<string> [p 1406] Returns the names of the fields in the result of the SQL query.

Methods
function findDataset(dataspace: dataspaceType, name: string): datasetType [p 1407] Returns the dataset specified by its name.
function findDataspace(name: string): dataspaceType [p 1407] Returns the dataspace specified by its name.
function findSnapshot(name: string): snapshotType [p 1407] Returns the snapshot specified by its name.
function findTable<tableType>(dataset: datasetType): tableType [p 1408] Returns a table of the specified dataset.
function lookupDataset(dataspace: dataspaceType, name: string): datasetType [p 1408] Returns the dataset specified by its name.
function lookupDatasetField(dataset: datasetType): fieldType [p 1409] This method is deprecated and replaced by lookupDatasetRoot.
function lookupDatasetFieldForUpdate(transaction: transactionType, dataset: datasetType): updatableFieldType [p 1409] This method is deprecated and replaced by lookupDatasetRootForUpdate.
procedure lookupDatasetRoot<modelAlias>(dataset: datasetType) [p 1409] Returns dataset root.
procedure lookupDatasetRootForUpdate<modelAlias>(transaction: transactionType, dataset : datasetType) updatableDatasetRootType [p 1410] Returns an updatable dataset root.
function lookupDataspace(name: string): dataspaceType [p 1410] Returns the dataspace specified by its name.
function lookupRecord(table: tableType, primaryKey: PrimaryType): recordType [p 1411] Returns a read only record specified by its primary key.
function lookupRecordByFullXPath(table: tableType, xPath: string): recordType [p 1411] Returns a read only record specified by an XPath.
function lookupRecordByFullXPathForUpdate(transaction:transactionType,table: tableType, xPath: string): recordType [p 1412] Returns an updatable record specified by an XPath.
function lookupRecordForUpdate(transaction: transactionType, table: tableType, tableRef: string): updatableRecordType [p 1412] Returns an updatable record specified by its primary key.
function lookupRecordForUpdate(transaction: transactionType, table: tableType, primaryKey: PrimaryKeyType): updatableRecordType [p 1413] Returns an updatable record specified by its primary key.

Methods
function lookupSnapshot(name: string): snapshotType [p 1413] Returns the snapshot specified by its name.
function record<recordType>(tuple: tupleType, alias: string): record. [p 1413] This method is deprecated and replaced by field.
function recordByIndex<recordType>(tuple: tupleType, index: int): recordType. [p 1414] This method is deprecated and replaced by fieldByIndex.
function recordOf(transaction: transactionType, table: tableType): recordType [p 1414] Create a new record for a table.
procedure saveDatasetRoot(root: updatableDatasetRootType) [p 1415] Saves the all dataset fields.
procedure saveField(field: updatableFieldType) [p 1415] This method is deprecated and replaced by saveDatasetRoot.
procedure saveRecord(record: updatableRecordType) [p 1415] Saves the record.
procedure setFetchSize(query: query, number: int) [p 1416] Provides a hint to the underlying data with the number of records to fetch at a time from the SQL query.
procedure setLocale(query: query, locale: locale) [p 1416] Sets the locale for the SQL query.
procedure setParameter(query: query, position: int, value: fieldType) [p 1416] Sets a value to a positional parameter in the SQL query.

This chapter contains the following topics:

1. addDataset
2. createQuery
3. deleteRecord
4. deleteRecordByPrimaryKey
5. executeQuery
6. explain
7. fetchFirst
8. fetchOne
9. field
10. fieldByIndex
11. fieldNames
12. findDataset
13. findDataspace
14. findSnapshot
15. findTable
16. lookupDataset
17. lookupDatasetField
18. lookupDatasetFieldForUpdate
19. lookupDatasetRoot
20. lookupDatasetRootForUpdate
21. lookupDataspace
22. lookupRecord
23. lookupRecordByFullXPath
24. lookupRecordByFullXPathForUpdate
25. lookupRecordByRefForUpdate
26. lookupRecordForUpdate
27. lookupSnapshot
28. record
29. recordByIndex
30. recordOf
31. saveDatasetRoot
32. saveField
33. saveRecord
34. setFetchSize
35. setLocale

36.setParameter

201.1 addDataset

procedure addDataset(sql: queryType, alias: string, dataset: datasetType)

Adds a dataset to a SQL query. This method must be called before executeQuery or explain.

Example:

```
uses core.data as d;
export procedure onAfterCreate()
begin
  var sql :='select t.calA, t.colB from d1.tableA t where t.id=xx';
  var query :=d.createQuery(sql);
  var dataset1 := d.findDataset(_ebx.dataspace, 'dataset-name1');
  d.addDataset(query, 'd1', dataset1);
end
```

Parameters:

query: the sql query.

alias: the alias of the dataset to be added to the query.

dataset: the dataset to be added to the query.

Can be used in:

201.2 createQuery

function createQuery(sql: string, datasets: string...):

Creates a SQL query. Raises an error if the sql string is not a valid select; for example, it contains syntax errors, it references missing tables, missing columns, etc.

Example:

```
uses core.data as d;
export procedure onAfterCreate()
begin
  var sql := 'select t.calA, t.colB from tableA t where t.id=xx';
  var query := d.createQuery(sql);
end
```

Parameters:

sql: the string representation of the sql query.

Return:

the query. Is never null.

Can be used in:

201.3 deleteRecord

procedure deleteRecord(transaction: transactionType, record: recordType)

Deletes a record. Raises an error if the deletion fails.

Example:

```
uses core.data as d;
export procedure onAfterDelete()
begin
  var table := d.findTable<.tableName>(_ebx.dataset);
  var table := d.findTable<.tableName>(_ebx.dataset);
  var record := d.lookupRecord(tableA,_ebx.record.foreignKey);
```

```
  // deletes the record
  d.deleteRecord(_ebx.transaction,record);
end
```

Parameters:

transaction: the transaction.

record: the record.

Can be used in:

201.4 deleteRecordByPrimaryKey

procedure deleteRecordByPrimaryKey(transaction: transactionType, table: tableType, primaryKey: primaryKeyType)

Deletes a record specified by its primary key. Raises an error if the deletion fails. Deleting a record that does not exist is not an error.

. Example 1:

```
uses core.data as d;
uses core.complex as c;
export procedure onAfterDelete()
begin
  var table := d.findTable<.tableName>(_ebx.dataset);

  // Creates the primary key.
  var pk := c.primaryKeyOf<.tableName>();
  pk.field1 := ...;
  pk.field2 := ...;

  // Deletes the record using the primary key.
  d.deleteRecordByPrimaryKey(_ebx.transaction, table, pk);
end
```

Example 2:

```
uses core.data as d;
export procedure onAfterDelete()
begin
  var table := d.findTable<.tableName>(_ebx.dataset);

  // Deletes the record referenced by a foreign key. A foreign key is also a primary key.
  d.deleteRecordByPrimaryKey(_ebx.transaction,table,_ebx.record.foreignKey);
end
```

Parameters:

transaction: the transaction.

table: the table.

primaryKey: the primary key.

Can be used in:

201.5 executeQuery

function executeQuery(query: query): list of tupleType.

Executes the SQL query. Raises an error if nodes used in the query are forbidden.

Example:

```
uses core.data as d;
export procedure onAfterCreate()
begin
  var query := d.createQuery('select t.colA, t.colB" from s1.tableA t where t.colC=?');
  d.addDataset(query, 's1', d.findDataset(_ebx.dataspace, 'dataset-name'));
  a.setParameter(query, 0, 'A value');
```

```
  var queryResult := d.executeQuery(query);
  for tuple in queryResult do
  begin
    // Do something with tuple.
    ...
  end
  d.close(queryResult)
end
```

Parameters:

query: the sql query.

Return:

the result for the query. Is never null.

Can be used in:

201.6 **explain**

function explain(query: query): string.

Returns a meaningful string representation of the SQL query plan.

Example:

```
uses core.data as d;
export procedure onAfterCreate()
begin
  var query := d.createQuery('select t."$pk" as id from table t');
  var queryPlan := d.explain(query);
  ...
end
```

Parameters:

query: the sql query.

Return:

the representation of the query plan.

Can be used in:

201.7 **fetchFirst**

function fetchFirst(query: query): tuple.

Executes the SQL query and returns the first tuple. Raises an error if nodes used in the query are forbidden.

Example:

```
uses core.data as d;
export procedure onAfterCreate()
begin
  var query := d.createQuery('select t.colA, t.colB" from s1.tableA t where t.colC=?');
  d.addDataset(query, 's1', d.findDataset(_ebx.dataspace, 'dataset-name'));
  a.setParameter(query, 0, 'A value');

  var tuple := d.fetchFirst(query);
  // Do something with tuple.
  ...
end
```

Parameters:

query: the sql query.

Return:

the first tuple for the query or null if query returns no result.

Can be used in:

201.8 fetchOne

function fetchOne(query: query): tuple.

Executes the SQL query and returns a single tuple or null if the number of tuple is not exactly one. Raises an error if nodes used in the query are forbidden.

Example:

```
uses core.data as d;
export procedure onAfterCreate()
begin
  var query := d.createQuery('select t.colA, t.colB" from s1.tableA t where t.colC=?');
  d.addDataset(query, 's1', d.findDataset(_ebx.dataspace, 'dataset-name'));
  a.setParameter(query, 0, 'A value');

  var tuple := d.fetchOne(query);
  // Do something with tuple.
  ...
end
```

Parameters:

query: the sql query.

Return:

the first tuple for the query or null if query returns no result.

Can be used in:

201.9 field

function field<fieldType>(tuple: tupleType, alias: string): fieldType.

Returns the field of the SQL tuple for the specified alias.

Note: This method is not recommended for tuples from embedded SQL queries because compiler cannot check the return type.

Example:

```
uses core.data as d;
export procedure onBeforeDelete()
begin
  var query := d.createQuery('select a.col1 as c1, a.col2 as c2 from tableA a');
  var queryResult := d.executeQuery(query);
  for tuple in queryResult do
  begin
    var field1 := d.field<string>(tuple,'c1');
    var field2 := d.field<decimal>(tuple,'c2');
    // Do something with field values
    ...
  end;
  ...
end
```

Function Types:

fieldType: the field type. Is mandatory.

Parameters:

tuple: the tuple of the query result.

alias: the alias of the field.

Can be used in:

201.10 fieldByIndex

function fieldByIndex<fieldType>(tuple: tupleType, index: int): fieldType.

Returns the field of the SQL tuple for the specified position.

Note: This method is not recommended for tuples from embedded SQL queries because compiler cannot check the return type.

Example:

```
uses core.data as d;
export procedure onBeforeModify()
begin
  var query := d.createQuery('select a.col1, a.col2 from tableA a');
  var queryResult := d.executeQuery(query);
  for tuple in queryResult do
  begin
    var field1 := d.fieldByIndex<string>(tuple,0);
    var field2 := d.fieldByIndex<date>(tuple,1);
    // Do something with field values.
    ...
  end;
  ...
end
```

Function Types:

fieldType: the field type. Is mandatory.

Parameters:

tuple: the tuple of the query result.

index: the position of the field. First position is 0.

Can be used in:

201.11 fieldNames

function fieldNames(result: query_result): list<string>

Returns the names of the fields in the result of the SQL query.

Example:

```
uses core.data as d;
export procedure onAfterCreate()
begin
  var query := d.createQuery('select t.* from table t');
  var names := d.fieldNames(query);
  for name in names do
  begin
    // Do something with name.
    ...
  end;
end
```

Parameters:

result: the sql query.

Return:

the names of the fields in the query result. Is never null.

Can be used in:

201.12 **findDataset**

function findDataset(dataspace: dataspaceType, name: string): datasetType

Returns the dataset specified by its name. Raises an error if the dataset is not found.

```
uses core.data as d;

procedure doSomething()
begin
  var dataspace := d.findDataspace('dataspace-name');
  var dataset := d.findDataset(dataspace, 'dataset-name');
  ...
end
```

Parameters:

dataspace: the dataspace that owns the dataset.

name: the name of the dataset.

Return:

the dataset specified by its name. Is never null.

Can be used in:

201.13 **findDataspace**

function findDataspace(name: string): dataspaceType

Returns the dataspace specified by its name. Raises an error if the dataspace is not found.

Example:

```
uses core.data as d;

begin
  var dataspace := d.findDataspace('dataspace-name');
  ...
end
```

Parameters:

name: the name of the dataspace.

Return:

the dataspace. Is never null.

Can be used in:

201.14 **findSnapshot**

function findSnapshot(name: string): snapshotType

Returns the snapshot specified by its name. Raises an error if the snapshot is not found.

Example:

```
uses core.data as d;

procedure doSomething()
begin
  var snapshot := d.findSnapshot('snapshot-name');
  ...
end
```

Parameters:

name: the name of the snapshot.

Return:

the snapshot. Is never null.

Can be used in:

201.15 findTable

function findTable<tableType>(dataset: datasetType): tableType

Returns a table of the specified dataset. Raises an error if the table is not found.

Example:

```
uses core.data as d;

procedure doSomething()
begin
  var customerTable := db.findTable<.Customer>(_ebx.dataset);
  ...
end
```

Function Types:

complexType: the table type.

Parameters:

dataset: the dataset that owns the table.

Return:

the table. Is never null.

Can be used in:

201.16 lookupDataset

function lookupDataset(dataspace: dataspaceType, name: string): datasetType

Returns the dataset specified by its name. Returns null if the dataset is not found.

```
uses core.data as d;

procedure doSomething()
begin
  var dataspace := d.lookupDataspace('dataspace-name');
  var dataset := d.lookupDataset(dataspace, 'dataset-name');
  ...
end
```

Parameters:

dataspace: the dataspace that owns the dataset.

name: the name of the dataset.

Return:

the dataset specified by its name or null if not found.

Can be used in:

201.17 lookupDatasetField

function lookupDatasetField(dataset: datasetType): fieldType

This method is deprecated and replaced by lookupDatasetRoot.

Returns a read only field specified by the dataset.

Parameters:

dataset: The dataset.

Return:

the read only field.

Can be used in:

201.18 lookupDatasetFieldForUpdate

function lookupDatasetFieldForUpdate(transaction: transactionType, dataset: datasetType): updatableFieldType

This method is deprecated and replaced by lookupDatasetRootForUpdate.

Returns an updatable field of the specified dataset. Raises an error if the dataset is not owned by the dataspace used to create the transaction.

Note: Currently, it's not possible to access simple dataset fields that are directly under the root.

Parameters:

transaction: the transaction.

dataset: The dataset must be owned by the dataspace used to create the transaction.

Return:

the updatable field.

Can be used in:

201.19 lookupDatasetRoot

procedure lookupDatasetRoot<modelAlias>(dataset: datasetType)

Returns dataset root. Raises an error if the operation fails.

Example:

```
references model "sample-model" as data;
uses core.data as db;

export procedure executeScriptTask()
begin
  const dataspace: dataspace := db.findDataspace('test-dataspace');
  const dataset: dataset := db.findDataset(dataspace, 'test-dataset');

  var root := db.lookupDatasetRoot<data>(dataset);
  var name := root.name;
  var city := root.address.city;
  ....
end
```

Parameters:

dataset: the dataset.

Return:

the dataset's root. Is never null.

Can be used in:

201.20 lookupDatasetRootForUpdate

procedure lookupDatasetRootForUpdate<modelAlias>(transaction: transactionType, dataset : datasetType) updatableDatasetRootType

Returns an updatable dataset root. Raises an error if the operation fails.

Example:

```
references model "sample-model" as data;
uses core.data as db;

export procedure executeScriptTask()
begin
  const dataspace: dataspace := db.findDataspace('test');

  execute transaction tr on dataspace
  begin
    var dataset := db.findDataset(dataspace, 'test-dataset');
    var root := db.lookupDatasetRootForUpdate<data>(tr, dataset);
    // Modify a dataset field.
    root.address.city := 'Paris';

    // Persist the changes.
    db.saveDatasetRoot(root);
  end
end
```

Parameters:

transaction: the transaction.

dataset: the dataset.

Return:

the updatable dataset's root. Is never null.

Can be used in:

201.21 lookupDataspace

function lookupDataspace(name: string): dataspaceType

Returns the dataspace specified by its name. Returns null if the dataspace is not found.

Example:

```
uses core.data as d;

procedure doSomething()
begin
  var dataspace := d.lookupDataspace('dataspace-name');
  ...
end
```

Parameters:

name: the name of the dataspace.

Return:

the dataspace or null if not found.

Can be used in:

201.22 lookupRecord

function lookupRecord(table: tableType, primaryKey: PrimaryType): recordType

Returns a read only record specified by its primary key. Returns null if the record does not exist.

Example:

```
uses core.data as d;

export procedure onAfterCreate()
begin
  var table := d.findTable<.tableName>(_ebx.dataset);
  // Fetch the record referenced by the foreign key.
  var record := d.lookupRecord(table, _ebx.record.foreignKey);
  // Do something with record.
  ..
end
```

Parameters:

table: the table.

primaryKey: the primary key.

Return:

the record specified by its primary key or null if not found.

Can be used in:

201.23 lookupRecordByFullXPath

function lookupRecordByFullXPath(table: tableType, xPath: string): recordType

Returns a read only record specified by an XPath. Returns null if the record does not exist.

The XPath must specify the value of each primary key field.

An example of XPath is:

```
/root/table[./pk-field1='a' and ./pk-field2='b']
```

Example:

```
uses core.data as d;

export procedure onAfterCreate()
begin
  var table := d.findTable<.tableName>(_ebx.dataset);
  var xpath:='/root/table[./pk-field1='a' and ./pk-field2='b']';
  // Fetch the record specified by an XPath.
  var record := d.lookupRecordByFullXPath(table, xpath);
  // Do something with record.
  ..
end
```

Parameters:

table: the table.

xPath: the xPath expression.

Return:

the record specified by the xPath or null if not found.

Can be used in:

201.24 lookupRecordByFullXPathForUpdate

function lookupRecordByFullXPathForUpdate(transaction:transactionType,table: tableType, xPath: string): recordType

Returns an updatable record specified by an XPath. Returns null if the record does not exist.

The XPath must specify the value of each primary key field.

An example of XPath is:

```
/root/table[./pk-field1='a' and ./pk-field2='b']
```

Example:

```
uses core.data as d;

export procedure onAfterCreate()
begin
  var table := d.findTable<.tableName>(_ebx.dataset);
  var xpath='/root/table[./pk-field1='a' and ./pk-field2='b']';
  // Fetch the record specified by an XPath.
  var record := d.lookupRecordByFullXPathForUpdate(_ebx.transaction,table, xpath);
  // Update the record.
  record.field3:='XX';
  record.field4:='YY';
  ..
  d.saveRecord(record);
end
```

Parameters:

transaction: the transaction.

table: the table.

xPath: the xPath expression.

Return:

the record specified by the xPath or null if not found.

Can be used in:

201.25 lookupRecordByRefForUpdate

function lookupRecordForUpdate(transaction: transactionType, table: tableType, tableRef: string): updatableRecordType

Returns an updatable record specified by its primary key. Returns null if the record does not exist or tableRef does not have a valid format

Parameters:

transaction: the transaction.

table: the table.

tableRef: the table ref.

Return:

the updatable record specified by its tableRef or null if not found or tableRef does not have a valid format.

Can be used in:

201.26 lookupRecordForUpdate

function lookupRecordForUpdate(transaction: transactionType, table: tableType, primaryKey: PrimaryKeyType): updatableRecordType

Returns an updatable record specified by its primary key. Returns null if the record does not exist.

Example:

```
uses core.data as d;

export procedure onAfterCreate()
begin
  var table := d.findTable<.tableName>(_ebx.dataset);
  // Fetch the record referenced by the foreign key.
  var record := d.lookupRecordForUpdate(_ebx.transaction,table, _ebx.record.foreignKey);
  // Update the record.
  record.fieldA:='A';
  record.fieldB:='B';
  ..
  d.saveRecord(record);
end
```

Parameters:

transaction: the transaction.

table: the table.

primaryKey: the primary key.

Return:

the updatable record specified by its primary key or null if not found.

Can be used in:

201.27 lookupSnapshot

function lookupSnapshot(name: string): snapshotType

Returns the snapshot specified by its name. Returns null if the specified snapshot is not found.

Example:

```
uses core.data as d;

procedure doSomething()
begin
  var snapshot := d.lookupSnapshot('snapshot-name');
  ...
end
```

Parameters:

name: the name of the snapshot.

Return:

the snapshot or null if not found.

Can be used in:

201.28 record

function record<recordType>(tuple: tupleType, alias: string): record.

This method is deprecated and replaced by field.

Returns a read-only record by specified alias.

Function Types:

recordType: the record type. Is mandatory.

Parameters:

tuple: the tuple of the query result.

alias: the alias of the field.

Can be used in:

201.29 recordByIndex

function recordByIndex<recordType>(tuple: tupleType, index: int): recordType.

This method is deprecated and replaced by fieldByIndex.

Returns a read-only record at index position.

Function Types:

complexType: the record type. Is mandatory.

Parameters:

tuple: the tuple of the query result.

index: the position of the field.

Can be used in:

201.30 recordOf

function recordOf(transaction: transactionType, table: tableType): recordType

Create a new record for a table. Default values are set as specified in the schema.

Example:

```
uses core.data as d;

export procedure onAfterCreate()
begin
  var table := d.findTable<.tableName>(_ebx.dataset);
  var record := d.recordOf(_ebx.transaction, table);
  // Update the record.
  record.pk:='IDA';
  record.field1:='A1';
  record.field2:='A2';
  ...
  d.saveRecord(record);
end
```

Parameters:

transaction: the transaction.

table: the table.

Return:

the new record. Is never null.

Can be used in:

201.31 saveDatasetRoot

procedure saveDatasetRoot(root: updatableDatasetRootType)

Saves the all dataset fields. Raises an error if the operation fails.

Example:

```
references model "sample-model" as data;
uses core.data as db;

export procedure executeScriptTask()
begin
  const dataspace: dataspace := db.findDataspace('test');

  execute transaction tr on dataspace
  begin
    var dataset := db.findDataset(dataspace, 'test-dataset');
    var root := db.lookupDatasetRootForUpdate<data>(tr, dataset);
    // Modify a dataset field.
    root.address.city := 'Paris';

    // Persist the changes.
    db.saveDatasetRoot(root);
  end
end
```

Parameters:

root: the root of the dataset to save.

Can be used in:

201.32 saveField

procedure saveField(field: updatableFieldType)

This method is deprecated and replaced by saveDatasetRoot.

Saves the dataset field. Raises an error if the operation fails.

references model "myModel" as m;

Parameters:

field: the dataset field to save.

Can be used in:

201.33 saveRecord

procedure saveRecord(record: updatableRecordType)

Saves the record. Raises an error if the operation fails.

The record is updated with new values. Example:

```
uses core.data as d;

export procedure onAfterCreate()
begin
  var table := d.findTable<.tableName>(_ebx.dataset);
  var record := d.recordOf(_ebx.transaction, table);
  // Update the record.
  ...
  d.saveRecord(record);
end
```

Parameters:

record: the record to save.

Can be used in:

201.34 setFetchSize

procedure setFetchSize(query: query, number: int)

Provides a hint to the underlying data with the number of records to fetch at a time from the SQL query.

Example:

```
uses core.data as d;
export procedure onAfterCreate()
begin
  var sql := 'select * from tableA where id=xx';
  var query := d.createQuery(sql);
  d.setFetchSize(query,50);
  ...
end
```

Parameters:

query: the query.

number: the number of record to fetch.

Can be used in:

201.35 setLocale

procedure setLocale(query: query, locale: locale)

Sets the locale for the SQL query. Should be called before method executeQuery.

Example:

```
uses core.data as d;
uses core.locale as locale;
export procedure onAfterModify()
begin
  var query := d.createQuery('select * from table tableA');
  d.setLocale(query,locale.of('fr','FR'));
  var queryResult := d.executeQuery(query);
  ...
end
```

Parameters:

query: the query.

locale: the locale.

Can be used in:

201.36 setParameter

procedure setParameter(query: query, position: int, value: fieldType)

Sets a value to a positional parameter in the SQL query. Example:

```
uses core.data as d;
uses core.locale as locale;
export procedure onAfterModify()
begin
  var query := d.createQuery('select * from table tableA t where t.col1=? and t.col2=?');
  d.setParameter(query, 0, 'Paris');
  d.setParameter(query, 1, d(1956-06-17));
  var queryResult := d.executeQuery(query);
  ...
```

```
end
```

Parameters:

query: the query.

position: the position of the parameter in the query.

value: value of the parameter.

Can be used in:

CHAPTER 202

Unit core.date

Script unit that provides date functions.

Methods
function addDays(value: date, days: decimal): date [p 1421] Adds days to a date.
function addMonths(value: date, months: decimal): datetime [p 1421] Adds months to a date.
function addYears(value: date, years: decimal): date [p 1421] Adds years to a date.
function days(startValue: date, endValue: date): decimal [p 1422] Returns the number of days between two dates.
function fromDatetime(value: datetime): date [p 1422] Converts a date and time to a date.
function fromIsoString(value: string): date [p 1422] Parses a date from a string in ISO format (yyyy-MM-dd)
function months(startValue: date, endValue: date): decimal [p 1423] Returns the number of months between two dates.
function now(): date [p 1423] Returns current date.
function dateOf(year: decimal, month: decimal, day: decimal): date [p 1423] Creates a date given year, month and day.
function toIsoString(value: date): string [p 1423] Converts a date to a string in ISO format (yyyy-MM-dd)
function years(startValue: date, endValue: date): decimal [p 1424] Returns the number of full years between two dates.

Methods

This chapter contains the following topics:

1. addDays
2. addMonths
3. addYears
4. days
5. fromDatetime
6. fromIsoString
7. months
8. now
9. of
10. toIsoString
11. years

202.1 addDays

function addDays(value: date, days: decimal): date

Adds days to a date.

Parameters:

value: the input date

months: the number of days to be added. Must be an integer.

Return:

the new date or null if any parameter is null.

Can be used in:

202.2 addMonths

function addMonths(value: date, months: decimal): datetime

Adds months to a date.

Parameters:

value: the input date

months: the number of months to be added. Must be an integer.

Return:

the new date or null if any parameter is null.

Can be used in:

202.3 addYears

function addYears(value: date, years: decimal): date

Adds years to a date.

Parameters:

value: the input date

years: the number of years to be added. Must be an integer.

Return:

the new date or null if any parameter is null.

Can be used in:

202.4 days

function days(startValue: date, endValue: date): decimal

Returns the number of days between two dates.

Parameters:

startValue: the input start date

endValue: the input end date

Return:

the number of days or null if any parameter is null.

Can be used in:

202.5 fromDatetime

function fromDatetime(value: datetime): date

Converts a date and time to a date. The input time part is ignored.

Parameters:

value: the input date and time

Return:

the new date or null if parameter is null.

Can be used in:

202.6 fromIsoString

function fromIsoString(value: string): date

Parses a date from a string in ISO format (yyyy-MM-dd)

Parameters:

value: the input string

Return:

the date or null if value is null or format is not valid.

Can be used in:

202.7 **months**

function months(startValue: date, endValue: date): decimal

Returns the number of months between two dates.

Parameters:

startValue: the input start date

endValue: the input end date

Return:

the number of months or null if any parameter is null.

Can be used in:

202.8 **now**

function now(): date

Returns current date.

Return:

the current date.

Can be used in:

202.9 **of**

function dateOf(year: decimal, month: decimal, day: decimal): date

Creates a date given year, month and day.

Parameters:

year: the input year

month: the input month

day: the input day

Return:

the new date or null if any parameter is null.

Can be used in:

202.10 **toIsoString**

function toIsoString(value: date): string

Converts a date to a string in ISO format (yyyy-MM-dd)

Parameters:

value: the input date

Return:

the string or null if value is null.

Can be used in:

202.11 **years**

function years(startValue: date, endValue: date): decimal

Returns the number of full years between two dates.

Parameters:

startValue: the input start date

endValue: the input end date

Return:

the number of years or null if any parameter is null.

Can be used in:

CHAPTER 203

Unit core.datetime

Script unit that provides datetime functions.

Methods
function addDays(value: datetime, days: decimal): datetime [p 1427] Adds days to a date and time.
function addHours(value: datetime, hours: decimal): datetime [p 1428] Adds hours to a date and time.
function addMinutes(value: datetime, minutes: decimal): datetime [p 1428] Adds minutes to a date and time.
function addMonths(value: datetime, months: decimal): datetime [p 1428] Adds months to a date and time.
function addSeconds(value: datetime, seconds: decimal): datetime [p 1428] Adds seconds to a date and time.
function addYears(value: datetime, years: decimal): datetime [p 1429] Adds years to a date and time.
function days(startValue: datetime, endValue: datetime): decimal [p 1429] Returns the number of days between two date and times.
function fromDate(value: date): datetime [p 1429] Converts a date to date and time with time set to 00:00:00.
function fromDateAndTime(dateValue: date, timeValue: time): datetime [p 1430] Creates a date and time given date and time values.
function fromIsoString(value: string): date [p 1430] Parses a date and time from a string in ISO format (yyyy-MM-dd'T'HH:mm:ss.
function months(startValue: date, endValue: date): decimal [p 1430] Returns the number of months between two date and times.

Methods
function now(): datetime [p 1430] Returns current date and time.
function of(year: decimal, month: decimal, day: decimal, hour: decimal, minute: decimal, second: decimal): datetime [p 1431] Creates a date and time given year, month, day, hours, minutes and seconds.
function seconds(startValue: datetime, endValue: datetime): decimal [p 1431] Returns the number of seconds between two date and times.
function toIsoString(value: date): string [p 1431] Converts a date and time to a string in ISO format (yyyy-MM-dd'T'HH:mm:ss.
function years(startValue: date, endValue: date): decimal [p 1432] Returns the number of full years between two date and times.

This chapter contains the following topics:

1. addDays
2. addHours
3. addMinutes
4. addMonths
5. addSeconds
6. addYears
7. days
8. fromDate
9. fromDateAndTime
10. fromIsoString
11. months
12. now
13. of
14. seconds
15. toIsoString
16. years

203.1 addDays

function addDays(value: datetime, days: decimal): datetime

Adds days to a date and time.

Parameters:

value: the input date and time

months: the number of days to be added. Must be an integer.

Return:

the new date and time or null if any parameter is null.

Can be used in:

203.2 addHours

function addHours(value: datetime, hours: decimal): datetime

Adds hours to a date and time.

Parameters:

value: the input date and time

hours: the number of hours to be added. Must be an integer.

Return:

the new date and time or null if any parameter is null.

Can be used in:

203.3 addMinutes

function addMinutes(value: datetime, minutes: decimal): datetime

Adds minutes to a date and time.

Parameters:

value: the input date and time

minutes: the number of minutes to be added. Must be an integer.

Return:

the new date and time or null if any parameter is null.

Can be used in:

203.4 addMonths

function addMonths(value: datetime, months: decimal): datetime

Adds months to a date and time.

Parameters:

value: the input date and time

months: the number of months to be added. Must be an integer.

Return:

the new date and time or null if any parameter is null.

Can be used in:

203.5 addSeconds

function addSeconds(value: datetime, seconds: decimal): datetime

Adds seconds to a date and time. Fractions of seconds are supported with millisecond precision.

Parameters:

value: the input date and time

seconds: the number of seconds to be added. Fractions are rounded to three decimal digits.

Return:

the new date and time or null if any parameter is null.

Can be used in:

203.6 addYears

function addYears(value: datetime, years: decimal): datetime

Adds years to a date and time.

Parameters:

value: the input date and time

months: the number of years to be added. Must be an integer.

Return:

the new date and time or null if any parameter is null.

Can be used in:

203.7 days

function days(startValue: datetime, endValue: datetime): decimal

Returns the number of days between two date and times. This function is equivalent to days(toDate(startValue), toDate(endValue)).

Parameters:

startValue: the input start date and time

endValue: the input end date and time

Return:

the number of days or null if any parameter is null.

Can be used in:

203.8 fromDate

function fromDate(value: date): datetime

Converts a date to date and time with time set to 00:00:00.

Parameters:

value: the input date

Return:

the new date and time or null if parameter is null.

Can be used in:

203.9 fromDateAndTime

function fromDateAndTime(dateValue: date, timeValue: time): datetime

Creates a date and time given date and time values.

Parameters:

dateValue: the input date

timeValue: the input time

Return:

the new date and time or null if any parameter is null.

Can be used in:

203.10 fromIsoString

function fromIsoString(value: string): date

Parses a date and time from a string in ISO format (yyyy-MM-dd'T'HH:mm:ss.SSS). The millisecond part is optional.

Parameters:

value: the input string

Return:

the date or null if value is null or format is not valid.

Can be used in:

203.11 months

function months(startValue: date, endValue: date): decimal

Returns the number of months between two date and times.

Parameters:

startValue: the input start date and time

endValue: the input end date and time

Return:

the number of months or null if any parameter is null.

Can be used in:

203.12 now

function now(): datetime

Returns current date and time.

Return:

the current date and time.

Can be used in:

203.13 **of**

function of(year: decimal, month: decimal, day: decimal, hour: decimal, minute: decimal, second: decimal): datetime

Creates a date and time given year, month, day, hours, minutes and seconds. Fractions of seconds are supported with millisecond precision.

Parameters:

year: the input year

month: the input month

day: the input day

hour: the input hour

minute: the input minute

second: the input second. Fractions are rounded to three decimal digits.

Return:

the new date and time or null if any parameter is null.

Can be used in:

203.14 **seconds**

function seconds(startValue: datetime, endValue: datetime): decimal

Returns the number of seconds between two date and times. Fractions of seconds are supported with millisecond precision.

Parameters:

startValue: the input start date and time inclusive

endValue: the input end date and time exclusive

Return:

the number of seconds or null if any parameter is null.

Can be used in:

203.15 **toIsoString**

function toIsoString(value: date): string

Converts a date and time to a string in ISO format (yyyy-MM-dd'T'HH:mm:ss.SSS)

Parameters:

value: the input date and time

Return:

the string or null if value is null.

Can be used in:

203.16 **years**

function years(startValue: date, endValue: date): decimal

Returns the number of full years between two date and times.

Parameters:

startValue: the input start date and time

endValue: the input end date and time

Return:

the number of years or null if any parameter is null.

Can be used in:

CHAPTER 204

Unit core.dictionary

Script unit that provides methods to create and manipulate dictionaries. A dictionary is a map with string keys and values of a specified type.

Methods
procedure clear(dict: dict) [p 1435] Removes all of the mappings from the dictionary.
function clone(dict: dict): dict [p 1436] Returns a shallow clone of the dictionary.
function containsKey(dict: dict, key: string): boolean [p 1436] Returns true if the dictionary contains a mapping for the specified key.
function containsValue(dict: dict, value: T): boolean [p 1436] Returns true if the dictionary maps one or more keys to the specified value.
function equals(dict1: dict, dict2: dict): boolean [p 1436] Compares two dictionaries for equality.
function get(dict: dict, key: string): T [p 1437] Returns the value to which the specified key is mapped, or null if the dictionary contains no mapping for the key.
function getOrDefault(dict: dict, key: string, defaultValue: T): T [p 1437] Returns the value to which the specified key is mapped, or the default value if the dictionary contains no mapping for the key.
function isEmpty(dict: dict): boolean [p 1437] Returns true if the dictionary is null or empty.
function keys(dict: dict): set [p 1438] Returns a set view of the keys contained in the dictionary.
function of(key1: string, value1: T [, key2: string, value2: T]...): dict [p 1438] Creates a dictionary with the specified key-value pairs.
procedure put(dict: dict, key: string, value: T) [p 1438]

Methods
Associates the specified value with the specified key in the dictionary.
procedure putAll(dest: dict, src: dict) [p 1438] Copies all of the mappings from the source dictionary to the destination dictionary.
function remove(dict: dict, key: string): T [p 1439] Removes the mapping for a key from the dictionary if it is present.
function size(dict: dict): integer [p 1439] Returns the number of key-value mappings in the dictionary.
function values(dict: dict): collection [p 1439] Returns a collection view of the values contained in the dictionary.

This chapter contains the following topics:

1. clear
2. clone
3. containsKey
4. containsValue
5. equals
6. get
7. getOrDefault
8. isEmpty
9. keys
10. of
11. put
12. putAll
13. remove
14. size
15. values

204.1 clear

procedure clear(dict: dict)

Removes all of the mappings from the dictionary.

Parameters:

dict: the dictionary to be cleared.

Can be used in:

204.2 clone

function clone(dict: dict): dict

Returns a shallow clone of the dictionary.

Parameters:

dict: the dictionary to clone.

Return:

a new dictionary with the same entries as the input dictionary.

Can be used in:

204.3 containsKey

function containsKey(dict: dict, key: string): boolean

Returns true if the dictionary contains a mapping for the specified key.

Parameters:

dict: the dictionary.

key: the key whose presence in the dictionary is to be tested.

Return:

true if the dictionary contains a mapping for the specified key, false otherwise. Returns null if the dictionary is null.

Can be used in:

204.4 containsValue

function containsValue(dict: dict, value: T): boolean

Returns true if the dictionary maps one or more keys to the specified value.

Parameters:

dict: the dictionary.

value: the value whose presence in the dictionary is to be tested.

Return:

true if the dictionary maps one or more keys to the specified value, false otherwise. Returns null if the dictionary is null.

Can be used in:

204.5 equals

function equals(dict1: dict, dict2: dict): boolean

Compares two dictionaries for equality.

Parameters:

dict1: the first dictionary.

dict2: the second dictionary.

Return:

true if both dictionaries have the same keys and values, false otherwise.

Can be used in:

204.6 get

function get(dict: dict, key: string): T

Returns the value to which the specified key is mapped, or null if the dictionary contains no mapping for the key.

Parameters:

dict: the dictionary.

key: the key whose associated value is to be returned.

Return:

the value to which the specified key is mapped, or null if the dictionary contains no mapping for the key. Returns null if the dictionary is null.

Can be used in:

204.7 getOrDefault

function getOrDefault(dict: dict, key: string, defaultValue: T): T

Returns the value to which the specified key is mapped, or the default value if the dictionary contains no mapping for the key.

Parameters:

dict: the dictionary.

key: the key whose associated value is to be returned.

defaultValue: the default value to return if the key is not found.

Return:

the value to which the specified key is mapped, or the default value if the dictionary contains no mapping for the key. Returns the default value if the dictionary is null.

Can be used in:

204.8 isEmpty

function isEmpty(dict: dict): boolean

Returns true if the dictionary is null or empty.

Parameters:

dict: the input dictionary.

Return:

true if the dictionary is null or empty, false otherwise.

Can be used in:

204.9 keys

function keys(dict: dict): set

Returns a set view of the keys contained in the dictionary.

Parameters:

dict: the dictionary.

Return:

a set view of the keys contained in the dictionary. Returns an empty set if the dictionary is null.

Can be used in:

204.10 of

function of(key1: string, value1: T [, key2: string, value2: T]...): dict

Creates a dictionary with the specified key-value pairs.

Parameters:

keysAndValues: an alternating sequence of keys (strings) and values (of type T).

Return:

a new dictionary containing the specified key-value pairs.

Can be used in:

204.11 put

procedure put(dict: dict, key: string, value: T)

Associates the specified value with the specified key in the dictionary.

Parameters:

dict: the dictionary.

key: the key with which the specified value is to be associated.

value: the value to be associated with the specified key.

Can be used in:

204.12 putAll

procedure putAll(dest: dict, src: dict)

Copies all of the mappings from the source dictionary to the destination dictionary.

Parameters:

dest: the destination dictionary.

src: the source dictionary.

Can be used in:

204.13 **remove**

function remove(dict: dict, key: string): T

Removes the mapping for a key from the dictionary if it is present.

Parameters:

dict: the dictionary.

key: the key whose mapping is to be removed from the dictionary.

Return:

the previous value associated with the key, or null if there was no mapping for the key.

Can be used in:

204.14 **size**

function size(dict: dict): integer

Returns the number of key-value mappings in the dictionary.

Parameters:

dict: the input dictionary.

Return:

the number of key-value mappings in the dictionary, or 0 if the dictionary is null.

Can be used in:

204.15 **values**

function values(dict: dict): collection

Returns a collection view of the values contained in the dictionary.

Parameters:

dict: the dictionary.

Return:

a collection view of the values contained in the dictionary. Returns an empty collection if the dictionary is null.

Can be used in:

CHAPTER 205

Unit core.http

Script unit that provides methods to call an external REST service.

Methods
procedure addParameter(request: http_request, parameter: string, value: string) [p 1444] Adds a query parameter to the request.
procedure addPath(request: http_request, path: string) [p 1445] Adds a path to the url of the request.
procedure addSegment(request: http_request, segment: string) [p 1445] Adds a segment to the url of the request.
function createRequest(id: string): http_request [p 1445] Creates a request specified by the configuration id.
function delete(request: http_request): http_response [p 1446] Sends a DELETE request and handle the response.
function get(request: http_request): http_response [p 1446] Sends a GET request and handle the response.
function getConfigNames(request: http_request): list<string> [p 1446] Gets the key names specified in the rest configuration.
function getConfigValue(request: http_request, name: string): string [p 1447] Gets the value of a given key name specified in the rest configuration.
function getHeaderFirstValue(response: http_response, headerName: string): string [p 1447] Returns the first response header value of the given named header.
function getHeaderNames(response: http_response): list<string> [p 1448] Returns the response header names.
function getHeaderValues(response: http_response, headerName: string): list<string> [p 1448] Returns the response header values of the given named header.

Methods
function getJsonBody(response: http_response): json_value [p 1449] Returns a JSON value of the body response.
function getRequestUrl(request: http_request): string [p 1449] Gets the url of the request.
function getResponseUrl(response: http_response): string [p 1449] Gets the url of the response.
function getStatusCode(response: http_response): int [p 1450] Returns the status code of the response.
function getStringBody(response: http_response): string [p 1450] Returns a string body of the response.
function hasBody(response: http_response): boolean [p 1451] Returns true if the response has a body.
function post(request: http_request, body: string): http_response [p 1451] Sends a POST request by setting the given body and handle the response.
function put(request: http_request, body: string): http_response [p 1452] Sends a PUT request by setting the given body and handle the response.
procedure setHeader(request: http_request, headerField: string, value: string) [p 1452] Sets a value to the header field of the request.
procedure setTimeout(request: http_request,timeout: int) [p 1452] Sets a timeout to the request in milliseconds.

This chapter contains the following topics:

1. addParameter
2. addPath
3. addSegment
4. createRequest
5. delete
6. get
7. getConfigNames
8. getConfigValue
9. getHeaderFirstValue
10. getHeaderNames
11. getHeaderValues
12. getJsonBody
13. getRequestUrl
14. getResponseUrl
15. getStatusCode
16. getStringBody
17. hasBody
18. post
19. put
20. setHeader
21. setTimeout

205.1 addParameter

procedure addParameter(request: http_request, parameter: string, value: string)

Adds a query parameter to the request. The parameter value is percentage encoded.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'book');
  client.addParameter(request,'status','sold');
  ...
end
```

Parameters:

request: the request.

parameter: the query parameter.

value: the parameter value.

Can be used in:

205.2 addPath

procedure addPath(request: http_request, path: string)

Adds a path to the url of the request. Note that this method does not encode the given path.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'book/findByStatus');
  ...
end
```

Parameters:

request: the request.

path: the path to be added.

Can be used in:

205.3 addSegment

procedure addSegment(request: http_request, segment: string)

Adds a segment to the url of the request. Note that the given segment is percentage encoded.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.addSegment(request,'search by term');
  ...
end
```

Parameters:

request: the request.

segment: the segment to be added.

Can be used in:

205.4 createRequest

function createRequest(id: string): http_request

Creates a request specified by the configuration id. Raises an error if the configuration id is not found. The id is specified in the configuration table of the EBX administration.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  ...
end
```

Parameters:

id: the id of the configuration.

Return:

the request.

Can be used in:

205.5 delete

function delete(request: http_request): http_response

Sends a DELETE request and handle the response. Raises an error if an I/O error occurs or the request times out. The returned response has status, headers and maybe a body.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'book');
  client.addPath(request,'B5253');
  var response := client.delete(request);
  ...
end
```

Parameters:

request: the request.

Return:

the received response.

Can be used in:

205.6 get

function get(request: http_request): http_response

Sends a GET request and handle the response. Raises an error if an I/O error occurs or the request times out. The returned response has status, headers and a body.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.setHeader(request,'Content-Type','application/json');
  var response := client.get(request);
  ...
end
```

Parameters:

request: the request.

Return:

the received response.

Can be used in:

205.7 getConfigNames

function getConfigNames(request: http_request): list<string>

Gets the key names specified in the rest configuration. Returns an empty list if no key name is found.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  var keyNames := client.getConfigNames(request);
  var keyValue := client.getConfigValue(request,keyNames[0]);
  ...
end
```

Parameters:

request: the request.

Return:

the string value.

Can be used in:

205.8 getConfigValue

function getConfigValue(request: http_request, name: string): string

Gets the value of a given key name specified in the rest configuration. The value of a secret key is always decrypted. Returns null if the key name is not found. Raises an error if the value of the given key can not be decrypted.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  var accessKey := client.getConfigValue(request,'EBX');
  if not isNull(accessKey) then
  begin
    client.setHeader(request,'Authorization','EBX '| accessKey);
    ...
  end
end
```

Parameters:

request: the request.

Return:

the string value.

Can be used in:

205.9 getHeaderFirstValue

function getHeaderFirstValue(response: http_response, headerName: string): string

Returns the first response header value of the given named header. Returns null if the header name is not found.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  var response := client.get(request);
  var encoding := client.getHeaderFirstValue(response,'content-encoding');
  ...
end
```

Parameters:

response: the response.

headerName: the header name.

Return:

the header values.

Can be used in:

205.10 getHeaderNames

function getHeaderNames(response: http_response): list<string>

Returns the response header names. Returns an empty list if no named header is found.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  var response := client.get(request);
  var headerNames := client.getHeaderNames(response);
  ...
end
```

Parameters:

response: the response.

Return:

a list of the header names.

Can be used in:

205.11 getHeaderValues

function getHeaderValues(response: http_response, headerName: string): list<string>

Returns the response header values of the given named header. Returns an empty list if the name is not found.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  var response := client.get(request);
  var allowedMethod := client.getHeaderValues(response,'access-control-allow-methods');
  ...
end
```

Parameters:

response: the response.

headerName: the header name.

Return:

a list of the header values.

Can be used in:

205.12 getJsonBody

function getJsonBody(response: http_response): json_value

Returns a JSON value of the body response. Returns null if the response has no body.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  var response := client.get(request);
  if (client.hasBody(response)) then
  begin
    var body:=client.getJsonBody(response).asObject;
    var id := json.getString(body, 'id');
    ...
  end
end
```

Parameters:

response: the response.

Return:

the JSON value of the body response. If the body of the response is not valid JSON or is null, returns null.

Can be used in:

205.13 getRequestUrl

function getRequestUrl(request: http_request): string

Gets the url of the request.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'author');
  var url:=client.getRequestUrl(response);
end
```

Parameters:

request: the request.

Return:

the url.

Can be used in:

205.14 getResponseUrl

function getResponseUrl(response: http_response): string

Gets the url of the response.

Example:

```
uses core.http as client;
...
begin
```

```
  var request := client.createRequest('ebx-1');
  client.addPath(request,'publisher');
  var response := client.get(request);
  var url:=client.getResponseUrl(response);
end
```

Parameters:

response: the response.

Return:

the url.

Can be used in:

205.15 getStatusCode

function getStatusCode(response: http_response): int

Returns the status code of the response.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'book');
  var response := client.get(request);
  var status:=client.getStatusCode(response);
  if (status = 200) then
  begin
    ...
  end
end
```

Parameters:

response: the response.

Return:

the status code.

Can be used in:

205.16 getStringBody

function getStringBody(response: http_response): string

Returns a string body of the response. Returns null if the response has no body.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  var response := client.get(request);
  if (client.hasBody(response)) then
  begin
    var body:=client.getStringBody(response);
    ...
  end
end
```

Parameters:

response: the response.

Return:

the string body of the response.

Can be used in:

205.17 **hasBody**

function hasBody(response: http_response): boolean

Returns true if the response has a body.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  var response := client.get(request);
  if (client.hasBody(response)) then
  begin
    ...
  end
end
```

Parameters:

response: the response.

Return:

true if the response body or else false.

Can be used in:

205.18 **post**

function post(request: http_request, body: string): http_response

Sends a POST request by setting the given body and handle the response. Raises an error if an I/O error occurs or the request times out. The returned response has status, headers and maybe a body.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'book');
  var body:='{  "content": {  "id": "B5253",
  "title":  "Da Vinci Code"
  }
  }';
  var response := client.post(request,body);
  ...
end
```

Parameters:

request: the request.

request: the body.

Return:

the received response.

Can be used in:

205.19 **put**

function put(request: http_request, body: string): http_response

Sends a PUT request by setting the given body and handle the response. Raises an error if an I/O error occurs or the request times out. The returned response has status, headers and maybe a body.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.addPath(request,'book');
  client.addPath(request,'B5253');
  var body:='{  "content": {
  "author":  "Dan Brown"
  }
  }';
  var response := client.put(request,body);
  ...
end
```

Parameters:

request: the request.

body: the body.

Return:

the received response.

Can be used in:

205.20 **setHeader**

procedure setHeader(request: http_request, headerField: string, value: string)

Sets a value to the header field of the request.

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.setHeader(request,'Content-Type','application/json');
  ...
end
```

Parameters:

request: the request.

headerField: the header field.

value: the value.

Can be used in:

205.21 **setTimeout**

procedure setTimeout(request: http_request,timeout: int)

Sets a timeout to the request in milliseconds. It must be less than the default value defined in ebx properties file otherwise it is ignored. The default value of the timeout is different according to the context (trigger or workflow).

Example:

```
uses core.http as client;
...
begin
  var request := client.createRequest('ebx-1');
  client.setTimeout(request,900);
  ...
end
```

Parameters:

request: the request.

timeout: the timeout.

Can be used in:

CHAPTER **206**

Unit core.json

Script unit that provides methods to manage json data.

This unit provides methods to create and parse any kind of JSON including string, decimal, boolean, null, objects and arrays. Items of an array do not need to all be of the same type.

A variable of type json_value can represent a json value of any kind. The following readonly boolean properties are never null and can be used to check the type of the json value:

- isNull: boolean - true if value is null

- isString: boolean - the value as a string

- isDecimal: boolean - the value as a decimal

- isBoolean: boolean - the value as a boolean

- isObject: boolean - the value as a object

- isArray: boolean - the value as a array

The following readonly properties can be used to get the json value. They return null if the value not of the given type or is null:

- asDecimal: string - the value as a string

- asBoolean: boolean - the value as a boolean

- asObject: json_object - the value as an object

- asArray: list<json_value> - the value as an array.

Datetime, date and time types are not natively supported in JSON. They can be converted from or to a string using functions toIsoString() and fromIsoString() provided by units core.datetime, core.date and core.time.

Samples The following is a JSON sample:

```
{
"id": "jsmith",
"firstName": "Jane",
"lastName": "Smith",
"age": 30,
"income": null,
"address": {
"street": "123 Main St",
"city": "San Francisco"
},
"variousItems": [
{
"id": "jdoe",
"firstName": "John",
"lastName": "Doe"
},
"Hello!",
-743.6534,
```

```
true,
null,
[54, 67, 89, -21]
]
}
```

The following script shows how to parse this JSON string:

```
uses core.json as json;

procedure processJson(json: string)
begin
  // Parse the JSON string into a JSON object
  // Note: Following code will NOT raise an error if json.parse(json) returns null, but variable user will simply
 be
  // set to null. For details see EBX Script reference.
  var user := json.parse(json).asObject;
  if isNull(user) then
  // Following will be called if:
  // - The json.parse() returned null because JSON is not valid,
  // - The JSON is not an object (e.g. an array).
  raise 1, 'Json parsing failed';

  // Note: Following code will NOT raise an error if the key does not exist in the JSON string, but variable id
  // will be set to null. For details see EBX Script reference.
  var id := json.get(user, 'id').asString;
  var firstName := json.get(user, 'firstName').asString;
  var lastName := json.get(user, 'lastName').asString;
  var age := json.get(user, 'age').asDecimal;
  var income := json.get(user, 'income').asDecimal;
  var address := json.get(user, 'address').asObject;
  var street := json.get(address, 'street').asString;
  var city := json.get(address, 'city').asString;
  // Field "unknown" does not exist in the JSON string, but will not raise an error, and value
  // will be set to null.
  var unknown := json.get(address, 'unknown').asString;

  // Variable income and unknown are null. The following checks if the JSON values were explicitly
  // set to null or if they do not exist in the JSON string.
  if json.get(user, 'income').isNull then
  reportMessage('Property income is explicitly set to null in the JSON string');

  if isNull(json.get(user, 'unknown')) then
  reportMessage('Property unknown is missing in the JSON string');

  // Iterate through the  JSON array of various items.
  // A JSON array is of type list<json_value>.
  for item in json.get(user, 'variousItems').asArray do
  begin
    if item.isString then
    reportMessage('Found string item ' | item.asString);
    else if item.isBoolean  then
    reportMessage('Found boolean item ' | item.asBoolean);
    else if item.isDecimal then
    reportMessage('Found decimal item ' | item.asDecimal);
    else if item.isNull then
    reportMessage('Found null item');
    else if item.isObject then
    reportMessage('Found object item with the number of keys equal to ' | json.keys(item.asObject).size);
    else if item.isArray then
    reportMessage('Found array item of size ' | item.asArray.size);
  end

  // Do something with the parsed data.
end
```

The following script shows how to create a JSON string identical to the above sample:

```
uses core.json as json;
uses core.list as list;

function createJson(): string
begin
  var user := instanceOf<json_object>();

  // Create simple properties.
  json.put(user, 'id', json.ofString('jsmith'));
  json.put(user, 'firstName', json.ofString('Jane'));
  json.put(user, 'lastName', json.ofString('Smith'));
  json.put(user, 'age', json.ofDecimal(30));
  json.put(user, 'income', json.ofNull());

  // Create an objecty property.
```

```
  begin
    var address := instanceOf<json_object>();
    json.put(user, 'address', json.ofObject(address));
    json.put(address, 'street', json.ofString('123 Main St'));
    json.put(address, 'city', json.ofString('San Francisco'));
  end

  // Create a property that is an array of items of various types.
  begin
    // The JSON array variousItems is a list if items of type json_value.
    var variousItems := list.of<json_value>();
    json.put(user, 'variousItems', json.ofArray(variousItems));

    // Add an object to the JSON array.
    begin
      var object := instanceOf<json_object>();
      list.add(variousItems, json.ofObject(object));
      json.put(object, 'id', json.ofString('jdoe'));
      json.put(object, 'firstName', json.ofString('John'));
      json.put(object, 'lastName', json.ofString('Doe'));
    end

    // Add simple values to the JSON array.
    list.add(variousItems, json.ofString('Hello!'));
    list.add(variousItems, json.ofDecimal( - 743.6534));
    list.add(variousItems, json.ofBoolean(true));
    list.add(variousItems, json.ofNull());

    // Add an array of integers.
    begin
      var intArray := list.of<json_value>();
      list.add(variousItems, json.ofArray(intArray));
      for value in list.of(54, 67, 89, - 21) do
      begin
        list.add(intArray, json.ofDecimal(value));
      end
    end
  end

  // Generate the JSON string.
  return json.toJson(json.ofObject(user));
end
```

Methods
function arrayToJoson(list: list): string [p 1462] This serializes a list of JSON values to a string.
function copyOfBooleans(value: list): json_value [p 1462] Converts a boolean list to a JSON value of type array.
function copyOfDecimals(value: list): json_value [p 1462] Converts a list of decimals to a JSON value of type array.
function copyOfObjects(value: list): json_value [p 1462] Converts a list of JSON objects to a JSON value of type array.
function copyOfStrings(value: list): json_value [p 1463] Converts a list of strings to a JSON value of type array.
function exists(json: json_object, key: string): boolean [p 1463] Checks if a key exists in the JSON object.
function get(json: json_object, key: string): json_value [p 1463] Returns the JSON value associated with the key in the JSON object.
function getArray(json: json_object, key: string): list [p 1463] This method is a shortcut for core.
function getBoolean(json: json_object, key: string): decimal [p 1463] This method is a shortcut for core.
function getDecimal(json: json_object, key: string): decimal [p 1463] This method is a shortcut for core.
function getObject(json: json_object, key: string): object [p 1464] This method is a shortcut for core.
function getString(json: json_object, key: string): string [p 1464] This method is a shortcut for core.
function isNotNull(json: json_object, key: string): boolean [p 1464] This method is a shortcut for not core.
function isNull(json: json_object, key: string): boolean [p 1464] Returns true if the JSON value associated with the key exists and a JSON null value.
function keys(object: json_object): list [p 1464] Returns the keys of the JSON object as a list of strings.

Methods
function objectToJson(object: json_object): string [p 1464] This serializes a JSON object to a string.
function ofArray(list: list): json_value [p 1465] Returns an array JSON value.
function ofBoolean(value: boolean): json_value [p 1465] Returns a boolean JSON value.
function ofDecimal(value: decimal): json_value [p 1465] Returns a decimal JSON value.
function ofNull(): json_value [p 1465] Returns a null JSON value.
function ofObject(value: json_object): json_value [p 1465] Returns an object JSON value.
function ofString(value: string): json_value [p 1466] Returns a string JSON value.
function parse(json: string): json_value [p 1466] Parses a JSON string.
function parseWithReport(json: string): json_report [p 1466] Parses a JSON string.
procedure put(json: json_object, key: string, value: json_value) [p 1467] Sets the JSON value associated with the key in the JSON object.
procedure putBoolean(json: json_object, key: string, value: list) [p 1467] This method is a shortcut for core.
procedure putBoolean(json: json_object, key: string, value: boolean) [p 1467] This method is a shortcut for core.
procedure putDecimal(json: json_object, key: string, value: decimal) [p 1467] This method is a shortcut for core.
procedure putBoolean(json: json_object, key: string) [p 1467] This method is a shortcut for core.
procedure putBoolean(json: json_object, key: string, value: json_object) [p 1467] This method is a shortcut for core.
procedure putBoolean(json: json_object, key: string, value: string) [p 1467]

Methods
This method is a shortcut for core.
procedure remove(builder: json_object, key: string) [p 1468] Removes the item identified by the key.
function toJson(jsonValue: json_value): string [p 1468] Serializes a JSON value to a string.

This chapter contains the following topics:

1. arrayToJson
2. copyOfBooleans
3. copyOfDecimals
4. copyOfObjects
5. copyOfStrings
6. exists
7. get
8. getArray
9. getBoolean
10. getDecimal
11. getObject
12. getString
13. isNotNull
14. isNull
15. keys
16. objectToJson
17. ofArray
18. ofBoolean
19. ofDecimal
20. ofNull
21. ofObject
22. ofString
23. parse
24. parseWithReport
25. put
26. putArray
27. putBoolean
28. putDecimal
29. putNull
30. putObject
31. putString
32. remove
33. toJson

206.1 arrayToJson

function arrayToJoson(list: list): string

This serializes a list of JSON values to a string.

Return:

the JSON string. Is null if the input list is null.

Can be used in:

206.2 copyOfBooleans

function copyOfBooleans(value: list): json_value

Converts a boolean list to a JSON value of type array. Modifying the input list will not modify the returned JSON value.

Parameters:

list: the input list.

Return:

the JSON value. If input list is null, is the same as ofNull(). Never null.

Can be used in:

206.3 copyOfDecimals

function copyOfDecimals(value: list): json_value

Converts a list of decimals to a JSON value of type array. Modifying the input list will not modify the returned JSON value.

Parameters:

list: the input list.

Return:

the JSON value. If input list is null, is the same as ofNull(). Never null.

Can be used in:

206.4 copyOfObjects

function copyOfObjects(value: list): json_value

Converts a list of JSON objects to a JSON value of type array. Modifying the input list will not modify the returned JSON value. The copy is not deep, so modifying the properties of an object in the list will modify the corresponding JSON value in the returned JSON value array.

Parameters:

list: the input list.

Return:

the JSON value. If input list is null, is the same as ofNull(). Never null.

Can be used in:

206.5 copyOfStrings

function copyOfStrings(value: list): json_value

Converts a list of strings to a JSON value of type array. Modifying the input list will not modify the returned JSON value.

Parameters:

list: the input list.

Return:

the JSON value. If input list is null, is the same as ofNull(). Never null.

Can be used in:

206.6 exists

function exists(json: json_object, key: string): boolean

Checks if a key exists in the JSON object.

Can be used in:

206.7 get

function get(json: json_object, key: string): json_value

Returns the JSON value associated with the key in the JSON object.

Can be used in:

206.8 getArray

function getArray(json: json_object, key: string): list

This method is a shortcut for core.json.get(json, key).asArray.

Can be used in:

206.9 getBoolean

function getBoolean(json: json_object, key: string): decimal

This method is a shortcut for core.json.get(json, key).asBoolean.

Can be used in:

206.10 getDecimal

function getDecimal(json: json_object, key: string): decimal

This method is a shortcut for core.json.get(json, key).asDecimal.

Can be used in:

206.11 getObject

function getObject(json: json_object, key: string): object

This method is a shortcut for core.json.get(json, key).asObject.

Can be used in:

206.12 getString

function getString(json: json_object, key: string): string

This method is a shortcut for core.json.get(json, key).asString.

Can be used in:

206.13 isNotNull

function isNotNull(json: json_object, key: string): boolean

This method is a shortcut for not core.json.isNull(json, key).

Can be used in:

206.14 isNull

function isNull(json: json_object, key: string): boolean

Returns true if the JSON value associated with the key exists and a JSON null value.

Can be used in:

206.15 keys

function keys(object: json_object): list

Returns the keys of the JSON object as a list of strings. The order of the keys is not guaranteed.

Parameters:

object: the JSON object.

Return:

the keys list or null of the object is null.

Can be used in:

206.16 objectToJson

function objectToJson(object: json_object): string

This serializes a JSON object to a string.

Return:

the JSON string. Is null if the input JSON object is null.

Can be used in:

206.17 ofArray

function ofArray(list: list): json_value

Returns an array JSON value. Modifying the input list will modify the returned JSON array.

Parameters:

value: the input value.

Return:

the JSON value. If input list is null, is the same as ofNull(). Never null.

Can be used in:

206.18 ofBoolean

function ofBoolean(value: boolean): json_value

Returns a boolean JSON value.

Parameters:

value: the input value.

Return:

the JSON value. If input value is null, is the same as ofNull(). Never null.

Can be used in:

206.19 ofDecimal

function ofDecimal(value: decimal): json_value

Returns a decimal JSON value.

Parameters:

value: the input value.

Return:

the JSON value. If input value is null, is the same as ofNull(). Never null.

Can be used in:

206.20 ofNull

function ofNull(): json_value

Returns a null JSON value.

Return:

the value that represents the null JSON value. Never null.

Can be used in:

206.21 ofObject

function ofObject(value: json_object): json_value

Returns an object JSON value. Modifying the input object's properties will modify the returned object JSON value's properties.

Parameters:

value: the input value.

Return:

the JSON value. If input value is null, is the same as ofNull(). Never null.

Can be used in:

206.22 ofString

function ofString(value: string): json_value

Returns a string JSON value.

Parameters:

value: the input value.

Return:

the JSON value. If input value is null, is the same as ofNull(). Never null.

Can be used in:

206.23 parse

function parse(json: string): json_value

Parses a JSON string. The JSON can be of any type.

If parsing failed, returns null without any details. If a detailed error message is needed, for example to diagnose an issue, use parseWithReport() instead.

Parameters:

json: the JSON string.

Return:

the JSON value. If the input string is not valid JSON or is null, returns null.

Can be used in:

206.24 parseWithReport

function parseWithReport(json: string): json_report

Parses a JSON string. The JSON can be of any type. If parsing failed, returns an error message that may be helpful to diagnose the issue.

The returned report has following readonly properties: value: json_value - The result of the parsing. Is never null if isSuccess is true or isError is false. isSuccess: boolean - true if and only the parsing succeeded. Never null. isError: boolean - false if and only the parsing failed. Never null. message: string - the error message. May be null if isSuccess is true or isError is false.

Parameters:

json: the JSON string.

Return:

the JSON report. Never null.

Can be used in:

206.25 put

procedure put(json: json_object, key: string, value: json_value)

Sets the JSON value associated with the key in the JSON object. Any previous value associated with the key is overwritten. Raises an error of the JSON object or the key are null.

Can be used in:

206.26 putArray

procedure putBoolean(json: json_object, key: string, value: list)

This method is a shortcut for core.json.put(json, key, core.json.ofArray(value)).

Can be used in:

206.27 putBoolean

procedure putBoolean(json: json_object, key: string, value: boolean)

This method is a shortcut for core.json.put(json, key, core.json.ofBoolean(value)).

Can be used in:

206.28 putDecimal

procedure putDecimal(json: json_object, key: string, value: decimal)

This method is a shortcut for core.json.put(json, key, core.json.ofDecimal(value)).

Can be used in:

206.29 putNull

procedure putBoolean(json: json_object, key: string)

This method is a shortcut for core.json.put(json, key, core.json.ofNull()).

Can be used in:

206.30 putObject

procedure putBoolean(json: json_object, key: string, value: json_object)

This method is a shortcut for core.json.put(json, key, core.json.ofObject(value)).

Can be used in:

206.31 putString

procedure putBoolean(json: json_object, key: string, value: string)

This method is a shortcut for core.json.put(json, key, core.json.ofString(value)).

Can be used in:

206.32 remove

procedure remove(builder: json_object, key: string)

Removes the item identified by the key. Removing a non-existing item has no effect.

Can be used in:

206.33 toJson

function toJson(jsonValue: json_value): string

Serializes a JSON value to a string.

Parameters:

jsonValue: the JSON value.

Return:

the JSON string. Is null if the input JSON value is null. The JSON null value is correctly serialized to valid JSON string 'null'.

Can be used in:

CHAPTER 207

Unit core.list

Script unit that provides methods to create and update lists.

Methods
procedure add(values: list<ItemType>, element: ItemType) [p 1472] Add an element to values list.
procedure addAll(values: list<ItemType>, elements: list<ItemType>): boolean [p 1472] Add all elements to values list.
procedure addFirst(list: list<ItemType>, element: ItemType) [p 1472] adds the given element at the beginning of the list.
procedure clear(list: list) [p 1472] Removes all elements from the list.
function clone(list: list): list [p 1473] Returns a shallow clone of the list.
function contains(values: list<ItemType>, element: ItemType): boolean [p 1473] Returns true if values list contains an element.
function containsAll(values: list<ItemType>, element: ItemType): boolean [p 1473] Returns true if values list contains all elements.
function indexOf(values: list<ItemType>, element: ItemType) [p 1473] Return the index of the first occurrence of element in values list.
function isEmpty(value: list<ItemType>): boolean [p 1474] Returns true if value is null or a zero size list.
function lastIndexOf(values: list<ItemType>, element: ItemType) [p 1474] Return the index of the last occurrence of element in values list.
function of<ItemType>(value: ItemType [, value: ItemType]...): list<ItemType> [p 1474] Creates a list with the elements passed as parameters.

Methods
function remove(values: list<ItemType>, element: ItemType): boolean [p 1475] Remove the first occurrence of an element from values list.
function removeAll(values: list<ItemType>, elements: list<ItemType>): boolean [p 1475] Remove all elements from values list.
function retainAll(values: list<ItemType>, elements: list<ItemType>): boolean [p 1475] Retains only the elements in the values list that are contained in the elements.
procedure reverse(values: list<ItemType>) [p 1475] Reverses the order of elements in values.
procedure reverseSort(values: list<ItemType>) [p 1476] Sorts the specified values list into descending order, according to the natural ordering of its elements.
procedure sort(values: list<ItemType>) [p 1476] Sorts the specified values list into ascending order, according to the natural ordering of its elements.
function subList<ItemType>(values: list<ItemType>, startIndex: decimal, endIndex: decimal): list<ItemType> [p 1476] Returns a new list that contains a subsequence of elements from values list.

This chapter contains the following topics:

1. add
2. addAll
3. addFirst
4. clear
5. clone
6. contains
7. containsAll
8. indexOf
9. isEmpty
10. lastIndexOf
11. of
12. remove
13. removeAll
14. retainAll
15. reverse
16. reverseSort
17. sort
18. subList

207.1 add

procedure add(values: list<ItemType>, element: ItemType)

Add an element to values list.

Parameters:

values: the input list.

element: the input element to add. Its type must be compatible with the type of the list.

Can be used in:

207.2 addAll

procedure addAll(values: list<ItemType>, elements: list<ItemType>): boolean

Add all elements to values list.

Parameters:

values: the input list.

elements: the elements to add. The element type must be compatible with the type of the list.

Return:

true after values list contents was modified by this call else false.

Can be used in:

207.3 addFirst

procedure addFirst(list: list<ItemType>, element: ItemType)

adds the given element at the beginning of the list.

Example:

```
uses core.list as l;
procedure doSomething()
begin
  var list := ["apple", "banana", "orange"];
  l.addFirst(list, "pear"); // ["pear", "apple", "banana", "orange"]
  ...
end
```

Parameters:

list The list to update

element The element to add at the beginning of the list

Can be used in:

207.4 clear

procedure clear(list: list)

Removes all elements from the list.

Parameters:

list: the list to clear.

Can be used in:

207.5 clone

function clone(list: list): list

Returns a shallow clone of the list.

Parameters:

list: the list to clone.

Return:

a new list with the same elements as the input list.

Can be used in:

207.6 contains

function contains(values: list<ItemType>, element: ItemType): boolean

Returns true if values list contains an element.

Parameters:

values: the input list. Contents must be of the same kind then element.

element: the input element to look for. Must be of the same kind than values contents.

Return:

true if value contains the element else false.

Returns null if values is null.

Can be used in:

207.7 containsAll

function containsAll(values: list<ItemType>, element: ItemType): boolean

Returns true if values list contains all elements.

Parameters:

values: the input list. Contents must be of the same kind then element.

elements: the input elements to look for. Must be of the same kind than values.

Return:

true if value contains all elements else false.

Returns null if any parameters is null.

Can be used in:

207.8 indexOf

function indexOf(values: list<ItemType>, element: ItemType)

Return the index of the first occurrence of element in values list.

Parameters:

values: the input list.

element: the input element to look for. Its type must be compatible with the type of the list.

Return:

the index of the first occurrence of element in values list or null if not found.

Can be used in:

207.9 isEmpty

function isEmpty(value: list<ItemType>): boolean

Returns true if value is null or a zero size list.

Parameters:

values: an input list.

Return:

true if value is null or a zero size list, or else false. Never returns null.

Can be used in:

207.10 lastIndexOf

function lastIndexOf(values: list<ItemType>, element: ItemType)

Return the index of the last occurrence of element in values list.

Parameters:

values: the input list.

element: the input element to look for.

Return:

the index of the last occurrence of element in values list or null if not found.

Can be used in:

207.11 of

function of<ItemType>(value: ItemType [, value: ItemType]...): list<ItemType>

Creates a list with the elements passed as parameters. All elements must be of same type.

Example:

```
uses core.list as list;

function getCities(): list<string>
begin
  return list.of('Paris', 'Bruxelles', 'Berlin');
end
```

Function Types:

ItemType: the type of an item of the returned list. Is optional if at least one value is specified.

Parameters:

value: an item to add to the list. All items must be of same type.

Return:

the new list. It is updatable.

Can be used in:

207.12 remove

function remove(values: list<ItemType>, element: ItemType): boolean

Remove the first occurrence of an element from values list.

Parameters:

values: the input list.

element: the input element to add. Its type must be compatible with the type of the list.

Return:

true if values list contents was modified by this call else false.

Can be used in:

207.13 removeAll

function removeAll(values: list<ItemType>, elements: list<ItemType>): boolean

Remove all elements from values list.

Parameters:

values: the input list.

elements: the elements to remove. The element type must be compatible with the type of the list.

Return:

true if values list contents was modified by this call else false.

Can be used in:

207.14 retainAll

function retainAll(values: list<ItemType>, elements: list<ItemType>): boolean

Retains only the elements in the values list that are contained in the elements. In other words, removes from the values list all of its elements that are not contained in the elements.

Parameters:

values: the input list.

elements: the elements to retain. The element type must be compatible with the type of the list.

Return:

true if values list contents was modified by this call else false.

Can be used in:

207.15 reverse

procedure reverse(values: list<ItemType>)

Reverses the order of elements in values. Does nothing if values is null or a zero size list.

Parameters:

values: the input list to reverse.

Can be used in:

207.16 reverseSort

procedure reverseSort(values: list<ItemType>)

Sorts the specified values list into descending order, according to the natural ordering of its elements. Does nothing if values is null or a zero size list.

Parameters:

values: the input list to sort.

Can be used in:

207.17 sort

procedure sort(values: list<ItemType>)

Sorts the specified values list into ascending order, according to the natural ordering of its elements. Does nothing if values is null or a zero size list.

Parameters:

values: the input list to sort

Can be used in:

207.18 subList

function subList<ItemType>(values: list<ItemType>, startIndex: decimal, endIndex: decimal): list<ItemType>

Returns a new list that contains a subsequence of elements from values list.

The subList start from the specified startIndex and extends to the endIndex of the input list.

Function Types:

ItemType: the type of an item of the returned list. It is optional.

Parameters:

values: the input list

startIndex: the input start index. Must be a positive integer or zero.

endIndex: the input end index. Must be a positive integer or zero.

Return:

the new list or null if any parameter is null.

Can be used in:

CHAPTER **208**

Unit core.locale

Script unit that provides functions for managing locales.

Methods
function fromString(value: string): locale [p 1477] Creates a locale from a string.
function of(language: string, country: string): locale [p 1477] Creates a locale.

This chapter contains the following topics:

1. fromString
2. of

208.1 fromString

function fromString(value: string): locale

Creates a locale from a string.

Parameters:

value: A locale, for example 'en-US'.

Return:

the locale.

Can be used in:

208.2 of

function of(language: string, country: string): locale

Creates a locale.

Parameters:

language: An ISO 639 alpha-2 or alpha-3 language code.

country: An ISO 3166 alpha-2 country code or a UN M.49 numeric-3 area code.

Return:

the locale.

Can be used in:

CHAPTER 209

Unit core.log

Script unit that provides methods to log messages.\n\nThis message can be viewed using administration user interface provide by EBX.

Methods
procedure error(message: string) [p 1479] Logs an error message.
procedure info(message: string) [p 1479] Logs an information message.
procedure warn(message: string) [p 1480] Logs a warning message.

This chapter contains the following topics:

1. error
2. info
3. warn

209.1 error

procedure error(message: string)

Logs an error message.

Parameters:

value: message: the message.

Can be used in:

209.2 info

procedure info(message: string)

Logs an information message.

Parameters:

value: message: the message.

Can be used in:

209.3 **warn**

procedure warn(message: string)

Logs a warning message.

Parameters:

value: message: the message.

Can be used in:

CHAPTER 210

Unit core.math

Script unit that provides mathematical functions and constants.

Methods
function abs(value: decimal): decimal [p 1485] Returns the absolute value of a decimal value.
function acos(value: decimal): decimal [p 1485] Returns the arc cosine of a value.
function asin(value: decimal): decimal [p 1485] Returns the arc sine of a value.
function atan(value: decimal): decimal [p 1485] Returns the arc tangent of a value.
function cbrt(value: decimal): decimal [p 1486] Returns the cube root of a decimal value.
function ceil(value: decimal): decimal [p 1486] Returns the integer ceil value of the input value.
function cos(value: decimal): decimal [p 1486] Returns the trigonometric cosine of an angle.
function e(): decimal [p 1487] Returns e value: 2.
function exp(value: decimal): decimal [p 1487] Returns Euler's number *e* raised to the power of a decimal value.
function floor(value: decimal): decimal [p 1487] Returns the integer floor value of the input value.
function log(value: decimal): decimal [p 1487] Returns the natural logarithm (base *e*) of the input value.

Methods
function max(a: decimal, b: decimal): decimal [p 1488] Returns the larger of two decimal values.
function min(a: decimal, b: decimal): decimal [p 1488] Returns the smaller of two decimal values.
function pi(): decimal [p 1488] Returns pi value: 3.
function pow(a: decimal, b: decimal): decimal [p 1489] Returns the value of the first argument raised to the power of the second argument.
function random(): decimal [p 1489] Returns a random decimal value with a positive sign, greater than or equal to 0 and less than 1.
function randomInt(maxValue: decimal): decimal [p 1489] Returns a random integer between 0 (inclusive) and the specified `bound` (exclusive).
function round(value: decimal): decimal [p 1490] Returns the integer round value of the input value.
function roundDown(value: decimal): decimal [p 1490] Returns the integer round down value of the input value.
function roundHalfDown(value: decimal): decimal [p 1490] Returns the integer round half down value of the input value.
function roundHalfEven(value: decimal): decimal [p 1490] Returns the integer round half even value of the input value.
function roundUp(value: decimal): decimal [p 1491] Returns the integer round up value of the input value.
function scaleByPowerOfTen(value: decimal, power: decimal): decimal [p 1491] Returns the value scale by power of ten.
function sin(value: decimal): decimal [p 1491] Returns the trigonometric sine of an angle.
function sqrt(value: decimal): decimal [p 1492] Returns the positive square root of a value.
function tan(value: decimal): decimal [p 1492] Returns the trigonometric tangent of an angle.
function toDegrees(value: decimal): decimal [p 1492]

Methods
Converts an angle measured in degrees to an approximately equivalent angle measured in radians.
function toRadians(value: decimal): decimal [p 1492] Converts an angle measured in radians to an approximately equivalent angle measured in degrees.

This chapter contains the following topics:

1. abs
2. acos
3. asin
4. atan
5. cbrt
6. ceil
7. cos
8. e
9. exp
10. floor
11. log
12. max
13. min
14. pi
15. pow
16. random
17. randomInt
18. round
19. roundDown
20. roundHalfDown
21. roundHalfEven
22. roundUp
23. scaleByPowerOfTen
24. sin
25. sqrt
26. tan
27. toDegrees
28. toRadians

210.1 abs

function abs(value: decimal): decimal

Returns the absolute value of a decimal value.

Parameters:

value: the input value.

Return:

the absolute value of the input value or null if input value is null.

Can be used in:

210.2 acos

function acos(value: decimal): decimal

Returns the arc cosine of a value.

The returned angle is in the range 0.0 through *pi*.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value. Absolute value must not be greater than 1.

Return:

the arc cosine of the input value or null if the input value is null.

Can be used in:

210.3 asin

function asin(value: decimal): decimal

Returns the arc sine of a value.

The returned angle is in the range -*pi*/2 through *pi*/2.

Special cases:

If the input value is zero, then the result is a zero.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value. Absolute value must not be greater than 1.

Return:

the arc sine of the input value or null if the input value is null.

Can be used in:

210.4 atan

function atan(value: decimal): decimal

Returns the arc tangent of a value.

The returned angle is in the range -*pi*/2 through *pi*/2.

Special cases:

If the input value is zero, then the result is a zero.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value.

Return:

the arc tangent of the input value or null if the input value is null.

Can be used in:

210.5 **cbrt**

function cbrt(value: decimal): decimal

Returns the cube root of a decimal value. Calculations use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value.

Return:

the cube root of the input value or null if input value is null.

Can be used in:

210.6 **ceil**

function ceil(value: decimal): decimal

Returns the integer ceil value of the input value.

Parameters:

value: the input value.

Return:

the ceil value of the input value or null if input value is null..

Can be used in:

210.7 **cos**

function cos(value: decimal): decimal

Returns the trigonometric cosine of an angle.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value in radians.

Return:

the cosine of the input value or null if the input value is null.

Can be used in:

210.8 e

function e(): decimal

Returns e value: 2.71828182845904523536028747135266 2

Return:

e value.

Can be used in:

210.9 exp

function exp(value: decimal): decimal

Returns Euler's number *e* raised to the power of a decimal value.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value.

Return:

the value *e^value* where *e* is the base of the natural logarithms.

If the input value is null returns null.

Can be used in:

210.10 floor

function floor(value: decimal): decimal

Returns the integer floor value of the input value.

Parameters:

value: the input value.

Return:

the floor value of the input value or null if input value is null.

Can be used in:

210.11 log

function log(value: decimal): decimal

Returns the natural logarithm (base *e*) of the input value.

Calculations use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value. Must be strictly greater than zero.

Return:

the natural logarithm of the input value a or null if input value is null.

Can be used in:

210.12 max

function max(a: decimal, b: decimal): decimal

Returns the larger of two decimal values.

Example:

```
uses core.math as m;
procedure doSomething()
begin
  var result := m.max(10, 5); // 10
  ...
end
```

Parameters:

a The first input value

b The second input value

Return:

The larger of the two input values

Can be used in:

210.13 min

function min(a: decimal, b: decimal): decimal

Returns the smaller of two decimal values.

Example:

```
uses core.math as m;
procedure doSomething()
begin
  var result := m.min(10, 5); // 5
  ...
end
```

Parameters:

a The first input value

b The second input value

Return:

The smaller of the two input values

Can be used in:

210.14 pi

function pi(): decimal

Returns pi value: 3.14159265358979323846264338327950 3

Return:

pi value.

Can be used in:

210.15 pow

function pow(a: decimal, b: decimal): decimal

Returns the value of the first argument raised to the power of the second argument. The parameter n must be in the range -999999999 through 999999999, inclusive. pow(0, 0) returns 1.

Example:

```
uses core.math as m;
procedure doSomething()
begin
  var result := m.pow(2, 3); // 8
  ...
end
```

Parameters:

a The base input value

b The exponent input value

Return:

The result of a raised to the power of b

Can be used in:

210.16 random

function random(): decimal

Returns a random decimal value with a positive sign, greater than or equal to 0 and less than 1. Attention, this method is not pure.

Example:

```
uses core.math as m;
procedure doSomething()
begin
  var result := m.random(); // 0.8574303492
  ...
end
```

Return:

A decimal value between 0 (inclusive) and 1 (exclusive)

Can be used in:

210.17 randomInt

function randomInt(maxValue: decimal): decimal

Returns a random integer between 0 (inclusive) and the specified `bound` (exclusive). Attention, this method is not pure.

Example:

```
uses core.math as m;
procedure doSomething()
begin
  var result := m.randomInt(10); // a random integer between 0 and 9
```

```
  ...
end
```

Parameters:

bound The upper bound (exclusive), must be positive

Return:

A random integer between 0 (inclusive) and `bound` (exclusive)

Can be used in:

210.18 round

function round(value: decimal): decimal

Returns the integer round value of the input value. This method uses the standard round half up rounding mode.

Parameters:

value: the input value.

Return:

the round half up value of the input value or null if input value is null.

Can be used in:

210.19 roundDown

function roundDown(value: decimal): decimal

Returns the integer round down value of the input value.

Parameters:

value: the input value.

Return:

the round down value of the input value or null if input value is null.

Can be used in:

210.20 roundHalfDown

function roundHalfDown(value: decimal): decimal

Returns the integer round half down value of the input value.

Parameters:

value: the input value.

Return:

the round half down value of the input value or null if input value is null.

Can be used in:

210.21 roundHalfEven

function roundHalfEven(value: decimal): decimal

Returns the integer round half even value of the input value.

Parameters:

value: the input value.

Return:

the round half even value of the input value or null if input value is null.

Can be used in:

210.22 roundUp

function roundUp(value: decimal): decimal

Returns the integer round up value of the input value.

Parameters:

value: the input value.

Return:

the round up value of the input value or null if input value is null.

Can be used in:

210.23 scaleByPowerOfTen

function scaleByPowerOfTen(value: decimal, power: decimal): decimal

Returns the value scale by power of ten.

Parameters:

value: the input value.

power: the power of ten that the value will be scale to to. Must be an integer.

Return:

the input value scaled by the power of ten or null if input value or power is null.

Can be used in:

210.24 sin

function sin(value: decimal): decimal

Returns the trigonometric sine of an angle.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Special cases:

If the input value is zero, then the result is a zero.

Parameters:

value: the input value in radians.

Return:

the sine of the input value or null if the input value is null.

Can be used in:

210.25 **sqrt**

function sqrt(value: decimal): decimal

Returns the positive square root of a value.

Calculations use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value. Must be positive or null.

Return:

the positive square root of the input value or null if input value is null.

Can be used in:

210.26 **tan**

function tan(value: decimal): decimal

Returns the trigonometric tangent of an angle.

Special cases:

If the input value is zero, then the result is a zero.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value in radians.

Return:

the tangent of the input value or null if the input value is null.

Can be used in:

210.27 **toDegrees**

function toDegrees(value: decimal): decimal

Converts an angle measured in degrees to an approximately equivalent angle measured in radians.

The conversion from degrees to radians is generally inexact.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value in radians

Return:

the measurement of the input value in degrees or null if the input value is null.

Can be used in:

210.28 **toRadians**

function toRadians(value: decimal): decimal

Converts an angle measured in radians to an approximately equivalent angle measured in degrees.

The conversion from radians to degrees is generally inexact;

users should *not* expect cos(toRadians(90.0)) to exactly equal 0.0.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters:

value: the input value in degrees

Return:

the measurement of the input value in radians or null if the input value is null.

Can be used in:

CHAPTER 211

Unit core.message

Script unit that provides methods to create and manage user messages

Methods
procedure addError(message: string) [p 1497] Adds the specified error message to the current validation context.
procedure addInfo(message: string) [p 1497] Adds the specified info message to the current validation context.
function addLocale(userMessage: user_message, message: string, locale: locale): user_message [p 1498] Adds a locale to a user message.
procedure addMessage(message: user_message) [p 1498] Adds the specified user message to the current validation context.
procedure addRecordMessage(record: record, message: user_message) [p 1498] Adds the info message to the current validation context for the specified record.
procedure addWarning(message: string) [p 1499] Adds the specified warning message to the current validation context.
procedure addXPathMessage(xPath: string, message: user_message) [p 1499] Adds the user message to the current validation context for the records that match the specified XPath predicate.
function of(message: string, locale: locale): user_message [p 1500] Creates an instance of a user message for a single locale.
function ofBundleError(path: string, key: string, parameters: list<object>): user_message [p 1500] Creates an instance of an error user message from messages bundle.
function ofBundleInfo(path: string, key: string, parameters: list<object>): user_message [p 1500] Creates an instance of an info user message from messages bundle.
function ofBundleWarning(path: string, key: string, parameters: list<object>): user_message [p 1501] Creates an instance of a warning user message from messages bundle.

Methods
function toString(userMessage: user_message, locale: locale): string [p 1501] Formats the message with the specified locale.

This chapter contains the following topics:

1. addError
2. addInfo
3. addLocale
4. addMessage
5. addRecordMessage
6. addWarning
7. addXPathMessage
8. of
9. ofBundleError
10. ofBundleInfo
11. ofBundleWarning
12. toString

211.1 addError

procedure addError(message: string)

Adds the specified error message to the current validation context.

Example:

```
uses core.message as msg;
...
export procedure checkRecord(record : .Project)
begin
  msg.addError('This is an error message');
end
...
```

Parameters:

message: the string message.

Can be used in:

211.2 addInfo

procedure addInfo(message: string)

Adds the specified info message to the current validation context.

Example:

```
uses core.message as msg;
...
export procedure checkRecord(record : .Project)
begin
  msg.addInfo('This is an info message');
end
```

```
...
```

Parameters:

message: the string message.

Can be used in:

211.3 addLocale

function addLocale(userMessage: user_message, message: string, locale: locale): user_message

Adds a locale to a user message.

Example:

```
uses core.message as msg;
export procedure executeScriptTask()
begin
  var fr:=locale.of('fr','FR');
  var en:=locale.of('en','EN');
  var messageEn:=msg.of('A new message',en);
  var message:=msg.addLocale(messageEn,'Un nouveau message',fr);
end
```

Parameters:

userMessage: the user message.

message: the string message in the locale"s language. If null, the empty string is used.

locale: the locale. if null, the default locale is used.

Return:

a new message with the added locale. If the input userMessage is null, creates a new user message with only the specified locale.

Can be used in:

211.4 addMessage

procedure addMessage(message: user_message)

Adds the specified user message to the current validation context.

Example:

```
uses core.message as msg;
...
export procedure checkRecord(record : .Project)
begin
  var userMessage:=msg.ofBundleError('Key.invalid.value', 'Bundle/path');
  msg.addMessage(userMessage);
end
...
```

Parameters:

message: the user message.

Can be used in:

211.5 addRecordMessage

procedure addRecordMessage(record: record, message: user_message)

Adds the info message to the current validation context for the specified record.

Example:

```
uses core.message as msg;
...
export procedure checkTable()
begin
  var xpath :='/root/table[./id=15]';
  var record := d.lookupRecordByFullXPath(table, xpath);
  var userMessage:=msg.ofBundleInfo('Key.key2', 'Bundle/path');
  msg.addRecordMessage(record,userMessage);
end
...
```

Parameters:

record: the record.

message: the user message.

Can be used in:

211.6 addWarning

procedure addWarning(message: string)

Adds the specified warning message to the current validation context.

Example:

```
uses core.message as msg;
...
export procedure checkRecord(record : .Project)
begin
  msg.addWarning('This is a warning message');
end
...
```

Parameters:

message: the string message.

Can be used in:

211.7 addXPathMessage

procedure addXPathMessage(xPath: string, message: user_message)

Adds the user message to the current validation context for the records that match the specified XPath predicate.

Example:

```
uses core.message as msg;
...
export procedure checkTable()
begin
  var userMessage:=msg.ofBundleError('key.key1', 'Bundle/path');
  msg.addXPathMessage('./number<100',userMessage);
end
...
```

Parameters:

xPath: the XPath predicate of the records to which the message applies.

message: the user message.

Can be used in:

211.8 **of**

function of(message: string, locale: locale): user_message

Creates an instance of a user message for a single locale.

Example:

```
uses core.message as msg;
export procedure executeScriptTask()
begin
  var en:=locale.of('en','EN');
  var message:=msg.of('A new message',en);
end
```

Parameters:

message: the string message in the locale"s language. If null, the empty string is used.

locale: the locale. if null, the default locale is used.

Return:

the new user message.

Can be used in:

211.9 **ofBundleError**

function ofBundleError(path: string, key: string, parameters: list<object>): user_message

Creates an instance of an error user message from messages bundle.

The user message from the bundle can be created by specifying a path and a key in the EBX IDE.

Example:

```
uses core.message as msg;
...
export procedure check()
begin
  // the error message from the bundle: 'The reference {0} must start with "TIBCO"'
  // the reference id is passed as parameter to the message
  var message:= message.ofBundleError('bundle/path','reference.starts.tibco',_ebx.root.reference_id);
end
...
```

Parameters:

key: the key of the message in the bundle.

parameters: the parameters of the message.

Return:

the new user message.

Can be used in:

211.10 **ofBundleInfo**

function ofBundleInfo(path: string, key: string, parameters: list<object>): user_message

Creates an instance of an info user message from messages bundle.

The user message from the bundle can be created by specifying a path and a key in the EBX IDE.

Example:

```
uses core.message as msg;
export procedure executeScriptTask()
begin
  // the info message from the bundle: 'The script task {0} has been executed successfully'
  // the workflow id is passed as parameter to the message
  var message:= message.ofBundleInfo('bundle/path','script.task.info',_ebx.workflow.id);
end
```

Parameters:

key: the key of the message in the bundle.

path: the path of the bundle.

parameters: the parameters of the message.

Return:

the new user message.

Can be used in:

211.11 ofBundleWarning

function ofBundleWarning(path: string, key: string, parameters: list<object>): user_message

Creates an instance of a warning user message from messages bundle.

The user message from the bundle can be created by specifying a path and a key in the EBX IDE.

Example:

```
uses core.message as msg;
...
export procedure checkRecord(record : .Department)
begin
  // the warning message from the bundle: 'The location of the department {0} is mandatory'
  // the department is passed as parameters to the message
  var message:= message.ofBundleWarning('bundle/path','location.mandatory',record.department_id);
end
....
```

Parameters:

key: the key of the message in the bundle.

path: the path of the bundle.

parameters: the parameters of the message.

Return:

the new user message.

Can be used in:

211.12 toString

function toString(userMessage: user_message, locale: locale): string

Formats the message with the specified locale.

Example:

```
uses core.message as msg;
export procedure executeScriptTask()
begin
  var label:=_ebx.workflow.label;
  var englishLabel:=msg.toString(label,locale.of('en','EN'));
  var frenchLabel:=msg.toString(label,locale.of('fr','FR'));
```

```
end
```

Parameters:

userMessage: the user message.

locale: the locale. If null, the default locale is used.

Return:

the string message formatted with the specified locale or null if the input userMessage is null.

Can be used in:

CHAPTER 212

Unit core.profile

Script unit that provides methods useful when managing profiles.

Methods
function forSpecificRole(roleName: string): profile [p 1503] Returns the specific role for the specified name.
function ofAdministrator(): profile [p 1504] Returns the built-in 'administrator' role.
function ofEveryone(): profile [p 1504] Returns the built-in 'everyone' role.
function ofOwner(): profile [p 1504] Returns the built-in 'owner' role.
function ofReadOnly(): profile [p 1504] Returns the built-in 'read-only' role.

This chapter contains the following topics:

1. forSpecificRole
2. ofAdministrator
3. ofEveryone
4. ofOwner
5. ofReadOnly

212.1 forSpecificRole

function forSpecificRole(roleName: string): profile

Returns the specific role for the specified name.

Can be used in:

212.2 ofAdministrator

function ofAdministrator(): profile

Returns the built-in 'administrator' role.

Can be used in:

212.3 ofEveryone

function ofEveryone(): profile

Returns the built-in 'everyone' role.

Can be used in:

212.4 ofOwner

function ofOwner(): profile

Returns the built-in 'owner' role.

Can be used in:

212.5 ofReadOnly

function ofReadOnly(): profile

Returns the built-in 'read-only' role.

Can be used in:

CHAPTER 213

Unit core.resource

This script unit methods that generates resource references compatible with the schema type ods:resource.

Example:

```
uses core.resource as resource;

export function getValue(): typeof record.functions.ResourceValue
begin
  return resource.toImage('ebx-test','on_anim_wait.gif');
end
```

Methods
function toHtml(moduleName: string, filePath: string): string [p 1506] Returns a reference to an image resource.
function toIcon(moduleName: string, filePath: string): string [p 1506] Returns a reference to an icon resource.
function toImage(moduleName: string, filePath: string): string [p 1506] Returns a reference to an image resource.
function toJavaScript(moduleName: string, filePath: string): string [p 1506] Returns a reference to an JavaScript file resource.
function toStyleSheet(moduleName: string, filePath: string): string [p 1507] Returns a reference to an style sheet file resource.

This chapter contains the following topics:

1. toHtml
2. toIcon
3. toImage
4. toJavaScript
5. toStyleSheet

213.1 **toHtml**

function toHtml(moduleName: string, filePath: string): string

Returns a reference to an image resource.

Parameters:

moduleName: the module name.

filePath: the local file path for the resource.

Return:

the resource string compatible with the schema type ods:resource.

Can be used in:

213.2 **toIcon**

function toIcon(moduleName: string, filePath: string): string

Returns a reference to an icon resource.

Parameters:

moduleName: the module name.

filePath: the local file path for the resource.

Return:

the resource string compatible with the schema type ods:resource.

Can be used in:

213.3 **toImage**

function toImage(moduleName: string, filePath: string): string

Returns a reference to an image resource.

Parameters:

moduleName: the module name.

filePath: the local file path for the resource.

Return:

the resource string compatible with the schema type ods:resource.

Can be used in:

213.4 **toJavaScript**

function toJavaScript(moduleName: string, filePath: string): string

Returns a reference to an JavaScript file resource.

Parameters:

moduleName: the module name.

filePath: the local file path for the resource.

Return:

the resource string compatible with the schema type ods:resource.

Can be used in:

213.5 toStyleSheet

function toStyleSheet(moduleName: string, filePath: string): string

Returns a reference to an style sheet file resource.

Parameters:

moduleName: the module name.

filePath: the local file path for the resource.

Return:

the resource string compatible with the schema type ods:resource.

Can be used in:

CHAPTER 214

Unit core.set

Script unit that provides methods to create and manipulate sets.

Methods
function add(set: set<ItemType>, element: ItemType): boolean [p 1512] Adds the specified element to the set if it is not already present.
function addAll(set: set<ItemType>, elements: collection<ItemType>): boolean [p 1512] Adds all elements from the specified collection to the set.
procedure clear(set: set<ItemType>) [p 1512] Removes all elements from the set.
function clone(set: set<ItemType>): set<ItemType> [p 1512] Returns a shallow clone of the set.
function contains(set: set<ItemType>, element: ItemType): boolean [p 1513] Returns true if set contains the specified element.
function containsAll(set: set<ItemType>, elements: collection<ItemType>): boolean [p 1513] Returns true if set contains all elements of the specified collection.
function difference(set1: set<ItemType>, set2: set<ItemType>): set<ItemType> [p 1513] Returns a new set containing elements that are in the first set but not in the second (difference).
function equals(set1: set<ItemType>, set2: set<ItemType>): boolean [p 1513] Compares two sets for equality.
function fromList(list: list<ItemType>): set<ItemType> [p 1514] Converts a list to a set containing the same elements, preserving order of first occurrence.
function intersection(set1: set<ItemType>, set2: set<ItemType>): set<ItemType> [p 1514] Returns a new set containing only elements that are present in both sets (intersection).
function isEmpty(value: set): boolean [p 1514] Returns true if value is null or an empty set.

Methods
function of(value: ItemType [, value: ItemType]...): set [p 1514] Creates a set with the elements passed as parameters.
function remove(set: set<ItemType>, element: ItemType): boolean [p 1515] Removes the specified element from the set if it is present.
function removeAll(set: set<ItemType>, elements: collection<ItemType>): boolean [p 1515] Removes from the set all of its elements that are contained in the specified collection.
function retainAll(set: set<ItemType>, elements: collection<ItemType>): boolean [p 1515] Retains only the elements in the set that are contained in the specified collection.
function size(set: set<ItemType>): integer [p 1515] Returns the number of elements in the set.
function toList(set: set<ItemType>): list<ItemType> [p 1516] Converts a set to a list containing the same elements, preserving order.
function union(set1: set<ItemType>, set2: set<ItemType>): set<ItemType> [p 1516] Returns a new set containing all elements from both sets (union).

This chapter contains the following topics:

1. add
2. addAll
3. clear
4. clone
5. contains
6. containsAll
7. difference
8. equals
9. fromList
10. intersection
11. isEmpty
12. of
13. remove
14. removeAll
15. retainAll
16. size
17. toList
18. union

214.1 add

function add(set: set<ItemType>, element: ItemType): boolean

Adds the specified element to the set if it is not already present.

Parameters:

set: the set to which the element is to be added.

element: the element to be added to the set.

Return:

true if the set did not already contain the specified element.

Can be used in:

214.2 addAll

function addAll(set: set<ItemType>, elements: collection<ItemType>): boolean

Adds all elements from the specified collection to the set.

Parameters:

set: the set to which elements are to be added.

elements: collection containing elements to be added to the set.

Return:

true if the set was modified as a result of the call.

Can be used in:

214.3 clear

procedure clear(set: set<ItemType>)

Removes all elements from the set.

Parameters:

set: the set to be cleared.

Can be used in:

214.4 clone

function clone(set: set<ItemType>): set<ItemType>

Returns a shallow clone of the set.

Parameters:

set: the set to clone.

Return:

a new set with the same elements as the input set.

Can be used in:

214.5 contains

function contains(set: set<ItemType>, element: ItemType): boolean

Returns true if set contains the specified element.

Parameters:

set: the input set.

element: the element whose presence in the set is to be tested.

Return:

true if the set contains the specified element, false otherwise. Returns null if set is null.

Can be used in:

214.6 containsAll

function containsAll(set: set<ItemType>, elements: collection<ItemType>): boolean

Returns true if set contains all elements of the specified collection.

Parameters:

set: the input set.

elements: collection to be checked for containment in the set.

Return:

true if the set contains all elements of the specified collection, false otherwise. Returns null if any parameter is null.

Can be used in:

214.7 difference

function difference(set1: set<ItemType>, set2: set<ItemType>): set<ItemType>

Returns a new set containing elements that are in the first set but not in the second (difference).

Parameters:

set1: the set from which elements are to be removed.

set2: the set containing elements to be removed from set1.

Return:

a new set containing elements in set1 but not in set2. Returns an empty set if set1 is null.

Can be used in:

214.8 equals

function equals(set1: set<ItemType>, set2: set<ItemType>): boolean

Compares two sets for equality.

Parameters:

set1: the first set.

set2: the second set.

Return:

true if both sets contain the same elements, false otherwise.

Can be used in:

214.9 fromList

function fromList(list: list<ItemType>): set<ItemType>

Converts a list to a set containing the same elements, preserving order of first occurrence.

Parameters:

list: the input list.

Return:

a set containing the unique elements of the list, in order of first occurrence.

Can be used in:

214.10 intersection

function intersection(set1: set<ItemType>, set2: set<ItemType>): set<ItemType>

Returns a new set containing only elements that are present in both sets (intersection).

Parameters:

set1: the first set.

set2: the second set.

Return:

a new set containing only elements present in both sets. Returns an empty set if any parameter is null.

Can be used in:

214.11 isEmpty

function isEmpty(value: set): boolean

Returns true if value is null or an empty set.

Parameters:

value: an input set.

Return:

true if value is null or an empty set, or else false.

Can be used in:

214.12 of

function of(value: ItemType [, value: ItemType]...): set

Creates a set with the elements passed as parameters. All elements must be of same type.

Parameters:

value: items to add to the set. All items must be of same type.

Return:

the new set. It is updatable.

Can be used in:

214.13 remove

function remove(set: set<ItemType>, element: ItemType): boolean

Removes the specified element from the set if it is present.

Parameters:

set: the set from which the element is to be removed.

element: the element to be removed from the set, if present.

Return:

true if the set contained the specified element.

Can be used in:

214.14 removeAll

function removeAll(set: set<ItemType>, elements: collection<ItemType>): boolean

Removes from the set all of its elements that are contained in the specified collection.

Parameters:

set: the set from which elements are to be removed.

elements: collection containing elements to be removed from the set.

Return:

true if the set was modified as a result of the call.

Can be used in:

214.15 retainAll

function retainAll(set: set<ItemType>, elements: collection<ItemType>): boolean

Retains only the elements in the set that are contained in the specified collection.

Parameters:

set: the set to modify.

elements: collection containing elements to be retained in the set.

Return:

true if the set was modified as a result of the call.

Can be used in:

214.16 size

function size(set: set<ItemType>): integer

Returns the number of elements in the set.

Parameters:

set: the input set.

Return:

the number of elements in the set, or 0 if the set is null.

Can be used in:

214.17 toList

function toList(set: set<ItemType>): list<ItemType>

Converts a set to a list containing the same elements, preserving order.

Parameters:

set: the input set.

Return:

a list containing the elements of the set, in iteration order.

Can be used in:

214.18 union

function union(set1: set<ItemType>, set2: set<ItemType>): set<ItemType>

Returns a new set containing all elements from both sets (union).

Parameters:

set1: the first set.

set2: the second set.

Return:

a new set containing all elements from both sets. Returns an empty set if both parameters are null.

Can be used in:

CHAPTER 215

Unit core.string

Script unit that provides string manipulation functions.

Methods
function capitalize(str: string): string [p 1521] Capitalizes a String changing the first character to title case as per Character.
function contains(value: string, searchString: string): boolean [p 1521] Returns true if a value string contains a searchString.
function endsWith(value: string, suffix: string): boolean [p 1521] Returns true if a value string ends with a suffix.
function format(format: string, args: string[]): string [p 1522] Returns a formatted string using the specified format string and arguments.
function formatWithLocale(format: string, args: string[]): string [p 1522] Returns a formatted string using the specified format string and arguments in the specified Locale.
function indexOf(str: string, searchStr: string): number [p 1522] Finds the first index within a String, handling null.
function indexOfFromIndex(str: string, searchStr: string, fromIndex: number): number [p 1523] Finds the first index within a String, starting from a specified index, handling null.
function isAlphanumeric(str: string): boolean [p 1523] Checks if a String is alphanumeric.
function isBlank(str: string): boolean [p 1524] Checks if a String is empty (""), null, or whitespace only.
function isEmpty(value: string): boolean [p 1524] Returns true if value is null or a zero length string.
function isNumeric(str: string): boolean [p 1524] Checks if the String contains only Unicode digits.

Methods
function join(delimiter: string [, substring: string]...): string [p 1525] Returns a new string composed of all input substrings separated by a delimiter.
function joinExtended(delimiter: string, prefix: string, suffix: string [, substring: string]...): string [p 1525] Returns a new string composed of all input substrings separated by a delimiter, with a prefix and a suffix.
function joinList(delimiter: string, prefix: string, suffix: string, list: list): string [p 1525] Returns a new string compose of all list items separated by a delimiter, with a prefix and a suffix.
function lastIndexOf(str: string, searchStr: string): number [p 1526] Returns the index within a string of the rightmost occurrence of the specified substring.
function lastIndexOfFromIndex(str: string, searchStr: string, fromIndex: number): number [p 1526] returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
function replaceAll(value: string, pattern: string, replacement: string): string [p 1527] Returns a new string with all pattern occurrences in input value string replaced with replacement.
function replaceFirst(value: string, pattern: string, replacement: string): string [p 1527] Returns a new string with the first pattern occurrence in input value string replaced with replacement.
function reverse(str: string): string [p 1527] Reverses a String.
function right(value: string, index: decimal): string [p 1528] Returns a new string that contains the right characters from value string.
function split(str: string): string[] [p 1528] Splits the provided text into an array, using whitespace as the separator.
function split(str: string, separator: string): string[] [p 1528] Splits the provided text into an array, with the specified separator.
function startsWith(value: string, prefix: string): boolean [p 1529] Returns true if a value string starts with a prefix.
function substring(value: string, startIndex: decimal, endIndex: decimal): string [p 1529] Returns a new string that contains a subsequence of characters from value string.
function toLowerCase(value: string): string [p 1529] Returns a new string with all characters from input value string to lower case.
function toUpperCase(value: string): string [p 1530] Returns a new string with all characters from input value string to upper case.

Methods
function trim(str: string): string [p 1530] Removes control characters (char <= 32) from both ends of this String, handling null by returning null.
function valueOf(obj: Object): string [p 1530] Returns the string representation of the Object argument.

This chapter contains the following topics:

1. capitalize
2. contains
3. endsWith
4. format
5. formatWithLocale
6. indexOf
7. indexOfFromIndex
8. isAlphanumeric
9. isBlank
10. isEmpty
11. isNumeric
12. join
13. joinExtended
14. joinList
15. lastIndexOf
16. lastIndexOfFromIndex
17. replaceAll
18. replaceFirst
19. reverse
20. right
21. split
22. splitWith
23. startsWith
24. substring
25. toLowerCase
26. toUpperCase
27. trim
28. valueOf

215.1 capitalize

function capitalize(str: string): string

Capitalizes a String changing the first character to title case as per Character.toTitleCase(Number).

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.capitalize('hello'); // "Hello"
  var result2 := s.capitalize('hello world'); // "Hello world"
  var result3 := s.capitalize(''); // ""
  var result4 := s.capitalize(null); // null
  ...
end
```

Parameters:

str The input string

Return:

The capitalized string

Can be used in:

215.2 contains

function contains(value: string, searchString: string): boolean

Returns true if a value string contains a searchString.

Parameters:

value: the input string

searchString: the input searchString to look for

Return:

true if value contains the searchString or else false.

Returns null if any parameters is null.

Can be used in:

215.3 endsWith

function endsWith(value: string, suffix: string): boolean

Returns true if a value string ends with a suffix.

Parameters:

value: the input string

prefix: the input suffix to look for

Return:

true if value ends with suffix or else false.

Returns null if any parameters is null.

Can be used in:

215.4 **format**

function format(format: string, args: string[]): string

Returns a formatted string using the specified format string and arguments. this will use Locale.US

Example:

```
uses core.string as s;
uses core.locale as l;
procedure doSomething()
begin
  var result := s.format(l.fromString(en-US), 'Hello, %s!', 'world'); // 'Hello, world!'
  var result2 := s.format(l.fromString(en-US),'The value is %.2f', 3.14159); // 'The value is 3.14'
  var result2 := s.format(l.fromString(fr-FR),'The value is %.2f', 3.14159); // 'The value is 3.14'
  var result3 := s.format(l.fromString(en-US), null); // null
  var result4 := s.format(l.fromString(en-US), 'No arguments'); // 'No arguments'
  ...
end
```

Parameters:

format The format string

args Arguments referenced by the format specifiers in the format string

Return:

The formatted string

Can be used in:

215.5 **formatWithLocale**

function formatWithLocale(format: string, args: string[]): string

Returns a formatted string using the specified format string and arguments in the specified Locale.

Example:

```
uses core.string as s;
uses core.locale as l;
procedure doSomething()
begin
  var result := s.formatWithLocale(l.fromString(en-US), 'Hello, %s!', 'world'); // 'Hello, world!'
  var result2 := s.formatWithLocale(l.fromString(en-US),'The value is %.2f', 3.14159); // 'The value is 3.14'
  var result2 := s.formatWithLocale(l.fromString(fr-FR),'The value is %.2f', 3.14159); // 'The value is 3.14'
  var result3 := s.formatWithLocale(l.fromString(en-US), null); // null
  var result4 := s.formatWithLocale(l.fromString(en-US), 'No arguments'); // 'No arguments'
  ...
end
```

Parameters:

local the local used for formatting

format The format string

args Arguments referenced by the format specifiers in the format string

Return:

The formatted string

Can be used in:

215.6 **indexOf**

function indexOf(str: string, searchStr: string): number

Finds the first index within a String, handling null.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.indexOf('hello', 'e'); // 1
  var result2 := s.indexOf('hello', 'z'); // -1
  var result3 := s.indexOf(null, 'e'); // -1
  ...
end
```

Parameters:

str The input string

searchStr The substring to search for

Return:

The index of the first occurrence of the substring within the string, or -1 if not found

Can be used in:

215.7 indexOfFromIndex

function indexOfFromIndex(str: string, searchStr: string, fromIndex: number): number

Finds the first index within a String, starting from a specified index, handling null.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.indexOfFromIndex('hello', 'e', 0); // 1
  var result2 := s.indexOfFromIndex('hello', 'z', 0); // -1
  var result3 := s.indexOfFromIndex(null, 'e', 0); // -1
  ...
end
```

Parameters:

str The input string

searchStr The substring to search for

fromIndex The index to start the search from

Return:

The index of the first occurrence of the substring within the string, or -1 if not found

Can be used in:

215.8 isAlphanumeric

function isAlphanumeric(str: string): boolean

Checks if a String is alphanumeric.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.isAlphanumeric("hello123"); // true
  var result2 := s.isAlphanumeric("123"); // true
  var result3 := s.isAlphanumeric("hello"); // true
  var result4 := s.isAlphanumeric(null); // false
  ...
```

```
end
```

Parameters:

str The input string

Return:

true if the string is alphanumeric, false otherwise

Can be used in:

215.9 isBlank

function isBlank(str: string): boolean

Checks if a String is empty (""), null, or whitespace only.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.isBlank(''); // true
  var result2 := s.isBlank('   '); // true
  var result3 := s.isBlank(null); // true
  var result4 := s.isBlank('abc'); // false
  ...
end
```

Parameters:

str The input string

Return:

true if the string is empty, null, or whitespace only, false otherwise

Can be used in:

215.10 isEmpty

function isEmpty(value: string): boolean

Returns true if value is null or a zero length string.

Parameters:

value: a string value.

Return:

true if value is null or a zero length string, or else false. Never returns null.

Can be used in:

215.11 isNumeric

function isNumeric(str: string): boolean

Checks if the String contains only Unicode digits. Note that the method does not allow for a leading sign, either positive or negative.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.isNumeric('12345'); // true
```

```
  var result2 := s.isNumeric('-12345'); // false
  var result3 := s.isNumeric('12.345'); // false
  var result4 := s.isNumeric(null); // false
  ...
end
```

Parameters:

str The input string

Return:

true if the string contains only Unicode digits, false otherwise

Can be used in:

215.12 join

function join(delimiter: string [, substring: string]...): string

Returns a new string composed of all input substrings separated by a delimiter.

Parameters:

delimiter: the input delimiter string

substring: a substring to add to the result string. All element must be string.

Return:

a new string composed of all input substrings separated by the delimiter.

If delimiter is null, returns null. Any null substring is replaced by string 'null'.

Can be used in:

215.13 joinExtended

function joinExtended(delimiter: string, prefix: string, suffix: string [, substring: string]...): string

Returns a new string composed of all input substrings separated by a delimiter, with a prefix and a suffix.

Parameters:

delimiter: the input delimiter string.

prefix: the input prefix string.

suffix: the input suffix string.

substring: a substring to add to the result string. All element must be string.

Return:

a new string composed of all input substrings separated by the delimiter.

If delimiter, prefix or suffix are null, returns null. Any null substring is replaced by string 'null'.

Can be used in:

215.14 joinList

function joinList(delimiter: string, prefix: string, suffix: string, list: list): string

Returns a new string compose of all list items separated by a delimiter, with a prefix and a suffix.

Parameters:

delimiter: the input delimiter string.

prefix: the input prefix string.

suffix: the input suffix string.

list: a list of strings to add to the result string.

Return:

a new string compose of all input substrings separated by the delimiter input.

If any parameters is null returns null. Any null item of the list is replaced by string 'null'.

Can be used in:

215.15 lastIndexOf

function lastIndexOf(str: string, searchStr: string): number

Returns the index within a string of the rightmost occurrence of the specified substring. The rightmost empty string "" is considered to occur at the index value str.length().

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.lastIndexOf('hello', 'o'); // 4
  var result2 := s.lastIndexOf('hello', 'z'); // -1
  var result3 := s.lastIndexOf(null, 'e'); // -1
  ...
end
```

Parameters:

str The input string

searchStr The substring to search for

Return:

The index of the last occurrence of the substring within the string, or -1 if not found

Can be used in:

215.16 lastIndexOfFromIndex

function lastIndexOfFromIndex(str: string, searchStr: string, fromIndex: number): number

returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.lastIndexOfFromIndex('hello', 'o', 3); // 4
  var result2 := s.lastIndexOfFromIndex('hello', 'o', 2); // 4
  var result3 := s.lastIndexOfFromIndex('hello', 'o', 0); // -1
  ...
end
```

Parameters:

str The input string

searchStr The substring to search for

fromIndex The index to start the search from

Return:

The index of the last occurrence of the substring within the string, or -1 if not found

Can be used in:

215.17 replaceAll

function replaceAll(value: string, pattern: string, replacement: string): string

Returns a new string with all pattern occurrences in input value string replaced with replacement.

Parameters:

value: the input string

pattern: the string pattern (regex) to replace

replacement: the replacement string

Return:

the new string or null if parameter is null.

Can be used in:

215.18 replaceFirst

function replaceFirst(value: string, pattern: string, replacement: string): string

Returns a new string with the first pattern occurrence in input value string replaced with replacement.

Parameters:

value: the input string

pattern: the string pattern (regex) to replace

replacement: the replacement string

Return:

the new string or null if any parameter is null.

Can be used in:

215.19 reverse

function reverse(str: string): string

Reverses a String.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.reverse('hello'); // "olleh"
  var result2 := s.reverse(''); // ''
  var result3 := s.reverse(null); // null
  ...
end
```

Parameters:

str The input string

Return:

The reversed string

Can be used in:

215.20 right

function right(value: string, index: decimal): string

Returns a new string that contains the right characters from value string.

The staring starts from the specified index and extends to the end of the string.

Parameters:

value: the input string

index: the input start index (inclusive). Must be an integer greater or equal to zero and less or equal than the length of the string

Return:

the new string or null if any parameter is null.

Can be used in:

215.21 split

function split(str: string): string[]

Splits the provided text into an array, using whitespace as the separator.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.split('hello world'); // ['hello', 'world']
  var result2 := s.split(''); // ['']
  var result3 := s.split(null); // null
  ...
end
```

Parameters:

str The input string

Return:

An array of substrings

Can be used in:

215.22 splitWith

function split(str: string, separator: string): string[]

Splits the provided text into an array, with the specified separator.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.split('hello,world', ','); // ['hello', 'world']
  var result2 := s.split('hello world', ' '); // ['hello', 'world']
```

```
  var result3 := s.split('', ' '); // ['']
  var result4 := s.split(null, ' '); // null
  ...
end
```

Parameters:

str The input string

separator The separator string

Return:

An array of substrings

Can be used in:

215.23 **startsWith**

function startsWith(value: string, prefix: string): boolean

Returns true if a value string starts with a prefix.

Parameters:

value: the input string

prefix: the input prefix to look for

Return:

true if value starts with prefix or else false.

Returns null if any parameters is null.

Can be used in:

215.24 **substring**

function substring(value: string, startIndex: decimal, endIndex: decimal): string

Returns a new string that contains a subsequence of characters from value string.

The sub string start from the specified startIndex and extends to the endIndex of the input string.

Parameters:

value: the input string

startIndex: the input start index (inclusive). Must be an integer greater or equal to zero and less or equal than the length of the string.

endIndex: the input end index (exclusive). Must be an integer greater or equal to startIndex and less or equal than the length of the string.

Return:

the new string or null if any parameter is null.

Can be used in:

215.25 **toLowerCase**

function toLowerCase(value: string): string

Returns a new string with all characters from input value string to lower case.

Parameters:

value: the input string

Return:

the lower cased string or null if parameter is null.

Can be used in:

215.26 toUpperCase

function toUpperCase(value: string): string

Returns a new string with all characters from input value string to upper case.

Parameters:

value: the input string

Return:

the upper cased string or null if parameter is null.

Can be used in:

215.27 trim

function trim(str: string): string

Removes control characters (char <= 32) from both ends of this String, handling null by returning null.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.trim('  hello  '); // "hello"
  var result2 := s.trim('   '); // ""
  var result3 := s.trim(null); // null
  ...
end
```

Parameters:

str The input string

Return:

The trimmed string

Can be used in:

215.28 valueOf

function valueOf(obj: Object): string

Returns the string representation of the Object argument.

Example:

```
uses core.string as s;
procedure doSomething()
begin
  var result := s.valueOf('hello'); // 'hello'
  var result2 := s.valueOf(123); // '123'
  var result3 := s.valueOf(null); // null
  ...
end
```

Parameters:

obj The object to convert to a string

Return:

The string representation of the object

Can be used in:

CHAPTER 216

Unit core.time

Script unit that provides time functions.

Methods
function addHours(value: time, hours: decimal): time [p 1534] Adds hours to a time.
function addMinutes(value: time, minutes: decimal): time [p 1534] Adds minutes to a time.
function addSeconds(value: time, seconds: decimal): time [p 1534] Adds seconds to a time.
function fromDatetime(value: datetime): datetime [p 1535] Converts a date and time to a time.
function fromIsoString(value: string): date [p 1535] * Parses a time from a string in ISO format (yyyy-MM-dd'T'HH:mm:ss.
function now(): time [p 1535] Returns current time.
function of(hours: decimal, minutes: decimal, day: decimal): time [p 1535] Creates a time given hours, minutes and seconds.
function seconds(startValue: time, endValue: time): decimal [p 1536] Returns the number of seconds between two times.
function toIsoString(value: date): string [p 1536] Converts a time to a string in ISO format (HH:mm:ss.

This chapter contains the following topics:

1. addHours
2. addMinutes
3. addSeconds
4. fromDatetime
5. fromIsoString
6. now
7. of
8. seconds
9. toIsoString

216.1 addHours

function addHours(value: time, hours: decimal): time

Adds hours to a time. The calculation wraps around midnight.

Parameters:

value: the input time

hours: the number of hours to be added. Must be an integer.

Return:

the new time or null if any parameter is null.

Can be used in:

216.2 addMinutes

function addMinutes(value: time, minutes: decimal): time

Adds minutes to a time. The calculation wraps around midnight.

Parameters:

value: the input time

minutes: the number of minutes to be added. Must be an integer.

Return:

the new time or null if any parameter is null.

Can be used in:

216.3 addSeconds

function addSeconds(value: time, seconds: decimal): time

Adds seconds to a time. Fractions of seconds are supported with millisecond precision. The calculation wraps around midnight.

Parameters:

value: the input time

seconds: the number of seconds to be added. Fractions are rounded to three decimal digits.

Return:

the new time or null if any parameter is null.

Can be used in:

216.4 fromDatetime

function fromDatetime(value: datetime): datetime

Converts a date and time to a time. The date part is ignored.

Parameters:

value: the input date and time

Return:

the new time or null if parameter is null.

Can be used in:

216.5 fromIsoString

function fromIsoString(value: string): date

* Parses a time from a string in ISO format (yyyy-MM-dd'T'HH:mm:ss.SSS). The millisecond part is optional.

Parameters:

value: the input string

Return:

the date or null if value is null or format is not valid.

Can be used in:

216.6 now

function now(): time

Returns current time.

Return:

the current time.

Can be used in:

216.7 of

function of(hours: decimal, minutes: decimal, day: decimal): time

Creates a time given hours, minutes and seconds. Fractions of seconds are supported with millisecond precision.

Parameters:

hour: the input hour

minute: the input minute

second: the input second. Fractions are rounded to three decimal digits.

Return:

the new time or null if any parameter is null.

Can be used in:

216.8 seconds

function seconds(startValue: time, endValue: time): decimal

Returns the number of seconds between two times.

Parameters:

startValue: the input start date and time inclusive

endValue: the input end date and time exclusive

Return:

the number of seconds or null if any parameter is null. Return value may have a fractional part.

Can be used in:

216.9 toIsoString

function toIsoString(value: date): string

Converts a time to a string in ISO format (HH:mm:ss.SSS)

Parameters:

value: the input time

Return:

the string or null if value is null.

Can be used in:

CHAPTER 217

Unit core.uri

Script unit that provides functions for managing Uniform Resource Identifier (URI) references.

Methods
function of(value: string): uri [p 1537] Creates an URI reference from a string.

This chapter contains the following topics:

1. of

217.1 of

function of(value: string): uri

Creates an URI reference from a string.

Parameters:

value: the input value, for example 'https://www.tibco.com'.

Return:

the URI.

Can be used in:

CHAPTER 218

Unit core.validation

Script unit that provides methods for setting dependencies in the context of a validation constraint.

Methods
procedure addDependencyToInsertAndDelete(path: string) [p 1540] Adds a dependency path to specify that the validation result of the current constraint depends on insertions and deletions of the specified table path.
procedure addDependencyToInsertAndDeleteAndModifyInOtherDataset(path: string, dataset: string, dataspace: string) [p 1540] Adds a dependency path to specify that the validation result of the current constraint depends on insertions, modifications and deletions of the table or the table field specified by its path in another dataset.
procedure addDependencyToInsertAndDeleteInOtherDataset(path: string, dataset: string, dataspace: string) [p 1541] Adds a dependency path to specify that the validation result of the current constraint depends on insertions and deletions of the specified table of a given dataset in a specific dataspace.
procedure addDependencyToInsertDeleteAndModify(path: string) [p 1541] Adds a dependency path to explicitly declare that the validation result of the current constraint depends on insertions, modifications and deletions of the table or the table field specified by its path.
procedure addDependencyToModify(path: string) [p 1542] Adds a dependency path to explicitly declare that the validation result of the current constraint depends on modifications of the field specified by its path.
procedure disableAjaxPrevalidation() [p 1543] Prevents, in the user interface, the constraint from being pre-validated dynamically when the related field is updated at user inputs.
procedure setBlocksCommitToNever() [p 1543] Specifies that the current constraint must never block an operation, even if it adds validation errors.
procedure setDependencyToLocalNode() [p 1543] Specifies that the validation result of the current constraint depends only on the local value of the node.

This chapter contains the following topics:

1. addDependencyToInsertAndDelete
2. addDependencyToInsertAndDeleteAndModifyInOthe…
3. addDependencyToInsertAndDeleteInOtherDataset
4. addDependencyToInsertDeleteAndModify
5. addDependencyToModify
6. disableAjaxPrevalidation
7. setBlocksCommitToNever
8. setDependencyToLocalNode

218.1 addDependencyToInsertAndDelete

procedure addDependencyToInsertAndDelete(path: string)

Adds a dependency path to specify that the validation result of the current constraint depends on insertions and deletions of the specified table path. Raises an error if the table path is not found at the specified path when publishing the script.

Node that this method must be called from the procedure setup of the constraint otherwise it is ignored.

Example:

```
uses core.validation as v;
uses core.message as msg;
export procedure checkTable()
begin
end

export procedure checkRecord(record : .Product)
begin
  if (record.Price < record.Threshold) then
  begin
    var message:='The price of the product '|_ebx.record.Price |' is under the threshold '|record.PriceThreshold;
    msg.addWarning(message);
  end

  export procedure setup()
  begin
    v.addDependencyToInsertDelete('/root/ProductCatalog');
  end
```

Parameters:

path: the table path.

Can be used in:

218.2 addDependencyToInsertAndDeleteAndModifyInOthe…

procedure addDependencyToInsertAndDeleteAndModifyInOtherDataset(path: string, dataset: string, dataspace: string)

Adds a dependency path to specify that the validation result of the current constraint depends on insertions, modifications and deletions of the table or the table field specified by its path in another dataset. Raises an error if the table path or the table field path is not found at the specified dataset when publishing the script.

Node that this method must be called from the procedure setup of the constraint otherwise it is ignored.

Example:

```
uses core.validation as v;
uses core.data as d;
export procedure check()
begin
  // do some validation checks
end

export procedure setup()
begin
  v.addDependencyToInsertAndDeleteAndModifyInOtherDataset("/root/table/field", 'otherDataset', 'otherDataspace');
end
```

Parameters:

path: the field path.

dataset: the other dataset.

dataspace: the other dataspace.

Can be used in:

218.3 addDependencyToInsertAndDeleteInOtherDataset

procedure addDependencyToInsertAndDeleteInOtherDataset(path: string, dataset: string, dataspace: string)

Adds a dependency path to specify that the validation result of the current constraint depends on insertions and deletions of the specified table of a given dataset in a specific dataspace. Raises an error if the path is not a table path or is not found at the specified dataset and dataspace when publishing the script.

Node that this method must be called from the procedure setup of the constraint otherwise it is ignored.

Example:

```
uses core.validation as v;
uses core.data as d;
export procedure checkTable()
begin
  // do some validation checks
end

export procedure checkRecord(record : .table)
begin
  // do some validation checks
end
export procedure setup()
begin
  v.addDependencyToInsertAndDeleteInOtherDataset("/root/table",'otherDataset','otherDataspace');
end
```

Parameters:

path: the table path.

dataset: the other dataset name.

dataspace: the other dataspace name.

Can be used in:

218.4 addDependencyToInsertDeleteAndModify

procedure addDependencyToInsertDeleteAndModify(path: string)

Adds a dependency path to explicitly declare that the validation result of the current constraint depends on insertions, modifications and deletions of the table or the table field specified by its path. Raises an error if the table path or the table field path is not found at the specified path when publishing the script.

Node that this method must be called from the procedure setup of the constraint otherwise it is ignored.

Example:

```
uses core.validation as v;
uses core.message as msg;
export procedure checkTable()
begin
end

export procedure checkRecord(record : .Employee)
begin
  if (record.number < 100) then
  begin
    msg.addError("The number must be at least 100");
  end
end

export procedure setup()
begin
  v.addDependencyToInsertDeleteAndModify('/root/Employee/number');
end
```

Parameters:

path: the table path or the table field path.

Can be used in:

218.5 addDependencyToModify

procedure addDependencyToModify(path: string)

Adds a dependency path to explicitly declare that the validation result of the current constraint depends on modifications of the field specified by its path. Raises an error if the field does not have a local value or is not found at the specified path when publishing the script.

Node that this method must be called from the procedure setup of the constraint otherwise it is ignored.

Example:

```
uses core.validation as v;
uses core.message as msg;
export procedure checkTable()
begin
end

export procedure checkRecord(record : .ProductCatalog)
begin
  if (not(str.endsWith(_ebx.root.code,'TIBCO_'))) then
  begin
    msg.addWarning("The code must end with 'TIBCO_'");
  end
end

export procedure setup()
begin
  v.addDependencyToModify('/root/code');
end
```

Parameters:

path: the field path.

Can be used in:

218.6 disableAjaxPrevalidation

procedure disableAjaxPrevalidation()

Prevents, in the user interface, the constraint from being pre-validated dynamically when the related field is updated at user inputs. This can be useful to improve performance or to avoid unnecessary validation errors during form updates. However, the constraint will still be validated when the form is submitted or when loading the record in the view.

Example:

```
uses core.validation as v;
uses core.message as msg;
export procedure checkTable()
begin
  // do some validation checks
end
export procedure checkRecord(record : .table)
begin
  // do some validation checks
end

export procedure setup()
begin
  v.disableAjaxPrevalidation();
end
```

Can be used in:

218.7 setBlocksCommitToNever

procedure setBlocksCommitToNever()

Specifies that the current constraint must never block an operation, even if it adds validation errors.

Example:

```
uses core.validation as v;
uses core.message as msg;
export procedure checkTable()
begin
  // do some validation checks
end
export procedure checkRecord(record : .table)
begin
  // do some validation checks
end

export procedure setup()
begin
  v.setBlocksCommitToNever();
end
```

Can be used in:

218.8 setDependencyToLocalNode

procedure setDependencyToLocalNode()

Specifies that the validation result of the current constraint depends only on the local value of the node.

Example:

```
uses core.validation as v;
uses core.message as msg;
export procedure check()
begin
  // do some validation checks
end
```

```
export procedure setup()
begin
  v.setDependencyToLocalNode();
end
```

Can be used in:

CHAPTER 219

Unit core.rest

Script unit that provides methods useful when writing REST services.

Methods
function emptyResponse(status: int): rest_response [p 1545] Creates a response without a body.
function responseOf(status: int, content: any): rest_response [p 1545] Creates a JSON response with an optional HTTP status.

This chapter contains the following topics:

1. emptyResponse
2. responseOf

219.1 emptyResponse

function emptyResponse(status: int): rest_response

Creates a response without a body.

Parameters:

status: the status code. If null, 204 is assumed.

Return:

the response.

Can be used in:

219.2 responseOf

function responseOf(status: int, content: any): rest_response

Creates a JSON response with an optional HTTP status.

Parameters:

status: the status code. If null, 200 is assumed.

content: the content to serialize. Supports typed objects from JSON schemas, generic json_value, and lists (including multi-dimensional) of these types. If null, the body is the JSON value null.

Return:

the JSON response.

Can be used in:

EBX® Script IDE

CHAPTER 220

Overview of the EBX® Script IDE

Accessible as a built-in EBX® perspective, the Script IDE allows you to write functions, triggers, workflow scripts, and scripted REST services using the EBX® Script Language (ESL) directly in your browser. The IDE provides significant benefits in terms of user accessibility, convenience, and efficiency. It also alleviates the need to use Java for the aforementioned tasks. It maintains the same performance, but involves less overhead as no compilation and deployment of custom modules is required. Additionally, ESL's design makes it accessible to those without a Java programming background.

The documentation for the IDE is organized as follows:

- How to guide [p 1549]: Similar to a quick start guide, this guide provides high level instructions on performing common tasks. This section is intended for those with programming experience or who are already comfortable with ESL.
- Tutorials: The tutorials are essentially lessons for users that don't have much programming experience and guide you through the following processes:
  - Functions [p 1553]
  - Triggers [p 1559]
  - Workflow scripts [p 1563]
  - Scripted REST services [p 1569]
- User interface Reference [p 1579]: This section highlights the IDEs UI components.

CHAPTER 221

How to guide

This chapter contains the following topics:

1. Overview
2. Create functions
3. Create and manage workflow script tasks
4. Create triggers
5. Create scripted REST services
6. Use history to roll back changes
7. Check the scripting logs

221.1 Overview

This section is intended for more experienced TIBCO EBX® users and provides high-level information on common tasks related to the EBX® Script IDE. For additional documentation resources:

- The *Administration Guide > EBX® Script* section of the documentation provides details about the EBX® Script Language (ESL).
- The *Administration Guide > EBX® Script IDE > Tutorials* section provides more detailed informations on some basic use cases that are designed for less experienced users.

221.2 Create functions

Prior to writing a function in the EBX® Script IDE, you must first create a data model field using the function type. After field creation, the script is available to edit in the Script IDE.

To add a function type field to your data model:

1. Create a new data model field and set the **Kind of element** to **Function**.
2. Use the **Data type** field to specify the desired data type. The data type selected will automatically be reflected in the script in the IDE.
3. Save and optionally publish your data model. Note that once the field is saved, the script outline is automatically created and available to work with in the IDE. However, you cannot publish the script until the data model publication is up to date. So it might be a good idea to publish the data model at this time.

4. If the EBX® Script IDE perspective is not open:
    1. Click the  icon at the top right of the EBX® toolbar and select **Script IDE**.
    2. In the Dock on the left, under **SCRIPTING**, select **Data model**.
5. Expand the data model where the function was added, write, and publish your script.

See also

Creating functions tutorial *[p 1553]*

Function field *[p 1343]*

221.3 Create and manage workflow script tasks

The creation of workflow script tasks can be done directly in the Script IDE. Once a workflow script is published, it is available to use in a workflow model.

To create a workflow script:

1. If the EBX® Script IDE perspective is not open:
    1. Click the  icon at the top right of the EBX® toolbar and select **Script IDE**.
    2. In the Dock on the left, under **SCRIPTING**, select **Workflow**.
2. Create a new script.
3. Add required information and parameters:
    1. Enter a script name, label, and optional description.
    2. Add parameters that are used as input/output to access EBX® components, such as dataspaces, datasets, tables, and records.
4. After writing the script, select the **PUBLISH** button at the bottom of the **Script** tab.

    The script is now available to select in a workflow model task by doing the following:
    1. Select **Script** from the **Step type** menu, and click **Next**.
    2. For the **Script** set the **Input mode** to **Script** and use the **Script name** menu to select the workflow script.

See also

Creating a workflow script tutorial *[p 1563]*

Script task *[p 1345]*

221.4 Create triggers

Prior to writing the logic for a trigger in the EBX® Script IDE, you must first add a trigger to a data model. Once you create the trigger in the data model, it is automatically available in the EBX® Script IDE.

To add a trigger to a table in your data model:

1. Open the data model and select the **Advanced properties** tab of the desired data model element.
2. Add and configure a trigger that uses the **[Built-in] EBX Script** component.

3. Save and then optionally publish your data model. Note that once the table is saved after adding the trigger, the trigger's outline is automatically created and available to work with. However, you cannot publish the script until the data model publication is up to date. So it might be a good idea to publish the data model at this time.
4. If the EBX® Script IDE perspective is not open:
    1. Click the  icon at the top right of the EBX® toolbar and select **Script IDE**.
    2. In the Dock on the left, under **SCRIPTING**, select **Data model**.
5. Expand the data model where the trigger was added, write, and publish your script.

See also

Creating triggers tutorial *[p 1559]*

Table trigger *[p 1351]*

Dataset trigger *[p 1355]*

221.5 Create scripted REST services

A scripted REST service is the combination of a REST endpoint and the scripted logic that executes when the endpoint is accessed.

To create a scripted REST service in the Script IDE:

1. If the EBX® Script IDE perspective is not open:
    1. Click the  icon at the top right of the EBX® toolbar and select **Script IDE**.
    2. In the Dock on the left, under **SCRIPTING**, select **REST service**.
2. Use one of the following methods to create your REST endpoints:
    1. Define the endpoint manually using the **API** and **DATA TYPES** tabs.
    2. When creating the initial REST service, select **UPLOAD FILE** to import the REST API definition in a YAML or JSON file that complies with the OpenAPI specification.
3. Use the **SCRIPT** tab to write the programmatic logic for the service.

> **Attention**
> When creating a function, its name must match the **Operation id** of the endpoint that will call the function.

4. Once published, use the **TEST API** option to use the IDE's built-in tester.

See also

Creating scripted REST services tutorial *[p 1569]*

Scripted REST service *[p 1359]*

221.6 Use history to roll back changes

Each script and API in the IDE includes a history of all its revisions. Each revision is created when the script is saved.

To view changes to a script:

1. If the EBX® Script IDE perspective is not open, click the  icon at the top right of the EBX® toolbar and select **Script IDE**.
2. Chose the script type in the Dock on the left.
3. Select either **SCRIPT HISTORY** or **API HISTORY** tab at the top of the workspace.
4. Use the **Revisions** column to browse changes and copy/paste or make edits to the current revision column.
5. Use the buttons at the bottom of the tab to save, compile, and publish the script.

221.7 Check the scripting logs

When you include logging in a script, you can access log output in the following location: *Administration > Scripting management > Script logs > Logs*.

CHAPTER 222

Creating functions

This chapter contains the following topics:

1. Introduction to functions
2. Function prerequisite: add functions to data model
3. Creating a function
4. Creating a non-indexed function

222.1 Introduction to functions

A function takes an instruction set designed to perform actions on your data. Think of it like a small application that takes data as input, processes it according to a set of rules, and then produces a desired output. For example, a function might combine the values from two fields, calculate the average value from a set of numbers, or simply check whether certain conditions are true.

Functions can also handle more complex data operations efficiently without requiring a deep understanding of the underlying technical complexities. In summary, even if you are new to data operations, functions allow you to manipulate and analyze data effectively, easing tasks that would otherwise require extensive technical knowledge.

Indexed vs non-indexed functions

When a function is indexed, it means that the field the function operates on is optimized for search queries. However, an indexed function cannot access the machine's local time or other methods that are incompatible with optimised search. This tutorial includes examples of each type of function.

How this tutorial is organized

This tutorial shows how to create and publish functions in the EBX® Script IDE. It does not cover complex use cases, or all of the user interface options. To follow along, you can create the basic data model shown below and leave out the fields highlighted in red as they are created in the tutorial prerequisite section:

Data model component	Label
Data Model: `docPerson`	N/A
Table: `person`	Person
PK field, Integer (set to **Auto-increment**): `id`	Id
String field: `first_name`	First name
String field: `last_name`	Last name
String field: `full_name`	Full name
Date field: `last_login`	Last login
Boolean field: `status`	Active user
Date field: `last_modification_date`	Last modified

The parts that make up this tutorial are:

1. Function prerequisites [p 1554]
2. Creating a function [p 1555]
3. Creating a non-indexed function [p 1556]

222.2 Function prerequisite: add functions to data model

Prior to writing a function in the EBX® Script IDE, you must first create a data model field and specify that it is a `function`. This field will display the output from your function. Therefore, the data type of this field must correspond to the expected return type of your function (e.g., date, string, integer, etc.).

To add a function type field to your data model:

1. Open your data model in the DMA, and create the following fields:
   - Field 1: The function written for this field will concatenate the first and last name fields and should return a string.
     - **Name**: `full_name`
     - **Label and description**: `Full name`
     - **Kind of element**: `Field` and tic the **Function** checkbox.
     - **Data type**: `String`
   - Field 2: The function written for this field will use a person's last login to determine whether they are considered an active user. Its return/data type will be `Boolean`.
     - **Name**: `status`
     - **Label and description**: `Active user`
     - **Kind of element**: `Field` and tic the **Function** checkbox.
     - **Data type**: `Boolean`

2. Save and optionally publish your data model. Note that once the field is saved, the script outline is automatically generated and available to work with. However, you cannot publish the script until the data model publication is up to date. So it might be a good idea to publish the data model at this time.

   What's next? Create an indexed [p 1555] or non-indexed [p 1556] function.

222.3 Creating a function

By default, when a function is created, it is set to be indexed. So no special consideration is required. However, you can check to see whether your function is indexed by selecting the **Information tab** and ensuring the **Index** property is checked.

Script tip: Familiarize yourself with built-in ESL functions and variables. See the API reference documentation for more information.

To create a function that can be indexed:

1. If the EBX® Script IDE perspective is not open, click the  icon at the top right of the EBX® toolbar and select **Script IDE**.
2. In the Activity Bar on the left, locate and select **Data model**.
3. In the middle Navigation Panel locate, and select the `Full name` function created in the prerequisite section above. The  icon indicates a function.

   As shown below, the function is automatically populated based on the field's data type (string) and returns a string value:

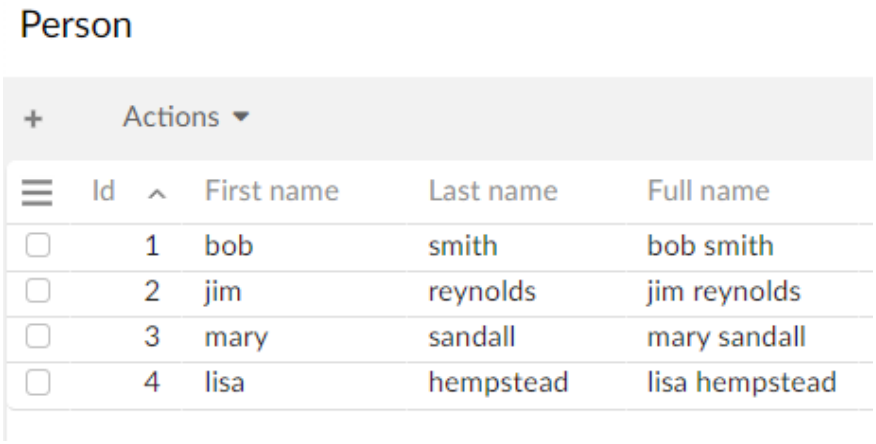
```
export function getValue(): string
begin
 return 'A string value';
end
```

4. In this example function, a value is computed as follows:
   - Retrieve the Person's first and last name using the `_ebx.record.first_name` and `_ebx.record.last_name` variables, respectively. In these variables, the `_ebx.record` is a built-in function that accesses the current record and the *.<name>* is the name of the field you want to retrieve in the data model. See Creating a non-indexed function [p 1556] for an example of how to use additional built-in functions.
   - Return a single value made of the combined first and last name with a space character separating the two values. All of this logic goes in a single `return` statement, which is terminated using a semi-colon (pipe characters add elements to the string):

```
export function getValue(): string
begin
return _ebx.record.first_name | ' ' | _ebx.record.last_name;
end
```

5. Save the script. The IDE automatically checks for any compilation errors and will display them in the **Messages** section below the script.
6. If no errors are found, select **Publish all** to publish the script.

The following image shows the result of the script where the Full name field is populated:



222.4 Creating a non-indexed function

This function example demonstrates how to create a non-indexed function. It also shows how to use an import to add built-in functionality to a script. In ESL, related functions and procedures are organized into Units. To use a unit's functions, declare the unit in a script using the following syntax: `uses core.<unit_name> as <alias>;`.

Script tip: Write out the pseudocode for more complicated functions before you start coding. The pseudocode for this example reads: Get the number of days between the last login and today; if this number is greater than 30 return false, otherwise (else) return true indicating this is an active user.

To create an non-indexed function:

1. If the EBX® Script IDE perspective is not open, click the icon at the top right of the EBX® toolbar and select **Script IDE**.
2. In the Activity Bar on the left, locate and select **Data model**.
3. In the middle Navigation Panel locate, and select the `Active` function created in the prerequisite section above. Functions are indicated by this icon.

   As shown below, the function is automatically populated based on the field's boolean data type, which returns true or false:

   ```
   export function getValue(): boolean
   begin
    return false;
   end
   ```

4. Disable indexing. As this script accesses the machine's local time, it cannot be indexed:
   1. Select the **Information** tab at the top of the script.
   2. Deselect the **Compatible with optimized search** checkbox.
   3. Return to the **Script tab**.
5. Write the logic that determines what the function will achieve. In this example, the function determines whether a user is considered active by getting the user's last login date, today's date, and calculating the difference. If the difference is less than 30 days, the user is active:

- Add the following line to the top of the script as the `core.date` unit contains the `days()` function to calculate the number of days between two dates.

```
uses core.date as date;
```

- Use an `if then else` statement to check whether a condition is true. The `if` uses the `days()` function to get the number of days since a user's last login and the `>` operator checks whether the number of days is greater than 30.

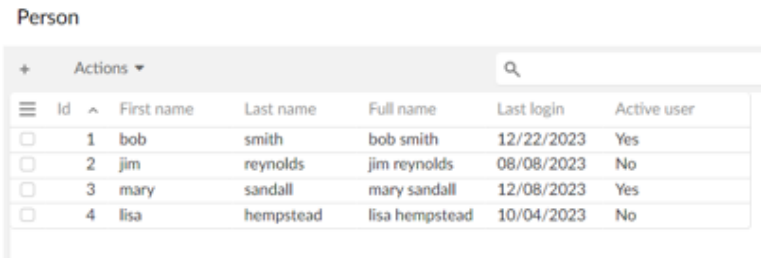
```
uses core.date as date;
export function getValue(): boolean
begin

if date.days(_ebx.record.last_login, date.now()) > 30 then
return false;
else
return true;

end
```

6. Save the script. The IDE automatically checks for any compilation errors and will display them in the **Messages** section below the script.
7. If no errors are found, select **Publish all** to publish the script.

   The following image shows the result of the script where the Active user field is populated to indicate whether a user is considered active:



CHAPTER 223

Creating triggers

This chapter contains the following topics:

1. Introduction to triggers
2. Trigger prerequisite: add trigger to data model table
3. Creating a triggered procedure

223.1 Introduction to triggers

In the EBX® Script Language (ESL), a trigger is a scripted procedure that automatically executes in response to specific events, such as changes to a record (creation, update, or deletion). You can use triggers to enforce business rules, validate data, or perform automated tasks aligned with these events. By ensuring that conditions are met, or actions are taken whenever an event occurs, it enhances data integrity and consistency.

How this tutorial is organized

This tutorial shows how to create and publish triggers in the EBX® Script IDE. It does not cover complex use cases, or all of the user interface options. To follow along, you can create the basic data model shown below (this model is a continuation of the model used in the function tutorial [p 1553]):

Data model component	Label
Data Model: `docPerson`	N/A
Table: `person`	Person
PK field, Integer (set to **Auto-increment**): `id`	Id
String field: `first_name`	First name
String field: `last_name`	Last name
String field: `full_name`	Full name
Date field: `last_login`	Last login
Boolean field: `status`	Active user
Date field: `last_modification_date`	Last modified

The parts that make up this tutorial are:

- Completing prerequisites [p 1560]
- Creating a triggered procedure [p 1561]

223.2 Trigger prerequisite: add trigger to data model table

Prior to writing the logic for a trigger in the EBX® Script IDE, you must first add a trigger to a data model element. This tutorial demonstrates adding a trigger to a table. Once you create the trigger in the data model, it is automatically available in the EBX® Script IDE.

To add a trigger to a data model table:

1. Open the `docPerson` data model and select the `person` table.
2. Select the **Advanced properties** tab and click the + icon next to **Triggers**.
3. Configure the trigger:
   - For the trigger's **Component**, select *[Built-in] EBX Script* .
   - For the trigger's **Parameters**:
     - **Script name**: Enter a name for the trigger. This name is used by the IDE to identify this trigger. For this example, enter: `lastModificationDate`.
     - **The type of trigger**: Use the menu to select the desired trigger type. For this tutorial, use: **Trigger after modification**.
4. Save and then optionally publish your data model. Note that once the table is saved after adding the trigger, the trigger's script outline is automatically created and available to work with in the IDE. However, you cannot publish the script until the data model publication is up to date. So it might be a good idea to publish the data model at this time.

   Whats next? create and publish a script [p 1561] for a triggered procedure.

223.3 Creating a triggered procedure

As procedures in ESL don't return a value, they are more suitable for triggers, which usually execute a set of instructions based on an event. The event used in this example that initiates the execution of the scripted procedure is modification of a record. The goal is to populate the Person table's Last modified field with the current date anytime a record is modified.

Script tip: use variables to hold values that can change over time, or are computed based on inputs or even other variables. This example uses variables to hold values that specify a data model, table, and PK. These values are required to update a record. Create variables using the following syntax: `var <variable_name> := <assignment>` where `var` declares the variable with a *<variable_name>*, the `:=` assigns a value, and the value can be a direct value, expression, or result of a method call.

1. If the EBX® Script IDE perspective is not open, click the  icon at the top right of the EBX® toolbar and select **Script IDE**.
2. In the Activity Bar on the left, locate and select **Data model**.
3. In the middle Navigation Panel locate the **@trigger** folder, and select the trigger created in the prerequisite steps [p 1560].

   As shown below, the trigger is based on the type of trigger selected in the prerequisite steps and set to execute after modification:

   ```
   export procedure onAfterModify()
   begin
   end
   ```

4. Write the instructions for the procedure:
   1. Import functionality from the following units: `core.data`, `core.date`, and `core.complex`. Additionally, add a reference to the data model containing the Person table, docPerson in this case:

      ```
      uses core.data as data;
      uses core.complex as complex;
      uses core.date as date;
      references model docPerson;
      ```

   2. Write the instructions for the procedure between the `begin` and `end` statements using variables to store the table, the table's primary key, and the record.
      - The `table` variable uses the `findTable()` function from the `core.data` unit to lookup a table. It requires the following input: *<dataModel.tableName>*`(_ebx.dataset)`. This example uses the `docPerson` model and the `person` table. The predefined variable `_ebx.dataset` provides required information on the dataset where the event is triggered.

        ```
        var table := data.findTable<docPerson.person>(_ebx.dataset);
        ```

      - The `core.complex` unit facilitates operations with complex data types, such as a primary key. As shown below, use the built-in `primaryKeyOf` function (from `core.complex`) for the table's primary key field. As a parameter, pass in the: *dataModel.tableName* in this case: `docPerson.person`. This creates a PK with no assigned values. The next line of code assigns the PK's id field to the value of a selected record:

        ```
        var pk := complex.primaryKeyOf<primarykeyof docPerson.person>();
        pk.id := _ebx.record.id;
        ```

- The `lookupRecordForUpdate` function from the `core.data` unit gets a modified record. Pass the current EBX® transaction as a parameter using the built-in variable: `_ebx.transaction`. Then, pass the `table` and `pk` variables that were defined above.

```
var record := data.lookupRecordForUpdate(_ebx.transaction, table, pk);
```

3. Add the modification date to the record and save it:
   - Specify which field to update using: `record.`*`fieldName`*. The field name in the example model is last_modification_date:

```
record.last_modification_date := date.now();
```

   - Save the updated record. Use the built-in `saveRecord()` function and pass the modified record as the single parameter:

```
data.saveRecord(record);
```

   - The complete procedure is:

```
uses core.data as data;
uses core.complex as complex;
uses core.date as date;
references model docPerson;

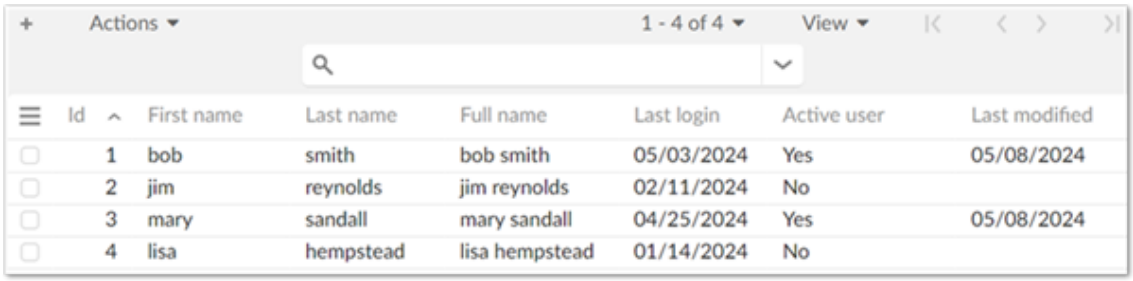
export procedure onAfterModify()
begin
    var table := data.findTable<docPerson.person>(_ebx.dataset);
    var pk := complex.primaryKeyOf<primarykeyof docPerson.person>();
    pk.id := _ebx.record.id;
    var record := data.lookupRecordForUpdate(_ebx.transaction, table, pk);

    record.last_modification_date := date.now();
    data.saveRecord(record);

end
```

5. Save the script. The IDE automatically checks for any compilation errors and will display them in the **Messages** section below the script.
6. If no errors are found, select **Publish all** to publish the script.

   The following image shows the result of the script where the Last modified field is updated each time a record is changed:



CHAPTER **224**

Creating a workflow script

This chapter contains the following topics:

1. Overview
2. Step 1: Writing and publishing the script
3. Step 2: Creating the workflow model
4. Step 3: Creating the workflow launcher and activation

224.1 Overview

Create and publish workflow script tasks directly from the EBX® Script IDE. Once a script is published, you can use it in any EBX® workflow model. For more information on workflows, see the *TIBCO EBX® User Guide* sections: *Introduction to workflow models* and *Introduction to data workflows*.

This tutorial demonstrates how to:

- Step 1 [p 1564]: Create and publish a workflow script task that includes input and output parameters. These parameters allow data to pass to and from the script.
- Step 2 [p 1566]: Create a workflow model and add the script task to a step in the model. During this task, a data context is created in the model that captures context-dependant data values. The values are populated when the workflow is launched. The values in the data context are mapped with the script's I/O parameters.
- Step 3 [p 1567]: Create and activate a workflow launcher. EBX® includes a variety of ways to initiate a workflow. This tutorial includes a simplified example to create and add a workflow launcher to a table's **Actions** menu.

Before you begin

This tutorial uses the same data model and scripts as the Creating functions [p 1553] and Creating triggers [p 1559] tutorials. To follow along with this tutorial, complete the other tutorials to have a data model in the same state, or create your own copy of the model to follow along. The following image shows the complete data model::

Data model component	Label
Data Model: `docPerson`	N/A
Table: `person`	Person
PK field, Integer (set to **Auto-increment**): `id`	Id
String field: `first_name`	First name
String field: `last_name`	Last name
String field: `full_name`	Full name
Date field: `last_login`	Last login
Boolean field: `status`	Active user
Date field: `last_modification_date`	Last modified

224.2 Step 1: Writing and publishing the script

You can create and write a workflow script directly in the Script IDE without any pre-requisites. This example shows how to add input parameters to a script, which you map with a workflow's data context. This enables data to pass to the workflow script and be processed.

Tutorial example use case: A business rule requires that no active users are deleted from the Person table. This table includes a field that indicates whether a user is active. A user launches a workflow when they want to delete a Person record. The script checks the users status. If the user is inactive, the record is deleted; if they are active, a message is written to the logs saying the record cannot be deleted.

Script tip: Make it a habit to save and compile your script at regular intervals. This helps catch any compile time errors and creates save points so that you can revert the script to a more stable state.

To write a workflow script task using the EBX® Script IDE:

1. If the EBX® Script IDE perspective is not open, click the  icon at the top right of the EBX® toolbar and select **Script IDE**.
2. In the Activity Bar on the left, locate and select **Workflow**.
3. Create a new script by selecting **Create** in the middle Navigation Panel.
4. Add required information and parameters:
    1. Enter the script name: `statusBeforeDelete`
    2. Enter the label and optional description: `Check status before delete`
    3. Add parameters to access data context components, such as a dataspace, dataset, table, and record. These values provide input to the script and are operated on by the script's logic.

       Add each of the following parameters and leave all options as default:

       - WF_Script_dataspace
       - WF_Script_dataset
       - WF_Script_table

- WF_Script_requestXPath

5. Click **Save** at the bottom of the workspace.
6. Write the logic for the workflow script task in the **Script tab**.
   1. Add the imports:

      This script leverages pre-built functions included in the following EBX® Script language (ESL) units: `core.data`, `core.complex`, and `core.log`. The syntax to declare a unit is: `uses core.<unit_name>;`. A reference to the data model is also a required part of the data context for this workflow example. The syntax to reference a model is: `references model <data_model_name>;` The following example shows the imports:

      ```
      uses core.data as data;
      uses core.complex as complex;
      uses core.log as log;

      references model docPerson;
      ```

   2. Create constants to hold the values based on user selection:

      When a user selects a record and starts the workflow from a table's **Actions** menu, information about the selected record is passed to the workflow script's input parameters. The constants must hold the: dataspace, dataset, table, and selected record. Think of it as a parent/child relationship; beginning with the dataspace, you declare the parent and when you declare a child, the parent is passed as input. The following code is added after the `begin` statement in the `executeScriptTask()` procedure:

      ```
      // Create constants to hold the dataspace, dataset, and table
          const dataspace := data.findDataspace(_ebx.parameters.WF_Script_dataspace);
          const dataset := data.findDataset(dataspace, _ebx.parameters.WF_Script_dataset);
          const table := data.findTable<docPerson.person>(dataset);
      ```

   3. Use a transaction to perform actions on selected data:

      It is imperative to use a transaction when operating on context dependent data. A transaction ensures data consistency by isolating affected data and only making changes if the transaction is completed. In ESL, transactions are executed on dataspaces and have the following syntax: `execute transaction <transaction_name> on <dataspace>` where <*transaction_name*> is the variable name assigned to the transaction and <*dataspace*> is the dataspace name where the transactions occurs. After initiating the transaction, the sample code:

      - Gets the current record using its XPath.
      - Creates a complex object to store the selected record's primary key.
      - Assigns the currently selected record's id to the PK's `id` attribute.

      ```
      // Initiate transaction
       execute transaction tr on dataspace
       begin
        // Get current record using XPath
        var currRecord := data.lookupRecordByFullXPathForUpdate(tr, table,
       _ebx.parameters.WF_Script_requestXPath);
        // Create 'primary key' object
        var pk := complex.primaryKeyOf<docPerson.person>();
        // assign the value from the currently selected record's 'id' field to the objects 'id' parameter
        pk.id := currRecord.id;
       end
      ```

   4. Write the logic to check the user's status and either delete the record or write a message to the log:

      In the example, an `if then else` statement passes in the selected record's `status` field. A status of `false` means the user is inactive and can be deleted, calling the `deleteRecordByPrimaryKey()` function. The `else` only executes if the user is active and sends

an info type log message. To view EBX® Script log messages, navigate to: *Administration > Scripting management > Script logs*.

```
  // If the user is considered 'not active', then delete
  // If they are active write an info message instead
  if currRecord.status = false then
  begin
   data.deleteRecordByPrimaryKey(tr, table, pk);
  end
  else
  log.info('Active users cannot be deleted.');
```

The full workflow script task example:

```
uses core.data as data;
uses core.complex as complex;
uses core.log as log;

references model docPerson;

export procedure executeScriptTask()
begin
// Create constants to hold the dataspace, dataset, and table
const dataspace := data.findDataspace(_ebx.parameters.WF_Script_dataspace);
const dataset := data.findDataset(dataspace, _ebx.parameters.WF_Script_dataset);
const table := data.findTable<docPerson.person>(dataset);

    // Initiate transaction
    execute transaction tr on dataspace
    begin
    // Get current record using XPath
    var currRecord := data.lookupRecordByFullXPathForUpdate(tr, table,
 _ebx.parameters.WF_Script_requestXPath);
    // Create 'primary key' object
    var pk := complex.primaryKeyOf<docPerson.person>();
        // assign the value from the currently selected record's 'id' field to the objects 'id'
 parameter
        pk.id := currRecord.id;

        // If the user is considered 'not active', then delete
        // If they are active write an info message instead
        if currRecord.status = false then
        begin
        data.deleteRecordByPrimaryKey(tr, table, pk);
        end
        else
        log.info('Active users cannot be deleted.');

        end
        end
```

7. Publish the script to make it available for use in a workflow model using the **Publish** button at the bottom of the **Script tab**.

What's next?: Create the workflow model [p 1566] that will use the script you wrote in the steps above.

224.3 Step 2: Creating the workflow model

A workflow model definition includes steps and their order of execution. These models are highly versatile and can range from straightforward, single-step processes to complex, multi-layered sequences. This tutorial focuses on the interaction between a script written in the IDE and a workflow model. A simple workflow model example is used to achieve this.

To create a workflow model:

1. In the main EBX® toolbar, select **Workflow models** and then click **Create a workflow model**.
2. Provide a unique name that does not include spaces or special characters. To follow along with this example, use `deletePerson`.
3. Create a data context:

   A data context is used to capture context-dependant data values, such as a selected record's id.

1. Navigate to *Workflow modeling* > *Workflow model configuration* > *Data context*.
2. Click the '+' icon to add the following list of variables and set each variable's **Input parameter** to **Yes**:
   - WF_Variable_XPath
   - WF_Variable_dataspace
   - WF_Variable_dataset
   - WF_Variable_table

   Each variable gets mapped to a parameter from the workflow script. The names should be similar to the script's parameter names, but it might be a good idea to include additional descriptive information. This helps you differentiate between workflow model variables and script parameters.

4. Add a step to the workflow model that executes the workflow script task:
   1. Navigate to *Workflow modeling* > *Workflow model steps* and in the workflow diagram.
   2. Hover your mouse over the line between **Initial transition** and **Final step**, and click the + icon.
   3. Provide the task name: `deletePersonScript`
   4. Select **Script** from the **Step type** menu, and click **Next**.
   5. Configure the script:
      1. Leave the **Hidden in progress view** and **Progress strategy** at their default values.

         For the **Script** group, set the **Input mode** to **Script**.
      2. Use the **Script name** menu to select the `statusBeforeDelete` script.

         A list of the script's input parameters displays.
   6. Use the wand icon to map each input parameter (these come from the workflow script) with the corresponding variable from the data context created in step 2.
   7. Click **Create**.
   8. Save and close the step.
   9. Select **Publish** to initiate the publication process and if the information displayed looks OK, select **Publish** at the bottom of the page.

224.4 Step 3: Creating the workflow launcher and activation

A workflow must be initiated by a workflow launcher. You can configure a workflow launcher in several ways. For the sake of brevity, this example demonstrates how to create a workflow that users can launch directly from a table's **Actions** menu. This requires creation of a workflow launcher and workflow activation.

To create a workflow launcher and activation:

1. Navigate to: *Administration* > *Workflow management* > *Workflow launchers* > *Launchers*.
2. Create a new record:
   1. Enter the launcher name: `DeleteInactiveUser`

2. Use the **Workflow publication name** menu to select the `deletePerson` workflow model publication.

   The **Data context variables** group displays below.

3. For **Documentation**, add the name displayed in the table menu: `Delete an inactive user`
4. Map the data context variables with the EBX® built-in data context types. These values are automatically populated based on the active context when a users selects data to delete and launches the workflow.
5. Save, but do not close the record.

   The **Launcher activations** section displays.

3. Create a workflow launcher activation

   A workflow launcher activation determines the accessibility of a workflow launcher. The launcher is available to the selected profiles when they are viewing the specified tables.

   1. Click + to create a new launcher activation.
   2. Select the `docPerson` data model.
   3. Select the `Person` table.
   4. For **Profiles**, ensure your profile is added.
   5. Save and close.

4. Save and close the record.

   The workflow that uses the script we created can now be launched from the `Person` table's **Actions** menu.

CHAPTER 225

Creating scripted REST services tutorial

This chapter contains the following topics:

1. Introduction to scripted REST services
2. Prerequisites
3. Creating a scripted REST service
4. Example scripted REST services for CRUD operations

225.1 Introduction to scripted REST services

A scripted REST service is the combination of a REST endpoint definition and its related script. Scripts handle programmatic actions for REST services and are written in the EBX® Script language. REST service definitions created in the EBX® Script IDE conform to a subset of the OpenAPI Specification (OAS). This allows you to import and export definitions that conform the OAS v3 standard.

During implementation use the options in the UI to define:

- Paths
- Operations
- Path parameters
- Query parameters
- Body and Responses
- Data types

Each REST service script can include multiple functions to handle logic for multiple endpoints and REST methods. After implementing and publishing the scripted REST service, it is accessible by an external consumer. Note that only JSON requests and responses are supported.

How this tutorial is organized

This tutorial shows how to create and publish scripted REST services in the EBX® Script IDE. It does not cover complex use cases, or all of the user interface options.

The parts that make up this tutorial are:

- Completing prerequisites [p 1570]

- Creating a scripted REST service [p 1571]
- Scripted REST service examples [p 1573]

225.2 Prerequisites

Before beginning this tutorial take a moment to review the following:

- Understanding the anatomy of a scripted REST service [p 1570]
- Tutorial data model [p 1570]

Understanding the anatomy of a scripted REST service

There are 2 main components to a scripted REST service:

- The *REST API definition* of the endpoint including components such as: paths, HTTP methods (GET, POST, PUT, DELETE), query parameters, access, and responses.
- The *script* where functions are written with the programmatic logic to: query or update data, or validate business rules or constraints. The function type called by a REST service must be an EBX® Script export function.

> **Attention**
> The bridge between the REST API and script is the **Operation id**. You must use the same value for the function name in your script as for the **Operation id** property when defining the REST API. It's this Id that allows EBX® to know what function to call when an endpoint is accessed.

Whats next? create and publish the tutorial's data model [p 1570].

Tutorial data model

To follow along, you can create the basic data model shown below:

Data model component	Label
Data Model: `userDirectory`	N/A
Dataspace: `Reference`	N/A
Dataset: `directoryDataset`	N/A
Table: `users`	Users
PK field, Integer (set to **Auto-increment**): `id`	Id
String field: `first_name`	First name
String field: `last_name`	Last name

Whats next? create and test a scripted REST service [p 1571].

225.3 Creating a scripted REST service

The goal of this scripted REST service is to create an endpoint and its related script that returns a list of user records from our Users table.

1. If the EBX® Script IDE perspective is not open, click the  icon at the top right of the EBX® toolbar and select **Script IDE**.
2. In the Dock on the left of the screen locate and select **REST service**.
3. Click the + icon to create a new script and after providing a name, save the script.
4. Create the REST path and operation:
   1. Select the **API** tab and create a new path.
   2. Enter `/users` for the **Path template** name and click to confirm.

      > **Note**
      >
      > Although not shown here, use curly brackets separated by forward slashes to add parameters. For example: `/somePath/{param1}/{param2}`.

   3. Create a new operation under the **GET** tab.
   4. Enter an **Operation id**, or optionally select **Generate** to automatically create an id. For this tutorial, we'll use the value `getUsers`.

      > **Attention**
      >
      > The **Operation id** serves as the map between this endpoint and its corresponding script function.

   5. Save the API.
5. Build a data type for the operation response:
   1. Select the **DATA TYPES** tab.
   2. Add a new Data type with the name `Person`.
   3. Add the following properties that will be mapped to data model fields in the script:
      - **Name**: `id`, **Type**: decimal
      - **Name**: `firstName`, **Type**: string
      - **Name**: `lastName`, **Type**: string
   4. Save the data type.
6. Add the data type to the REST response:
   1. Open the **API DEFINITION** tab.
   2. Expand the `/users` path.
   3. In the **Responses** group, choose `Person` from the **Type** menu and tic the **Is Array** box.
   4. Save the API.
7. Create the script:

1. On the right side of the **Operation id** field, click the **SCRIPT** button to automatically generate a function template with the same name as the operation id.

   The generated function is shown below:

```
export function getUsers(): rest_response
begin
//Replace the following code with your implementation.

  var object1 := instanceOf<User>();
  //Set object1 properties.
  var object2 := instanceOf<User>();
  //Set object2 properties.

  var content := list.of(object1, object2);
  return rest.responseOf(200, content);
end
```

2. Before writing the logic for our function, create helper methods to retrieve the dataspace and dataset where we want this script to run. Note that in the following example, you'll need to provide the names of your dataspace and dataset, if different from `Reference` and `userDirectory`, respectively:

```
uses core.data as data;

//Helpers to retrieve a specific dataspace and data set
function getDirectoryDataspace(): dataspace
begin
 return data.findDataspace('Reference');
end

function getDirectoryDataset(dataspace: dataspace): dataset
begin
 return data.findDataset(dataspace, 'userDirectory');
end
```

3. Write a script to handle the service's business logic.

   The following sample is annotated to describe the function's implementation:

```
//Import core units
//The core.rest unit methods are used in REST services
uses core.rest as rest;
//The core.list unit methods enable creating and updating lists
//We need this as the return type we specified requires an Array
uses core.list as list;
//The core.data unit provides database CRUD operations
uses core.data as data;

//Create a model reference
//Be sure to supply your model's name, ours is "userDirectory"
references model "userDirectory" as dir;

export function getUsers(): rest_response
begin
    //Set the dataset location using the helper methods
 const directory := getDirectoryDataset(getDirectoryDataspace());

    //Create the SQL query to retrieve the desired response data
    //Be sure to use the correct field and table names defined in your data model
 var query := sql `select u.id, u.firstName, u.lastName from dir."users" u order by u.id`;
    //Add the dataset to the query
 data.addDataset(query, 'dir', directory);

    //Create a list for the 'Person' data type
 var listOfUsers := list.of<Person>();

    //Create a variable to store the query result
 var queryResult := data.executeQuery(query);
    //Loop over, and add query results to the 'listOfUsers'
 for tuple in queryResult do
 begin
        //Create an Object of type 'Person'
  var user := instanceOf<Person>();
        //Map each Object property with the corresponding query result
  user.id := tuple.id;
  user.firstName := tuple.firstName;
  user.lastName := tuple.lastName;
```

```
        //Add each user to the list
  list.add(listOfUsers, user);
 end;

    //Return the optional REST response code and the list of users
 return rest.responseOf(200, listOfUsers);
end
```

8. Save the script. The IDE automatically checks for any compilation errors and will display them in the **Messages** section below the script.
9. If no errors are found, select **PUBLISH**.
10. Test the service using the Script IDE's built-in testing capabilities. Alternatively, you can test using a tool such as Postman:
    1. In the upper-right click **TEST API**.
    2. Select the API to test, `users` in our case.
    3. Click **Test Request** and then **Send** in next screen that displays.

       As shown below, the response Body should include all records in the users table:



Whats next? Build on what you've learned in the tutorial by creating additional scripted CRUD operations [p 1573].

225.4 Example scripted REST services for CRUD operations

This section contains examples of scripted REST services for various CRUD operations. Each example includes the path, REST methods, operation ids, and data types used. The samples build on the above tutorial and are meant to demonstrate how to implement some common uses cases. Be sure to review the Examples prerequisites [p 1574] before testing these examples.

Examples prerequisites

If you want to use these examples with your existing data, you'll need to update the sample code with the appropriate data model, dataspace, and dataset information. To use the sample code as it is set up your environment as follows:

- Data model [p 1574]
- Data types [p 1574]
- Shared code template [p 1574]

Data model

The sample code is based on the same data model [p 1570] used in the tutorial. This is reflected in any referenced data model component naming in the code samples.

Data types

Ensure you have the following types create under the **DATA TYPES** tab:

Data types	Properties
`Person`	• `id` of type **decimal** • `firstName` of type **string** • `lastName` of type **string**
`Error`	`message` of type **string**

Shared code template

The following code is required for all of the following examples. It includes required core units, data model reference, and two helper functions. Later examples reference this template and show only the code that is specific to each operation.

```
//Required core units
uses core.rest as rest;
uses core.list as list;
uses core.data as data;
uses core.complex as cp;

//Specifies the data model as 'userDirectory'
references model "userDirectory" as dir;


//Helper to retrieve the 'Reference' dataspace
function getDirectoryDataspace(): dataspace
begin
 return data.findDataspace('Reference');
end

//Helper to retrieve the 'directoryDataset' dataset
function getDirectoryDataset(dataspace: dataspace): dataset
begin
 return data.findDataset(dataspace, 'directoryDataset');
end
```

Create a user

Use the following API definition and script to create a user.

API definition

Path	`/users`
Path parameters	N/A
REST Method	POST
Operation id:	`createUser`
Query parameter	N/A
Body	**Type**: `Person` **Required**: `Yes`
Responses:	**Type**: `Person`

Script

The following script includes 2 functions, 1 procedure, and does the following:

- Checks whether the user specified in the request Body exists.
- If the user does not exist, the user is created in the database.
- If the user does exist, an error message is returned. No data is updated.

```
export function createUser(const user: Person): rest_response
begin
 ref response: rest_response;

 execute transaction tr on getDirectoryDataspace() as system
 begin
  response := doCreateUser(tr, user);
 end

 return response;
end

function doCreateUser(tr: transaction, user: Person): rest_response
begin

 const directory := getDirectoryDataset(tr.dataspace);
 const table := data.findTable<dir.users>(directory);

 var pk := cp.primaryKeyOf<dir.users>();
 pk.id := user.id;

 var record := data.lookupRecord(table, pk);
 if isNotNull(record) then
 begin
  var error := instanceOf < Error > ();
  error.message := 'User "' | user.id | '" already exists';
  return rest.responseOf(400, error);
 end

 tr.isAllPrivileges := true;
 createOrUpdateUser(tr, user.id, user.firstName, user.lastName);

 return rest.responseOf(201, user);
end

procedure createOrUpdateUser(tr: transaction, id: int, firstName: string, lastName: string)
begin
```

```
 const directory := getDirectoryDataset(tr.dataspace);
 const table := data.findTable<dir.users>(directory);

 var pk := cp.primaryKeyOf<dir.users>();
 pk.id := id;
 var user := data.lookupRecordForUpdate(tr, table, pk);
 if isNull(user) then
 begin
  user := data.recordOf(tr, table);
  user.id := id;
 end

 user.firstName := firstName;
 user.lastName := lastName;
 data.saveRecord(user);
end
```

Getting a user by id

Use the following API definition and script to return a user using their id.

API definition

Path	`/users/{id}`
Path parameters	**Name**: id **Type**: **integer**
REST Method	GET
Operation id:	`getUser`
Responses:	**Type**: `Person` **Is Array**: checked

Script

The following script queries the database for a user, and returns an error message if that user is not found.

```
export function getUser(id: int): rest_response
begin
 const directory := getDirectoryDataset(getDirectoryDataspace());

 var query := sql `select u.id, u.firstName, u.lastName from dir."users" u where u.id = ?`;
 data.addDataset(query, 'dir', directory);
 data.setParameter(query, 0, id);

 var tuple := data.fetchOne(query);
 if isNull(tuple) then
 begin
  var error := instanceOf <Error>();
  error.message := 'User not found';
  return rest.responseOf(404, error);
 end

 var user := instanceOf<Person>();
 user.id := tuple.id;
 user.firstName := tuple.firstName;
 user.lastName := tuple.lastName;

 return rest.responseOf(200, user);
end
```

Update a user

Use the following API definition and script to update a user.

API definition

Path	`/user`
Path parameters	N/A
REST Method	PUT
Operation id:	`updateUser`
Query parameter	N/A
Body	**Type**: `Person` **Required**: `Yes`
Responses:	None. Check **No Response Body**

Script

The following script includes 2 functions and does the following:

- Checks if the user specified in the request Body exists.
- If the user does not exist, an error is returned. No data is updated.
- If the user does exist, their record is updated with the supplied data.

```
export function updateUser(const user: Person): rest_response
begin
 ref response: rest_response;

 execute transaction tr on getDirectoryDataspace() as system
 begin
  response := doUpdateUser(tr, user);

 end

 return response;
end

function doUpdateUser(tr: transaction, user: Person): rest_response
begin
 const directory := getDirectoryDataset(tr.dataspace);
 const table := data.findTable<dir.users>(directory);

 var pk := cp.primaryKeyOf<dir.users>();
 pk.id := user.id;

 var record := data.lookupRecordForUpdate(tr, table, pk);
 if isNull(record) then
 begin
  var error := instanceOf < Error > ();
  error.message := 'User "' | user.id | '" not found';
  return rest.responseOf(404, error);
 end

 record.firstName := user.firstName;
 record.lastName := user.lastName;
```

```
 data.saveRecord(record);

 return rest.responseOf(200, user);
end
```

Deleting a user by id

Use the following API definition and script to delete a user using their id.

API definition

Path	`/users/{id}`
Path parameters	**Name**: id **Type**: **integer**
REST Method	DELETE
Operation id:	`deleteUser`
Responses:	**No Response Body**: checked

Script

The following script queries the database for a user, and deletes them if the user is found.

```
export function deleteUser(const id: int): rest_response
begin
 execute transaction tr on getDirectoryDataspace() as system
 begin
  const directory := getDirectoryDataset(tr.dataspace);
  const table := data.findTable<dir.users>(directory);

  var pk := cp.primaryKeyOf<dir.users>();
  pk.id := id;

  var record := data.lookupRecord(table, pk);
  if isNotNull(record) then
  begin
   data.deleteRecord(tr, record);
  end
 end

 return rest.emptyResponse(204);
end
```

CHAPTER 226

User interface reference

This chapter contains the following topics:

1. Overview
2. Dock
3. Navigation Panel
4. Workspace

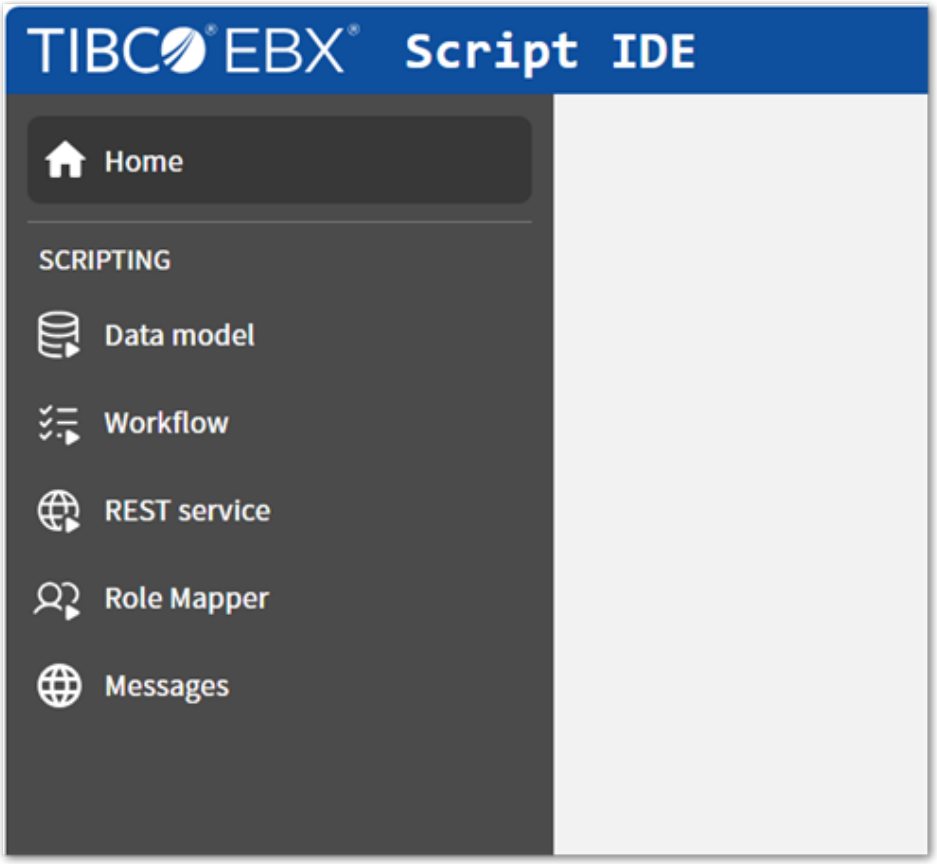
226.1 Overview

The EBX® Script IDE includes the following main UI components:

- Dock: Allows you to access the different types of script workspaces (**Data model**, **Workflow**, **REST service**, **Role Mapper**, and **Messages**).
- Navigation Panel: Depending on the type of script workspace you're viewing, you can browse, select, create, import, and export.
- Workspace: Location where you can write scripts, build REST endpoints, etc.

226.2 **Dock**

The Dock is the left-most component on the screen. It allows you to access the available scripting workspaces. Each workspace is specially tailored to one type of scripting. The Dock is resizeable and includes the following:

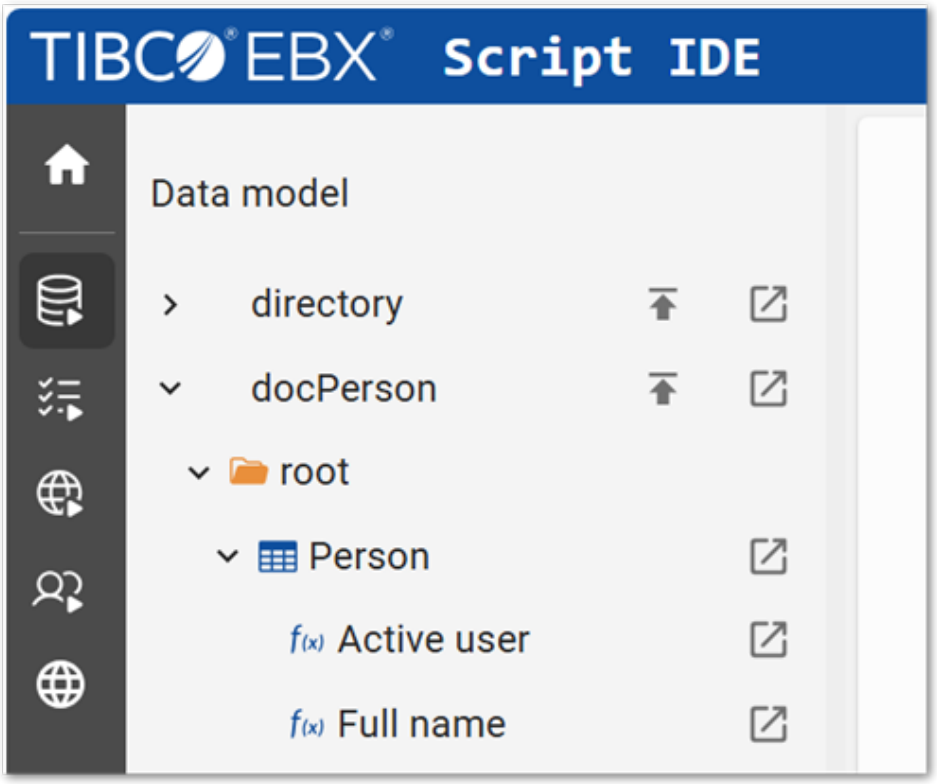


226.3 **Navigation Panel**

The Navigation Panel is the middle UI component and functions much like a file explorer. It's where you can browse and select your scripts. Each type of script has designated quick action icons that:

- Create, delete, and publish.
- Take you to specific components in the Data Model Assistant (DMA).

- Import and export.



The following table describes available icons and actions in the **Navigation pane**:

	The **Publish all** icon publishes all scripts in a given data model.
	The **View in DMA** icon opens the Data model assistant (DMA) with the selected item in focus.
	The **Create** button is available for workflows, REST services, the Role Mapper, and Messages.
	The **Delete** icon workflows, REST services, the Role Mapper, and Messages.
	The **Import** icon allows you to import a script.
	The **Export** icon allows you to export a script.

226.4 **Workspace**

The Workspace is the main area where you write and edit scripts; it's where the coding takes place. For scripted REST services, it's also where you can create the REST API definition.

The Workspace contains the following tabs and sections:

- **SCRIPT** tab: You can write, edit, and view your scripts in this tab. It includes the following features:
  - Code completion
  - Hover for tooltips/help on individual components
  - Syntax highlighting
  - Compilation and error detection
  - Script saving and publishing

  The **SCRIPT** tab includes a toolbar at the top with icons to call out various actions. Hover your mouse over each icon for a brief description of each.

  The main body of the **SCRIPT** tab is where the actual script is written. At the bottom is an expandable messages section that displays information after saving, or compiling the script. Below the messages are the following buttons:
  - **SAVE**: automatically compiles and then saves the script.
  - **COMPILE**: complies the script without saving it.

- **PUBLISH ALL**: publishes all scripts contained in the related data model.



- **INFORMATION** tab: Displays details about the selected script. It is also where you specify whether capable scripts are indexed for search optimization.
- **PARAMETERS** tab (workflow scripts only): Allows you to add and manage parameters applied to the script.
- **API DEFINITION** tab (REST service scripts only): Allows you to define a REST endpoint, including its paths, path parameters, method, query parameters, body, and response.

- **DATA TYPES** tab (REST service scripts only): Enables you to create data types to use with API definitions. Data types can also be used when creating the script for a REST service.
- **API HISTORY** tab (REST service scripts only): Displays a list of API definition revisions and allows you to compare the differences between the current revision.
- **HISTORY** tab: Efficiently manage script revisions in this tab. It offers a clear, chronological overview of all saved changes to your script. Each entry in the history is a snapshot of a specific change, allowing you to quickly understand how your script has evolved over time.

  Rollback functionality: Use the arrows to undo individual changes and rollback your script to an earlier state.

Migration Guide

CHAPTER 227

Migration introduction

This chapter contains the following topics:

1. Overview
2. Migration roadmap

227.1 Overview

Repository migration from TIBCO EBX® 5.9 to 6 is automatic on server startup. However, you must perform several technical and functional tasks prior to startup to ensure the automatic migration is successful. Some of these tasks include:

- Backing up and cleaning your existing 5.9 environment. Please note that this guide is intended for use during migration from 5.9 to 6. However, it is critical that you back up your existing repository even when upgrading from an older version of EBX® 6.
- Addressing custom modules.
- Understanding the new scalable architecture and its impact on custom Java code performance.
- Aligning with new hardware and memory requirements.

> **Note**
>
> The version numbering conventions used in this document are as follows:
>
> - 5.9 refers to any release from 5.9.0 up to, and including, the most recent service pack for 5.9.
> - 6 refers to any release from 6.0.0 up to, and including, the most recent release.
> - 6.0 refers to any release from 6.0.0 up to, and including, the most recent service pack for 6.0.
> - 6.x refers to any release from 6.1.0 up to, and including, the most recent release.

Likely the tasks mentioned above will be performed by different roles in your organization, in different environments (dev, pre-prod, prod, etc.), and using multiple versions of your repository. Note that some tasks must be performed multiple times; once in each environment. See the following to get started:

- Migration roadmap [p 1587] outlines the different phases of the migration process and provides links to the relevant documentation.

- Environment and hardware requirements [p 1589] outlines the impacts of migration on existing hardware.
- Changes to supported add-ons [p 1593] provides details on the add-ons migrated to EBX® 6, those with re-worked versions, and those that were discontinued.

Additional documentation resources

Review the following documentation for additional information related to the migration process:

- In the *All release notes* sections available at docs.tibco.com, carefully review the *Backwards compatibility* sections for the releases leading up to the version you are migrating to. These sections contain important information that can assist your migration efforts.
- In the *TIBCO EBX® Administration Guide*, read the *Installation and configuration* section. This section contains performance tips, hardware guidelines, deployment information, and environment-specific installation instructions.

227.2 **Migration roadmap**

The exact migration process depends on your organization's business requirements and implementation of TIBCO EBX®. What follows is a general guideline that addresses common

requirements and provides a foundation that you can build on and adapt to address your specific needs. The links direct you to the relevant sections in the *Migration Guide*:

Step 1) Local environment (Developers)	This phase involves: • Downloading EBX® 6 JARs • Understanding new EBX® 6 architecture • Refactoring custom code to leverage performance and ensure compatibility • Compiling custom code with new JARs • Running a test migration • Generating binaries See Step 1: Adapting custom code [p 1599] for detailed steps.
Step 2) Pre-production environment (Dev Sec Ops)	This phase involves: • Preparing a *pre-production* version of the repository. See Step 2: Preparing the repository [p 1603] for detailed steps. • Understanding new environment and hardware requirements. See Environment and hardware requirements [p 1589] for more information. • Deploying and testing in pre-production environment. See Step 3: Deploying [p 1611] for detailed steps.
Step 3) Production environment (Dev Sec Ops)	This phase involves: • Preparing the *production* version of the repository. See Step 2: Preparing the repository [p 1603] for detailed steps. • Ensuring the production environment complies with the new environment and hardware requirements. See Environment and hardware requirements [p 1589] for more information. • Deploying and testing in the production environment. See Step 3: Deploying [p 1611] for detailed steps.

CHAPTER 228

Environment and hardware requirements

This chapter contains the following topics:

1. Supported environments
2. Hardware resources

228.1 Supported environments

Only an EBX® 5.9 repository can be migrated to EBX® 6. If you are using an earlier version, the repository must first be migrated to version 5.9. Additionally, migration for 6.0 Beta release repositories to a 6 GA version is not supported. We always recommend migrating to the latest EBX® release.

See Supported environments [p 524] for more details on the necessary upgrades show below:

Browser support	Please note that Microsoft Internet Explorer and the non-Chromium version of Microsoft Edge are not supported anymore.
JVM support	Required Java Runtime Environment: either JRE 17, 21, or 25 LTS. Java 8 and 11 are no longer supported.
IBM support	WebSphere Application Server 9 is no longer supported. WebSphere Application Server Liberty, for Java EE 8 only, is supported from versions 6.0.0 up to, but excluding, 6.2.0. Subsequently, from version 6.2.0 onwards and due to Jakarta EE 10+ migration, only Websphere Application Server Liberty 23, or higher, for Jakarta EE 10, is supported.
Oracle support	Oracle 12c and 21c are no longer supported. The minimum supported version of Oracle is now 19c, 23c or higher. WebLogic Server 15c is supported.
PostgreSQL support	PostgreSQL 10 to 13 are no longer supported. The minimum supported version of PostgreSQL is now 14 and the latest supported version is 18.
Microsoft support	SQL Server 2014 is no longer supported. The minimum supported version of SQL Server is now 2016.

228.2 Hardware resources

You might need to upgrade your hardware as EBX® 6 introduces Apache Lucene to manage data indexes. Indexes are now offloaded from memory to persistent storage. In addition to the high-level

recommendations in the following table, you should carefully review the information contained in Performance and tuning [p 615]:

	Disk	Memory
Recommended	• Ensuring sufficient disk space is available to persist the Lucene indexes. • Using a high-performing drive such as an SSD. • If using a Storage Area Network (SAN), pairing it with a other high performance technology such as Fiber Channel (FC). See the performance section on disks [p 531] for more information regarding disk requirements.	• Lowering memory allocation to the Java Virtual Machine (JVM). • Ensuring memory allocation to the OS is sufficient. • Tuning the garbage collector to improve performance. See the performance section on memory [p 615] for more information regarding memory requirements.
Not recommended	• Sharing the EBX® root directory with other EBX® instances. This includes sharing between the active and standby instances when failover with hot standby is configured. • Storing the root directory, which contains the Lucene indexes, on a network attached volume (NAT). • Using anti-virus software on the root directory. See the performance section on disks [p 531] for more information regarding disk requirements.	

CHAPTER 229

Changes to supported add-ons

This chapter contains the following topics:

1. Overview
2. Manually migrating add-ons
3. Data Model and Visualization Add-on additional steps
4. Digital Asset Manager Add-on additional steps

229.1 Overview

Some add-ons were migrated to EBX® 6, others were replaced with re-worked versions, and a few were phased out. You can find comprehensive details of these updates and discontinuations in the

table below. If you have any data model dependencies on removed or replaced add-ons, these must be addressed prior to initiating the automatic migration process.

Add-on	EBX® status and migration requirements
TIBCO EBX® Match and Merge Add-on (Previously Match and Cleanse (DAQA))	This add-on was discontinued and replaced by a reworked version in EBX® 6: the TIBCO EBX® Match and Merge Add-on (MAME). The the configuration for DAQA is not migrated. If you used this add-on in TIBCO EBX® 5.9, you must follow the instructions in Includes for discontinued add-ons [p 1607] to ensure no artifacts remain that could prevent migration.
TIBCO EBX® Match and Merge Add-on (MAME)	This add-on is a reworked version of, and replaces, the TIBCO EBX® Match and Merge Add-on (previously Match and Cleanse DAQA). Since this add-on is new, no special consideration is required during the migration process.
TIBCO EBX® Data Exchange Add-on (ADIX)	In EBX® 6 there is a reworked version of this add-on: the TIBCO EBX® Data Exchange Add-on (New) (DINT). The legacy add-on's configuration information is automatically migrated. Even so, it is good practice to export a back up of the configuration.
TIBCO EBX® Data Exchange Add-on (New) (DINT)	This add-on is a reworked version of the TIBCO EBX® Data Exchange Add-on (ADIX). Since this add-on is new, no special consideration is required during the migration process. Note that the legacy version of this add-on is still available in EBX® 6. To help designate between versions of the add-on, the menus in the UI append (New) to the reworked version.
TIBCO EBX® Insight Add-on (DQID)	This add-on is available in EBX® 6 and its configuration information is automatically migrated. A reworked version of this add-on is available, the TIBCO EBX® Insight Add-on (New) (DPRA). Note that if you do not deploy the legacy version of the add-on, you should remove any related Java extensions. See Java extensions for discontinued add-ons [p 1608] for additional information.
TIBCO EBX® Insight Add-on (New) (DPRA)	This add-on is a reworked version of the TIBCO EBX® Insight Add-on (DQID). Since this add-on is new, no special consideration is required during the migration process. Note that the legacy version of this add-on is still available in EBX® 6. To help designate between versions of the add-on, the menus in the UI append (New) to the reworked version.
TIBCO EBX® Information Search Add-on (TESE)	Some of this add-on's functionality is now included in the core product and its configuration is now significantly different. The add-on's configuration is not automatically migrated.
TIBCO EBX® Digital Asset Manager (DAMA)	The add-on's configuration is automatically migrated. APIs that were deprecated in earlier versions are no longer available. Therefore, code review might be required to replace the deprecated APIs with their new versions.
TIBCO EBX® Data Model and Data Visualization (DMDV)	The add-on's configuration is not automatically migrated. You must export an archive of the configuration and re-import it after successfully completing the migration process. See Manually migrating add-ons [p 1595] for the steps to perform this task. There is a new feature that defines link orientation. This is a required field and must be defined for all links after migration. You can set it once and then apply the modification to other records. See the instructions here [p 1596] for more details.
TIBCO EBX® EBX GO Add-on (MODA)	This add-on's configuration is not automatically migrated. You must export an archive of the configuration file and import it after successful migration. See Manually migrating add-ons [p 1595] for the steps to perform this task.
TIBCO EBX® Rules Portfolio Add-on (RPFL)	This add-on was discontinued. If this add-on was previously used, you should remove any related Java extensions. See Java extensions for discontinued add-ons [p 1608] for additional information.

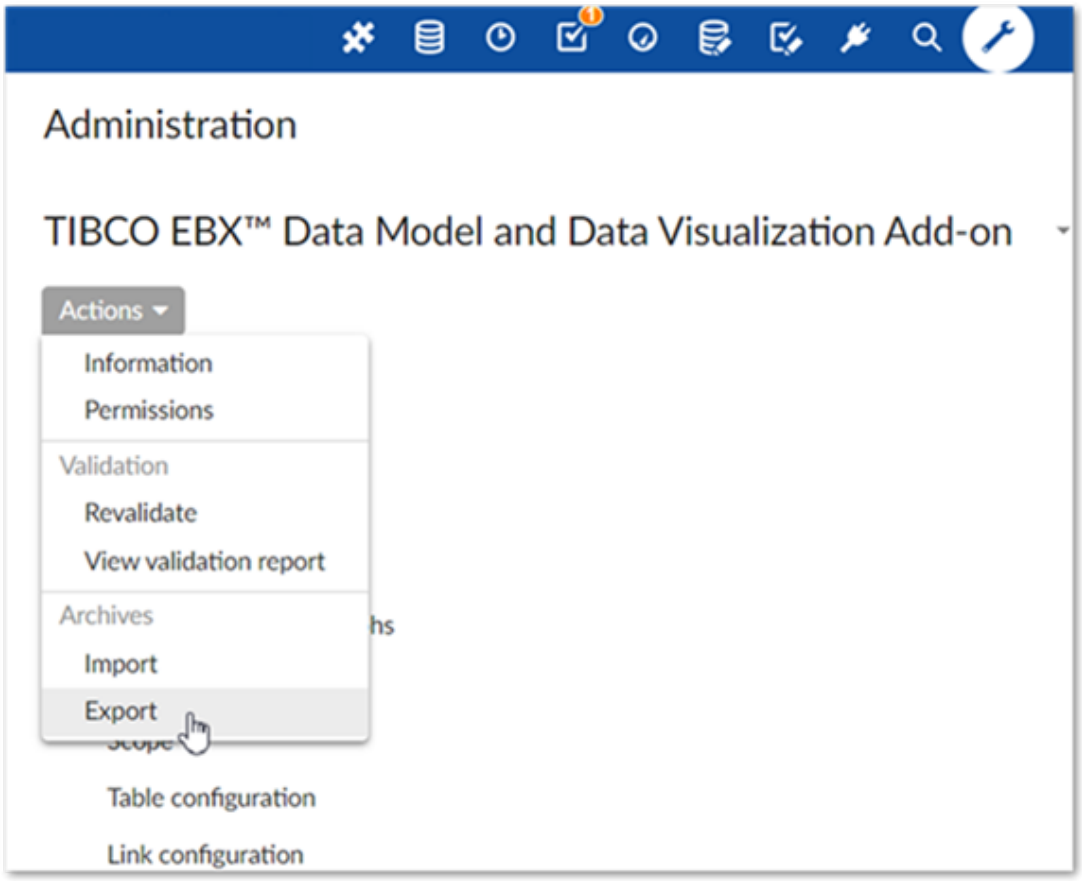
Add-on	EBX® status and migration requirements
TIBCO EBX® Graph View Add-on (`GRAM`)	This add-on was discontinued.
TIBCO EBX® Information Governance Add-on (`IGOV`)	This add-on was discontinued. If this add-on was previously used, you should remove any related Java extensions. See Java extensions for discontinued add-ons [p 1608] for additional information.
TIBCO EBX® Oracle Hyperion EPM Add-on (`HMFH`)	This add-on was discontinued.
TIBCO EBX® Activity Monitoring Add-on (`MTRN`)	This add-on was discontinued.

229.2 Manually migrating add-ons

To manually migrate add-on configurations, perform the following actions:

1. Export an archive of the add-on's configuration:
    1. Navigate to the desired add-on:
        - *Administration > User interface > TIBCO EBX® Data Model and Data Visualization Add-on*
        - *Administration > TIBCO EBX® GO Add-on > Configuration*

2. As shown in the following image, from the **Actions** menu, select *Archives* > *Export*.



3. Supply a name for the archive, leave the default settings, and select **Export**.

   No messages display upon successful export.

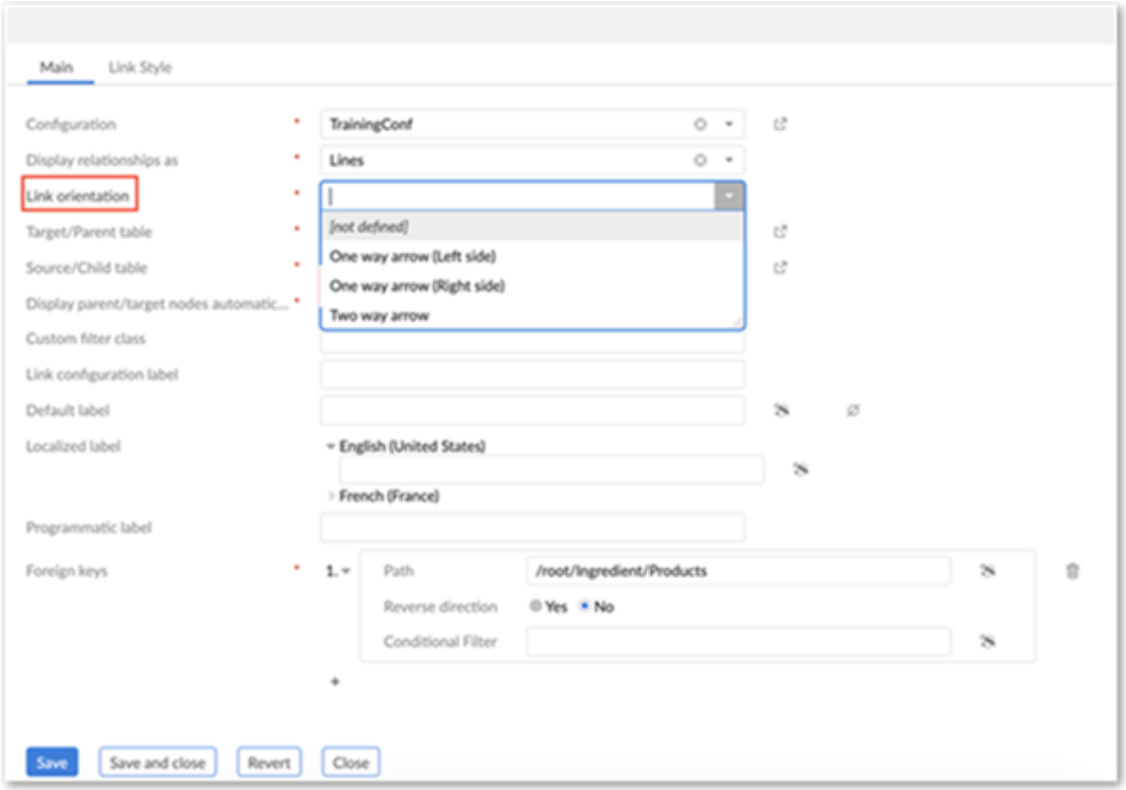
2. Import the configuration archive:
   1. Navigate to the desired add-on.
   2. From the **Actions** menu, select *Archives* > *Export*.
   3. From the **Archive to import** menu, select the name of the archive you exported in step 1.
   4. Select **Import**.

229.3 Data Model and Visualization Add-on additional steps

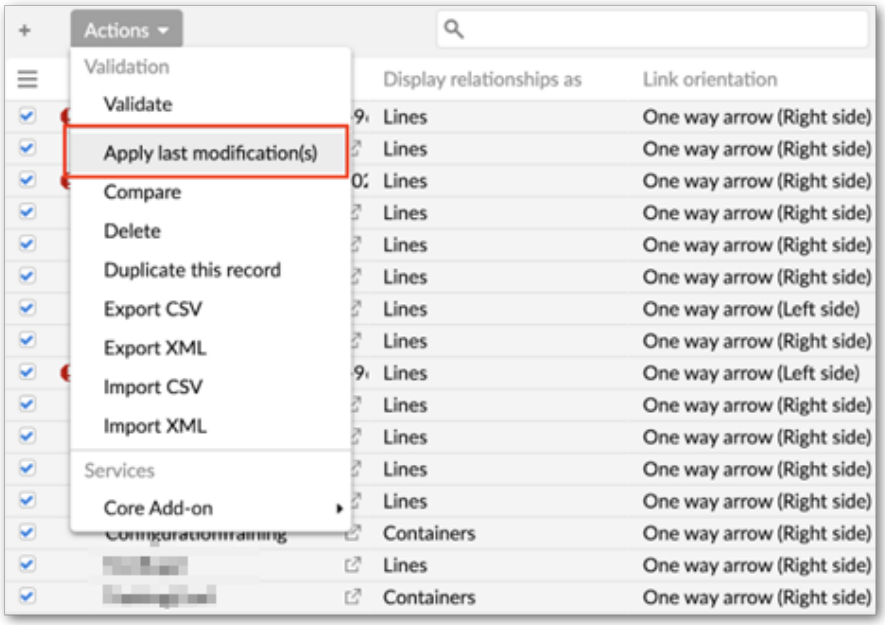
A new feature is included with the add-on that allows users to define link orientation. This is a required field for all links and must be initialized. After migrating your repository and importing the add-on's configuration archive, perform the following steps:

1. Navigate to *Administration* > *User interface* > *TIBCO EBX® Data Model and Data Visualization Add-on* > *Link configuration*.

2. Open a record, set the **Link orientation** property, save and close the record.



3. Optionally apply the same modification to all records in the **Link configuration** table:
   1. Select all records in the table.
   2. Open the **Actions** menu and select **Apply last modification**.



229.4 Digital Asset Manager Add-on additional steps

Use the following steps to migrate the add-on from a version that is compatible with EBX® 5.9.X to a version compatible with 6:

1. In the environment containing the old version of the add-on, navigate to *Administration > User interface > {addon.label.full}*.
2. From the **Actions** menu, select *Archives > Export*.
3. In the environment containing the new version of the add-on, navigate to *Administration > User interface > {addon.label.full}*.
4. From the **Actions** menu, select *Archives > Export*.

   Note that since there is not a direct mapping of all properties from version 5 to 6, some tables will have errors. Complete the following step to resolve these errors.
5. From the **Actions** menu, select **Migrate digital asset manager data**.

The following lists some of the terminology changes take place between versions 5 and 6 (other changes are very minor):

- Drive type --> Connector
- File system --> Default connector
- External management --> External manager
- Display configuration --> Record detail view
- Editor configuration --> Drive view configuration

CHAPTER 230

Step 1: Adapting custom code

This chapter contains the following topics:

1. Overview
2. New architecture impact on custom code performance
3. Adapt code and test migration

230.1 Overview

As a developer, your role to support migration is to ensure that any custom code used in your EBX® implementation compiles against the new EBX® binaries and is capable of successful deployment. This task requires that you understand the differences [p 1600] in the new EBX® 6 architecture and how it impacts performance of existing custom Java code. You should also familiarize yourself with the information regarding backwards compatibility contained in the *TIBCO EBX® Release Notes*.

Begin the process of updating your custom code by reviewing the new architecture's impact on custom code [p 1600]. Then follow the steps in Adapt code and test migration [p 1601]. After successful completion of the test phase, you will have:

- Adjusted and tested your code.
- Made changes in your data models and projects required to successfully test migration in your local environment.
- Generated new binaries capable of deployment and testing in the pre-production environment.

230.2 New architecture impact on custom code performance

The new scalable architecture can impact the performance of custom Java code. Some scenarios that worked well before might require adjustments. In particular, focus on replacing or optimizing the following:

Repeated lookup by primary key	Previously, when the cache in our custom EBX® in-memory indexes could contain an entire table's data, it was cheap to access records using random table lookups. This type of access must now be rewritten to take advantage of the new SQL features. Take for example Java code that performed a nested loop join where a `RequestResult`[API] from table A was iterated on and the `osd:tableRef` was looked up in table B. You should now implement a `Query`[API] that uses an SQL join to handle the lookup.
Repeatedly creating similar requests	EBX® now embeds the Apache Calcite framework to optimize the submitted instances of `Query`[API] and `Request`[API]. Query optimization does have a cost; it is generally low and paid back when compared to executing a non-optimized query. However, it can create significant overhead when a loop generates numerous queries that differ by only a single parameter. In this case, use of parameterized requests (see `Request.setXPathParameter`[API]) or parameterized queries (see `Query.setParameter`[API]) is optimal.
Programmatic access rules	Programmatic access rules implement the Java interface `AccessRule`. On query execution, setting a rule on a table implies execution of the method `getPermission` for each record. This negates the advantages of optimizing against an index. Wherever possible, you should replace these programmatic access rules with the new scripted record permission rules.
Programmatic labels	Programmatic labels implement the Java interfaces `TableRefDisplay`, `UILabelRenderer` and `ConstraintEnumeration`. On query execution, setting a rule on a foreign key field implies execution of the method `displayOccurrence` for each record. This negates the advantages of optimizing against an index. Wherever possible, use a pattern string instead.
Programmatic filters	Programmatic filters implement the Java interface `AdaptationFilter`. On query execution, using this filter implies execution of the method `accept` for each record. This negates the advantages of optimizing against an index.

	Where possible, replace programmatic filters with the new SQL queries.
Constraints with unknown dependencies	Avoid constraints with unknown dependencies on large datasets, as they are checked on each validation request. See Data validation [p 617] for more information.
Unnecessary index refresh	To improve transaction performance (Java implementations of `Procedure` and also of `TableTrigger`), any unnecessary index refresh should be avoided. For more information, see Unnecessary index refresh [p 623].
Inherited fields	As explained in the Search limitations [p 512], inherited fields cannot benefit from the new index optimizations. As a consequence all operations (querying, validation, data comparison, etc.) that will be performed on inherited fields will be slower compared to versions prior to EBX® 6. It is strongly advised to not use inherited fields to avoid performance issues. Wherever possible, depending of the use case, it is recommendable to convert these fields into linked fields [p 933] or to substitute the behavior of inherited fields by simulating inheritance using triggers for instance.
Compression optimization	From release 6.2.3, the compression implementation uses direct/native memory. This makes it more important to explicitly close `RequestResult` and `QueryResult` objects as soon as possible so that memory (and index and database resources) is returned promptly. Previously, `ReverseRelationshipResult.getResult()` returned the same `RequestResult` instance. It now always returns a new `RequestResult`. Update your custom code to ensure that you close the `RequestResult` after each use.

230.3 Adapt code and test migration

It is important that prior to testing migration, you remove all JAR and WAR files related to EBX® 5 from your environment.

1. Repository preparation:

   The following lists high-level tasks for preparing your local repository for your development environment. When preparing a pre-production or production repository for migration, or for more details on repository cleanup tasks mentioned in the following steps, see Step 2: Preparing the repository [p 1603].

   1. Backup your existing repository.
   2. Start EBX® and ensure all data models compile without errors.
   3. Ensure that no data model dependencies on discontinued add-ons exist. See Changes to supported add-ons [p 1593] for more information.

4. Clean dataspaces and workflows. This step is optional. However, note that any closed dataspaces are deleted during migration. Re-open any dataspace you want to keep. This step is more beneficial for larger repositories where large numbers of dataspaces could lead to longer migration times.
5. Ensure the URL policy conforms to 6.x requirements. Navigate to *Administration* > *User interface* > *User interface configuration* > *Advanced perspective* > *Interface configuration* > *URL Policy* and revert any existing settings. To update the URL computing configuration, you can edit the `ebx.properties` file. See URLs computing [p 590] for more information.
6. Shutdown and then backup your clean repository.

2. Code preparation:
   1. Download EBX® 6 binaries from TIBCO eDelivery (an account is mandatory in order to access eDelivery, please contact the support team to request one) and deploy the following JARs in your local environment (located in `<your download directory>/ebx.software/lib`):
      - When migrating to EBX® 6.0.x:
        - For EBX® APIs: `ebx.jar`
        - To support the new compression algorithm: `ebx-lz4.jar`
      - When migrating to EBX® 6.x:
        - Only for versions prior to 6.2.3, to support the new compression algorithm: `ebx-lz4.jar`
        - For EBX® Java APIs: `ebx-api.jar`
        - For third-party Java APIs: `ebx-resources-api.jar`
   2. Review the following to ensure any custom code complies with new recommendations and requirements:
      - New architecture impact on custom code performance [p 1600]
      - *Release Notes* > *Backward compatibility* sections.
      - Java API changes [p 1613]
   3. Recompile any custom libraries or webapps.
3. Initiate auto migration by starting your local server. The process should terminate successfully. Look for the following log entry in the `kernel.log` file: `EBX® Started and Initialized`.

   The following are common issues that can problems:
   - Libraries not compiled using EBX® 6. If this is the case, EBX® throws the following exception: `java.lang.IncompatibleClassChangeError`.
   - Data model errors. If any errors were left unresolved, the migration process is automatically terminated.
   - Compile error in the project if the project isn't built correctly.

Once migration completes successfully in your local environment, you can deploy and test in a pre-production environment with a larger repository. However, when testing in a pre-production environment, it is imperative to follow the process outlined in Step 2: Preparing the repository [p 1603] to prepare the repository prior to starting the application server.

CHAPTER 231

Step 2: Preparing the repository

This chapter contains the following topics:

1. Introduction
2. Task 1: Create a backup
3. Task 2: Prepare the repository
4. Task 3: Back up your clean repository

231.1 Introduction

It is strongly recommended to perform all migration tasks in at least a test and pre-production environment prior to migrating a production environment. Once a server with the EBX® 6 artifacts is started, the repository migration begins automatically; *the process is not reversible*. Migration can only complete successfully if the data models in the repository contain no errors.

> **Note**
>
> You might want to use a copy of, or one that closely resembles, the repository in production. This will allow you to estimate the effort needed to prepare the production repository and allow you to see how long the automatic migration process will take, so you can plan downtime accordingly.

To prepare your repository for migration, complete the steps in the following tasks:

- Task 1: Create a backup [p 1604]
- Task 2: Prepare the repository [p 1604]
- Task 3: Back up your clean repository [p 1609]

231.2 Task 1: Create a backup

The migration process is not reversible. With this in mind it is important to backup your repository in its current state.

> **Attention**
> Whether you are migrating a repository from 5.9, or upgrading from a previous version 6, it is critical that you back up your repository.

To back up your repository:

1. Shut down the application server hosting the EBX® 5 instance.
2. Back up your EBX® 5 repository. Include any relational database objects whose name starts with the current repository prefix. This prefix is specified in the `ebx.properties` file under `ebx.persistene.table.prefix`.
3. Back up and delete any EBX® logs, or move them to another folder.
4. Back up add-on configurations:

   If you have the TIBCO EBX® Data Model and Data Visualization Add-on, or the TIBCO EBX® GO Add-on, you will need to export the configuration archive and import it again after the migration completes. See Manually migrating add-ons [p 1595] for instructions.

231.3 Task 2: Prepare the repository

It is important to clean your EBX® 5.9 environment to accelerate the migration process, especially for repositories with large volumes of data. This process requires administrative access to EBX® as the processes must be performed in the EBX® user interface.

Due to data compression changes beginning with version 6.2.3, it is especially important to delete all unnecessary dataspaces. Migration will create dictionaries, rewrite all blocks to the new format, search for blocks that can share content, recompress the blocks using the dictionary.

You should clean the following repository elements:

- Dataspaces [p 1604]
- Workflows [p 1605]
- Data Models [p 1606]
- URL policy [p 1609]

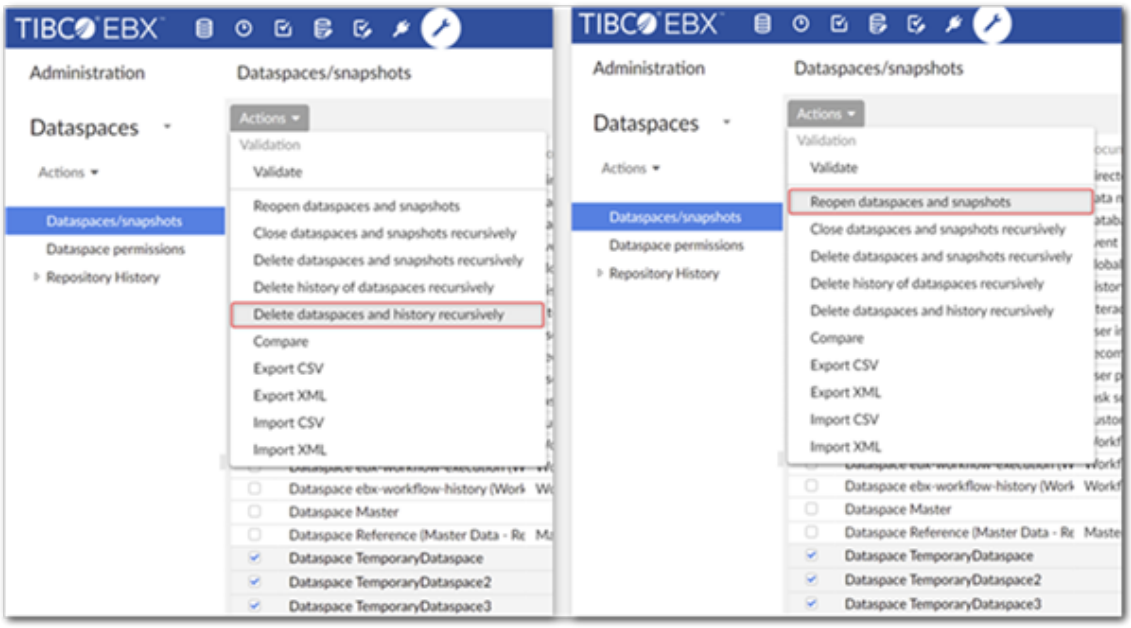
Dataspace cleanup

Closed dataspaces are deleted during migration. You must re-open any dataspace you want to keep prior to migration. We recommend that any closed dataspaces that will not be kept are deleted and purged prior to migration as this can help optimize the migration processing time.

To manage your dataspaces:

1. Navigate to *Administration* > *Dataspaces* > *Dataspaces/snapshots*.
2. Reopen or delete dataspaces:

- To delete: Select the dataspaces to delete and from the table's **Actions** menu select **Delete dataspaces and history recursively**.
- To reopen: Select the dataspaces to delete and from the table's **Actions** menu select **Reopen dataspaces and snapshots**.



3. After purging, check the G_BLK table. If any of the following are true, contact support before migration to get assistance with additional cleanup to reduce migration time:
   - The G_BLK table has more than 10M rows, or
   - A payload exceeding 100GB, or
   - The G_TRV table has more than 10K rows.

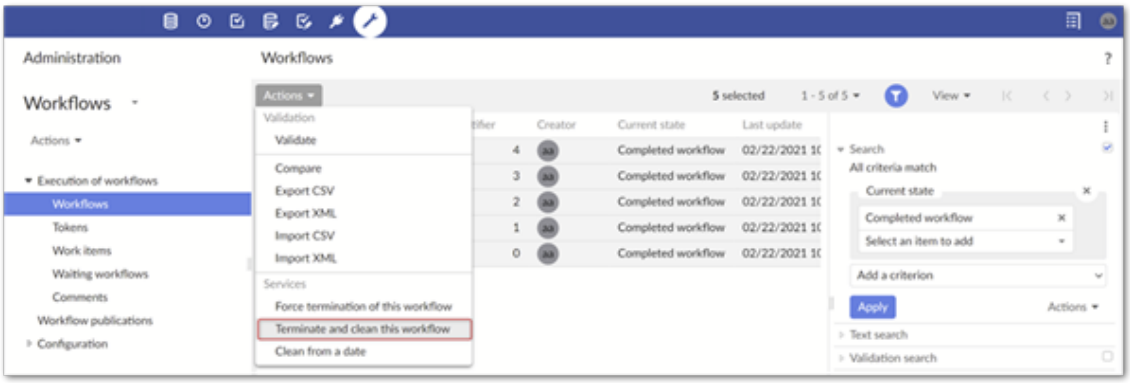
Workflow cleanup

Workflow cleanup is an optional step. However, all workflow tables are automatically migrated and can increase the migration processing time as these tables can be large. If your business use case does not require workflow history, or if it could be externalized in an audit trail "at rest" database, purging the tables can speed up the migration process.

To delete completed workflows:

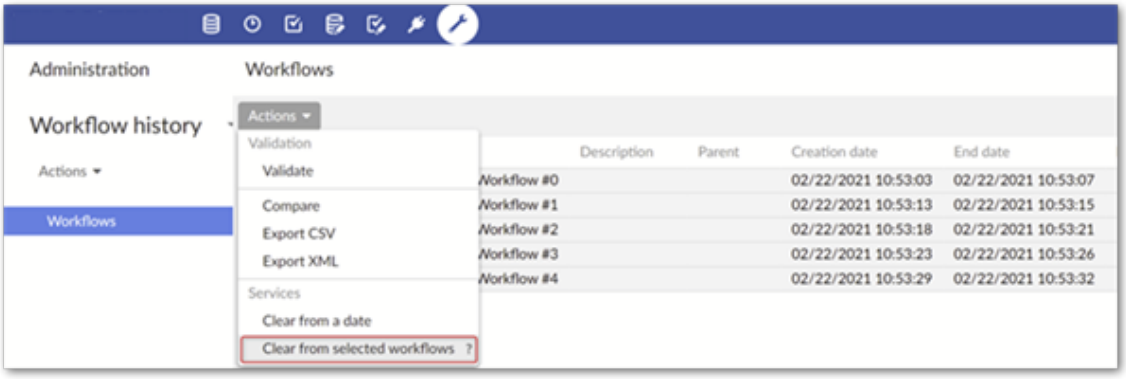
1. Navigate to *Administration* > *Workflow* > *Execution of workflows* > *Workflows*.
2. Create a filter on the table where **Current state** is set to **Completed workflow**.

3. Select all records and from the table's **Actions** menu, select **Terminate and clean this workflow**.



To delete workflow history:

1. Navigate to *Administration* > *Workflow history* > *Workflows*.
2. Select all records to delete and from the table's **Actions** menu select **Clear from selected workflows**.



Data model cleanup

In order for successful completion of the migration process, all data models used by the repository's datasets must compile without errors. This strict requirement is in place to prevent loss of data and to ensure a consistent repository state at the end of the migration process. Additionally, any artifacts from discontinued add-ons should be removed from data models.

To clean your data models, see the topics addressed in the following sections:

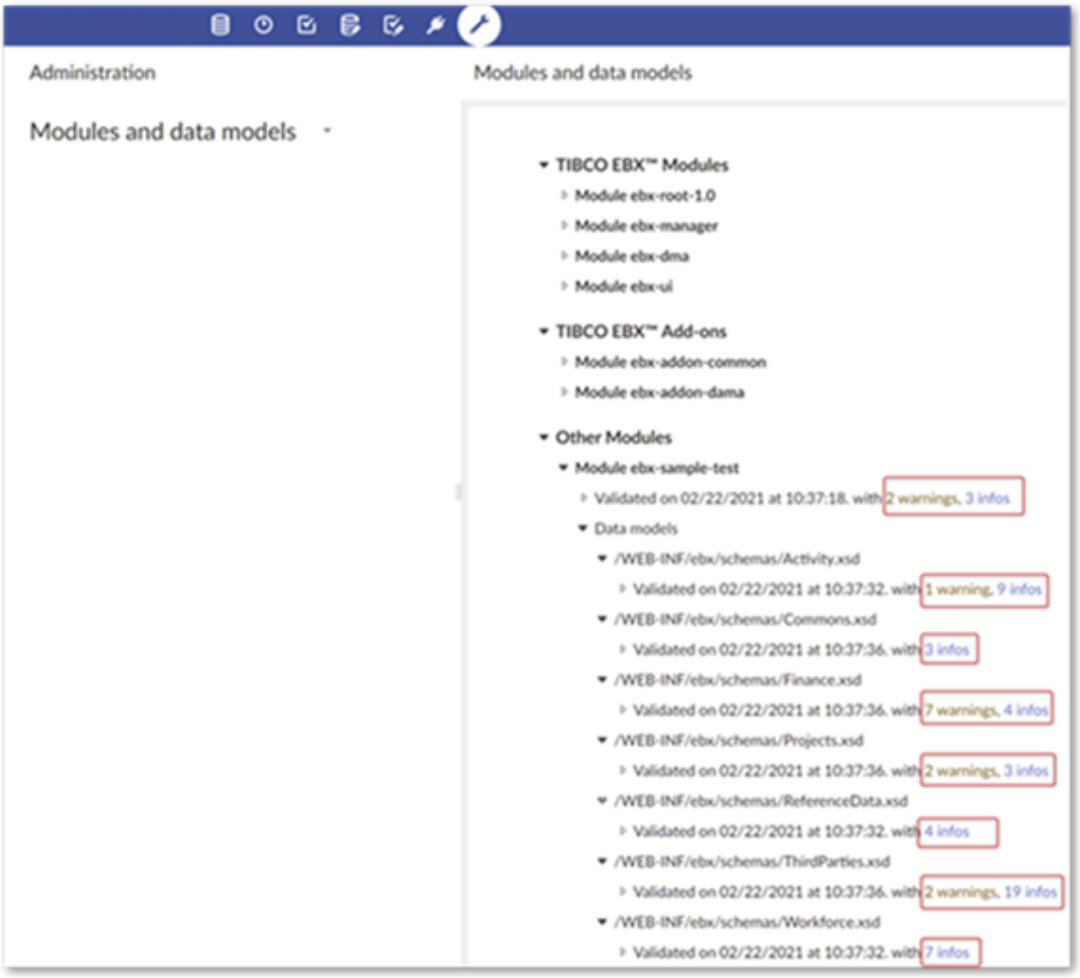
- Data model errors [p 1606]
- Discontinued add-ons [p 1607]
- Java extensions [p 1608]

Resolve data model errors

To resolve data model errors:

1. Locate data model errors by checking the following locations:

- Navigate to *Administration* > *Technical configuration* > *Modules and data models* and look for any data model errors.



- Open the `kernel.log` file and search for `error [-]`.

```
****** Schema[module: ebx-sample-test, path: /WEB-INF/ebx/schemas/ThirdParties.xsd, iid=022]
**         info [-]: Include: Module: ebx-addon-dama, path: /WEB-INF/ebx/schema/media-type.xsd.
**         info [-]: Include: Module: ebx-sample-test, path: /WEB-INF/ebx/schemas/Commons.xsd.
**         error [-]: xs:element name="name" minOccurs="1" maxOccurs="1": element declaration has no type defined.
**         info [-]: Available locales: en_US, fr_FR.
**         info [-]: Import: Model URI: http://schema.orchestranetworks.com/common_1.0.xsd.
**         info [-]: Import: Model URI: http://schema.orchestranetworks.com/session_1.0.xsd.
****** 1 errors, 0 warnings.
```

2. Perform the actions required to fix the data model errors. Alternatively, if the data model is no longer in use, you can delete it. However, this must also be done in any dataspaces that define a dataset based on the data model, including any child dataspaces.
3. Restart the application server and double check the **Modules and data models** table, and the logs to be sure no models are still in a state of error.

Includes for discontinued add-ons

The TIBCO EBX® Match and Merge Add-on (DAQA) might be included in a data model. If this is the case, the include instruction must be removed from the data model as well as any elements based on its data type.

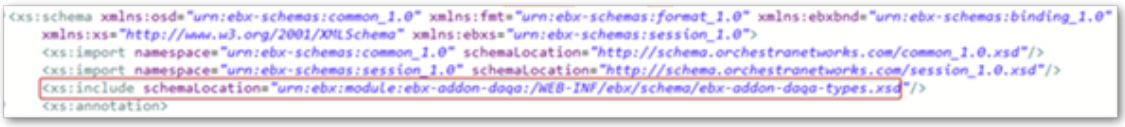
To address discontinued add-ons:

1. Open the `kernel.log` file and search for any data model compilation report that begins with `****** Schema`. Check whether the report contains the following info line: `Include: Module: ebx-addon-daqa`.

   In the following example, the custom module `ebx-sample-test` contained the inclusion in the Persons model:



2. Open your data model's XSD and remove the following: `/WEB_INF/ebx/schema/ebx-addon-daqa-types.xsd`.



3. Search for and remove any elements with a defined type of `DaqaMetaData`.



4. Search for and modify any other elements that were using the elements modified in the step above.



Java extensions for discontinued add-ons

Any Java extension provided by a discontinued add-on must be removed from data models. You must check all XSD files and delete any reference to the non-exhaustive list below:

- For the TIBCO EBX Information Governance Add-on (IGOV), check for use of the following: `com.orchestranetworks.addon.igov.IGovLabelingSchemaDocumentation`

- For the TIBCO EBX Rules Portfolio Add-on (RPFL), check for use of the following data model extension: `com.orchestranetworks.addon.rpfl.DefaultSchemaExtension`
- For the TIBCO EBX Information Search Add-on (TESE), check for use of the following table filter: `com.orchestranetworks.addon.tese.SearchTableFilter`. This add-on is not discontinued, however if you replace it with the reworked version and do not deploy the legacy version, you must remove the trigger.
- For the TIBCO EBX Insight Add-on (DQID), check for use of the following trigger: `com.orchestranetworks.addon.dqid.controller.DQIdTrigger`. This add-on is not discontinued, however if you replace it with the reworked version and do not deploy the legacy version, you must remove the trigger.

In addition to checking for the items listed above, it is a good idea to check toolbars defined by the data model as they can sometimes include user services from the add-ons.

URL policy

The URL policy from 5.9 is deprecated and if you do not update it, issues can occur when attempting to access the UI. To ensure the URL policy conforms to 6.x requirements, navigate to *Administration > User interface > User interface configuration > Advanced perspective > Interface configuration > URL Policy* and revert any existing settings. To update the URL computing configuration, you can edit the `ebx.properties` file. See URLs computing [p 590] for more information.

231.4 Task 3: Back up your clean repository

Back up your current repository:

1. Shut down the application server hosting the EBX® 5.9 instance.
2. Back up your 5.9 repository. Include any relational database objects whose name starts with the current repository prefix. This prefix is specified in the `ebx.properties` file under `ebx.persistene.table.prefix`.
3. Back up and delete any EBX® logs, or move them to another folder.
4. Back up add-on configurations:

   If you have the TIBCO EBX® Data Model and Data Visualization Add-on, or the TIBCO EBX® GO Add-on, you will need to export the configuration archive and import it again after the migration completes. See Manually migrating add-ons [p 1595] for instructions.

CHAPTER 232

Step 3: Deploying

This chapter contains the following topics:

1. Overview

232.1 Overview

What follows are the high-level steps to deploy. If you have questions, refer to the *TIBCO EBX® Installation Guide* as it contains more detailed information and includes platform specific notes.

Attention

If you are deploying add-ons, it is a prerequisite to launch the repackaging `ant` task included in the Add-ons Bundle. Its location is as follows (see the *TIBCO EBX® Add-ons Versioning and Packaging Guide* for additional details): `<Add-ons Bundle location>\script\ebxRepackageWithLGPL.xml`.

To deploy:

1. Stop your application server.
2. Deploy libraries:
    1. For the EBX® core product, the following are located in `<EBX® 6 bundle location>\ebx.software\lib`:
        - When migrating to EBX® 6.0.x:
            - For EBX®: `ebx.jar`
            - To support the new compression algorithm: `ebx-lz4.jar`
            - For email support: `jakarta.mail-2.0.5.jar`
        - When migrating to EBX® 6.x.x:
            - For EBX®: `ebx.jar`
            - Only if migrating to a version prior to 6.2.3, to support the new compression algorithm: `ebx-lz4.jar`
            - For email support: `jakarta.mail-2.0.5.jar`
    2. For EBX® Add-ons: `<EBX® Add-ons Bundle location>\lib\ebx-addons.jar`.
    3. Any custom libraries compiled using EBX® 6.
3. Deploy web applications:

1. For the EBX® core product (located in `<EBX® 6 bundle location>\ebx.software\webapps\wars-packaging`):
   - `ebx-dataservices.war`
   - `ebx-dma.war`
   - `ebx-manager.war`
   - `ebx-root-1.0.war`
   - `ebx-ui.war`
   - `ebx.war`
   - `ebx-authentication.war` (for version 6.1.3 and newer only)
   - `ebx-ide.war` (for version 6.2.0 and newer only)
   - `ebx-hub.war` (for version 6.2.0 and newer only)
   - `ebx-assistant.war` (for version 6.2.2 and newer only)
2. For EBX® Add-ons (located in `<EBX® Add-ons bundle location>\wars`):
   - `ebx-addon-common.war` (mandatory)
   - `ebx-addon-adix.war`
   - `ebx-addon-dama.war`
   - `ebx-addon-dint.war`
   - `ebx-addon-dmdv.war`
   - `ebx-addon-dpra.war`
   - `ebx-addon-dqid.war`
   - `ebx-addon-mame.war`
   - `ebx-addon-moda.war`
   - `ebx-addon-tese.war`
3. All of your project-specific WAR files complied using EBX® 6.

4. Start your application server. The amount of time that the automatic migration process takes depends on the volume of migrated data.

   Attention

   Before beginning the migration, allocate as much memory as possible to Java (Xmx). This can improve migration performance. After migration, reset the allocated memory to 25% - 33% of the available physical memory.

   Successful migration is indicated by the following message in the `kernel.log` file: `******** EBX® started and initialized ********`.

   If a data model is still in error, migration will stop. Use the `kernel.log` file to find the root cause and follow the instructions in Data model cleanup [p 1606].

CHAPTER 233

Java API changes

The following resources list the Java API changes between versions. During the migration process it is important to review these changes and update your custom code accordingly. If migrating from 5.9 to 6.x, you should review the content in both links. Note that the links open the documents directly on TIBCO's documentation site to ensure access to the most up-to-date changes.

- Changes between 6.0 and 6.x
- Changes between 5.9 and 6.0

Notices

CHAPTER 234

Documentation and Support

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

This chapter contains the following topics:

1. How to Access TIBCO Documentation
2. Product-Specific Documentation
3. How to Contact Support for TIBCO Products
4. How to Join TIBCO Community

234.1 How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website website, mainly in HTML and PDF formats.

The Product Documentation website website is updated frequently and is more current than any other documentation included with the product.

234.2 Product-Specific Documentation

The documentation for the TIBCO EBX® is available on the TIBCO EBX® Product Documentation page. This page contains the latest version of each document.

The documentation for the TIBCO EBX® Add-ons is available on the TIBCO EBX® Add-ons Product Documentation page. This page contains the latest version of each document.

To view the documents for Add-on Bundles that are compatible with other versions of TIBCO EBX®, use the **Bundle version** menu to select the desired release.

234.3 How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

- To submit ideas, suggestions, or feedback for TIBCO products, visit our product Support website.

234.4 How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

CHAPTER 235

Legal and Third-Party

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, TIBCO EBX®, TIBCO EBX® Data Exchange Add-on, TIBCO EBX® Add-on's Root Module, TIBCO EBX® Digital Asset Manager Add-on, TIBCO EBX® Match and Merge Add-on, TIBCO EBX® Data Model and Data Visualization Add-on, TIBCO EBX® Insight Add-on, TIBCO EBX® Graph View Add-on, TIBCO EBX® Add-on for Oracle Hyperion EPM, TIBCO EBX® Information Governance Add-on, TIBCO EBX® GO Add-on, TIBCO EBX® Activity Monitoring Add-on, TIBCO EBX® Rule Portfolio Add-on, and TIBCO EBX® Information Search Add-on are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (https://www.cloud.com/legal) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: https://scripts.sil.org/OFL.

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at https://www.cloud.com/legal.



Copyright© 2006-2026. Cloud Software Group, Inc. All Rights Reserved