

ユーザーガイド

- はじめに
 - TIBCO EBX® の仕組み
 - ユーザーインターフェイスの使用
 - 新しいユーザーインターフェイスのプレビュー
 - 用語集
- データモデル
 - データモデルの概要
 - データモデルの実装
 - データモデルの作成
 - データモデルの構成
 - データモデル構造の実装
 - データモデルエレメントのプロパティ
 - エレメントのデータ検証コントロール
 - データモデルの拡張機能
 - 既存のデータモデルの操作
 - データモデルの公開とバージョン管理
 - データモデルの公開
 - 埋め込みデータモデルのバージョン管理
- データスペース
 - データスペースの概要
 - データスペースの作成
 - 既存のデータスペースの操作
 - スナップショット
- データセット

- データセットの概要
- データセットの作成
- データの表示
- データの編集
- ファイルのインポートおよびエクスポートサービス
 - XML のインポートとエクスポート
 - CSV のインポートとエクスポート
- 既存のデータセットの操作
- データセットの継承
- コラボレーション機能 (チームアップ)
 - コラボレーション機能 (チームアップ) の概要
 - コメント
 - 評価
 - タグ
- ワークフローモデル
 - ワークフローモデルの概要
 - ワークフローモデルの作成と実装
 - ワークフローモデルの構成
 - ワークフローモデルの公開
- データワークフロー
 - データワークフローの概要
 - データワークフロー領域のユーザーインターフェイスの使用
 - 作業項目
 - データワークフローの管理
 - データワークフローの起動と監視
 - データワークフローの管理
- データサービス

- データサービスの概要
- データサービス WSDL の生成
- EBX® メタデータ管理
 - メタデータ管理の概要
 - キーワード
 - メタデータパースペクティブへのアクセス
 - ユーザーインターフェイス
 - ユーザーインターフェイスの概要
 - ナビゲーションペイン
 - ナビゲーションペインの概要
 - メタデータカタログ
 - カタログの概要
 - アプリケーションの作成
 - データエレメントの作成
 - プロビジョニング
 - 正規表現
 - ガバナンス
 - ガバナンスの概要
 - ビジネス用語の定義
 - メタデータ
 - メタデータセクションの概要
 - 資産の管理
 - テクノロジー
 - テクノロジーの概要
 - インスタンスの作成
 - システムの作成
 - システムタイプの作成

- インフラストラクチャーの作成
- プロバイダーの作成
- プライバシー
 - プライバシーの概要
 - 処理アクティビティ
 - 同意
 - データ侵害
 - リクエスト
- ワークスペース
- ホームページ
- カスタムビュー
- 高度なデータサービス
 - 高度なデータサービスの概要
 - メタデータの収集
 - 資産のサンプリング
 - 資産分類
 - リネージ
- コラボレーション
 - コラボレーションの概要
 - メタデータコメント
 - メタデータ評価
 - メタデータのタグ付け
 - コラボレーションワークフローによるメタデータの管理

はじめに

- TIBCO EBX® の仕組み
- ユーザーインターフェイスの使用
- 新しいユーザーインターフェイスのプレビュー

- 用語集

TIBCO EBX® の仕組み

- 製品の概要
- EBX® アーキテクチャ

製品の概要

マスターデータ管理 (MDM) は、共有データをモデル化、管理、および最終的に管理する方法です。さまざまな IT システムやさまざまなビジネスチームでデータを共有する必要がある場合は、単一の管理されたバージョンのマスターデータを用意することが重要です。

EBX® を使用すると、ビジネスユーザーと IT ユーザーは、データモデルを設計し、マスターデータコンテンツを管理するために、単一の統合ソリューションで共同作業を行うことができます。

EBX® は、コラボレーションワークフロー、データオーサリング、階層管理、バージョン管理、ロールベースのセキュリティなどの豊富な機能を使用して、あらゆるタイプのマスターデータのモデリングとガバナンスの実装を可能にする MDM ソフトウェアです。

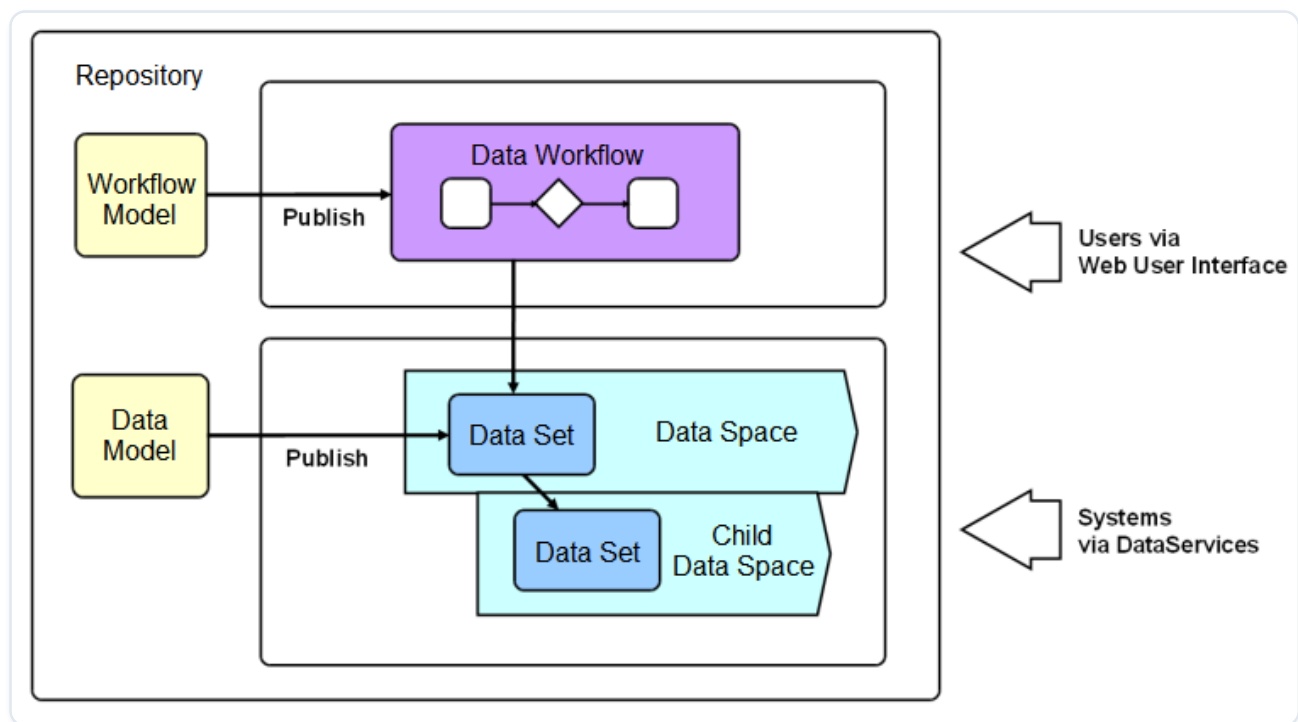
EBX® を使用する MDM プロジェクトは、データモデルの作成から始まります。ここで、マスターデータに関連するテーブル、項目、リンク、およびビジネスルールが定義されます。モデル化されたデータの例には、製品カタログ、財務階層、サプライヤーのリスト、または単純な参照テーブルが含まれます。

次に、データモデルを公開して、データセットで利用できるようにします。データセットは、データモデルで定義された構造に基づいて実際のマスターデータを保存します。データセットは整理され、データスペース内に含まれます。これは、更新を相互に分離するコンテナです。データスペースを使用すると、他のバージョンに影響を与える変更を加えることなく、データの並列バージョンで作業できます。

ワークフローは、管理された変更管理またはデータ承認を実行するための非常に貴重な機能です。これらは、人間と自動の両方の複数のユーザーが関与する段階的なプロセスをモデル化する機能を提供します。

ワークフローモデルは、実行するタスクと、タスクに関連する関係者の詳細を示します。ワークフローモデルが公開されると、データワークフローとして実行できます。データワークフローは、コラボレーションコンテキストに関連するイベントや未解決の作業をユーザーに通知できます。

データサービスは、外部システムがリポジトリ内のデータにアクセスできるようにするか、Web サービスを介してデータスペースとワークフローを管理できるようにすることで、EBX® をサードパーティシステム (ミドルウェア) と統合するのに役立ちます。

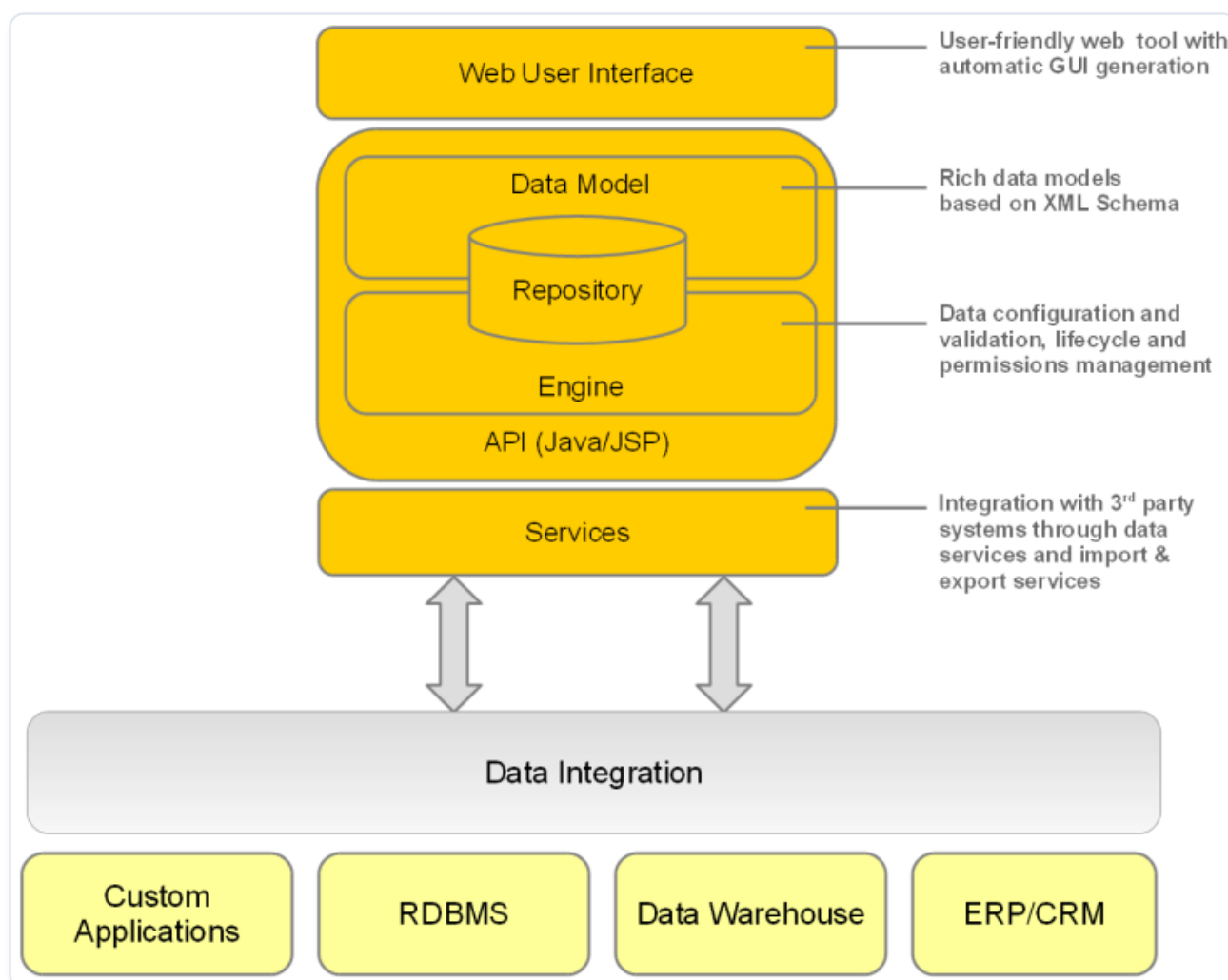


以下も参照してください。:

- データモデリング
- データセット
- データスペース
- ワークフローモデリング
- データワークフロー
- データサービス

EBX® アーキテクチャ

次の図は EBX® アーキテクチャを示しています。



ユーザーガイド 目次

ユーザーインターフェイスの使用

- 概要
- 詳細パースペクティブ
- パースペクティブ
 - パースペクティブモード
- ユーザーペイン
 - アバター
 - 密度
- ユーザーインターフェイス機能
 - ナビゲーションペイン幅のリセット
- EBX® ヘルプの場所

- 状況依存型ヘルプ
- エLEMENTに関する状況依存型ヘルプ

概要

TIBCO EBX® ワークスペースの一般的なレイアウトは、パースペクティブ管理者が完全にカスタマイズできます。

複数のカスタマイズされたパースペクティブが作成されている場合、タイルアイコンのを使用すると、ユーザーは使用可能なパースペクティブを切り替えることができます。

詳細パースペクティブはデフォルトでアクセス可能です。

以下も参照してください。:

- UI 管理

詳細パースペクティブ

デフォルトでは、EBX® の詳細パースペクティブはすべてのユーザーが利用できますが、そのアクセスは選択したプロファイルに制限できます。ビューはいくつかの一般的な領域に分かれており、ドキュメントでは次のように呼ばれています。

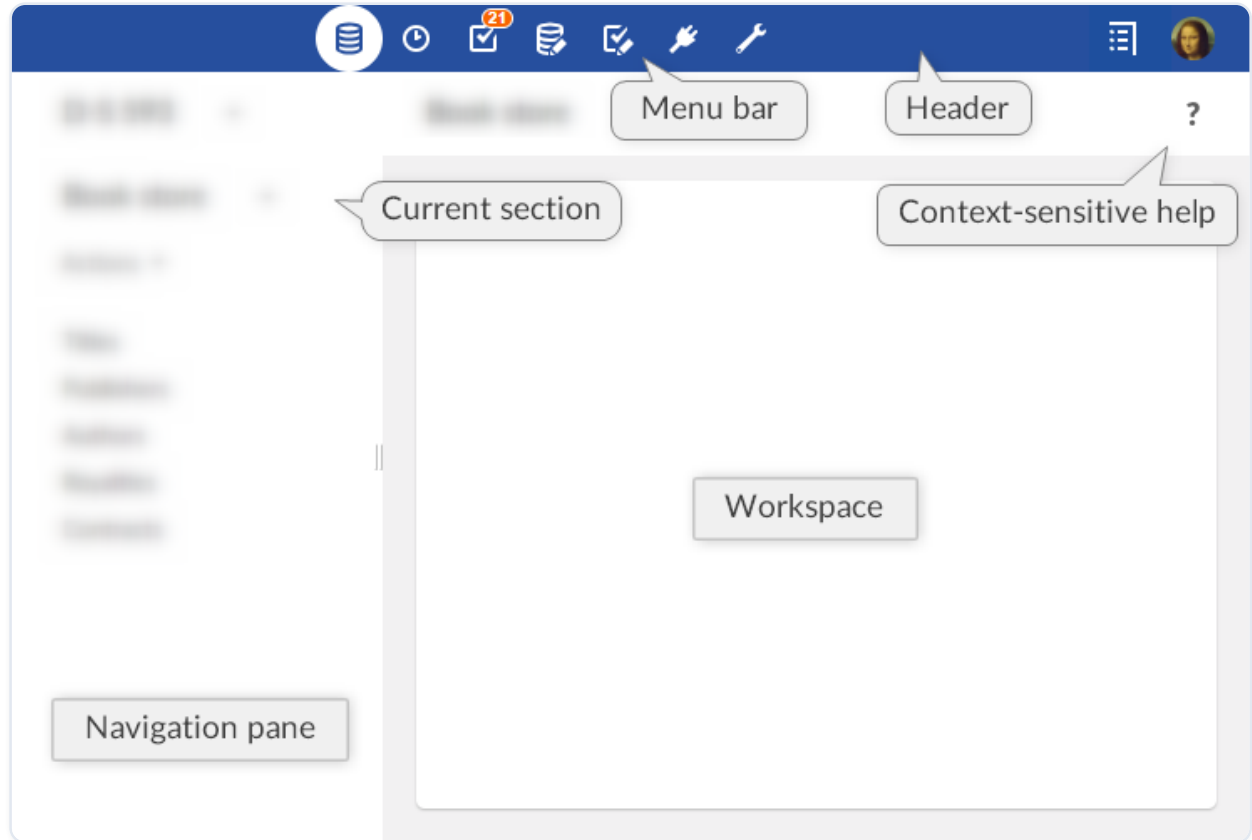
注意:

詳細パースペクティブは、明示的な選択 (Webコンポーネントなど) を通じてユーザーが引き続きアクセスできます。他のパースペクティブとは異なり、ユーザーが自分で適用できないように、ユーザーインターフェイスでのみ「非表示」にすることができます。

- **ヘッダー**: **現在ログインしているユーザーのアバターとパースペクティブセレクトターを表示します。ユーザーのアバターをクリックすると、ユーザーペインにアクセスできます。
- **メニューバー**: **現在のユーザーがアクセスできる機能カテゴリ。
- **ナビゲーションペイン**: **状況に依存するナビゲーションオプションを表示します (例: データセット内のテーブル、またはワークフロー内の作業項目の選択)。

- ****ワークスペース：****インターフェイスで状況依存のメイン作業領域。たとえば、ナビゲーションペインで選択したテーブルがワークスペースに表示されたり、現在の作業項目がワークスペースで実行されたりします。

現在のユーザーの権限に応じて、、、、、 機能領域が表示されます。



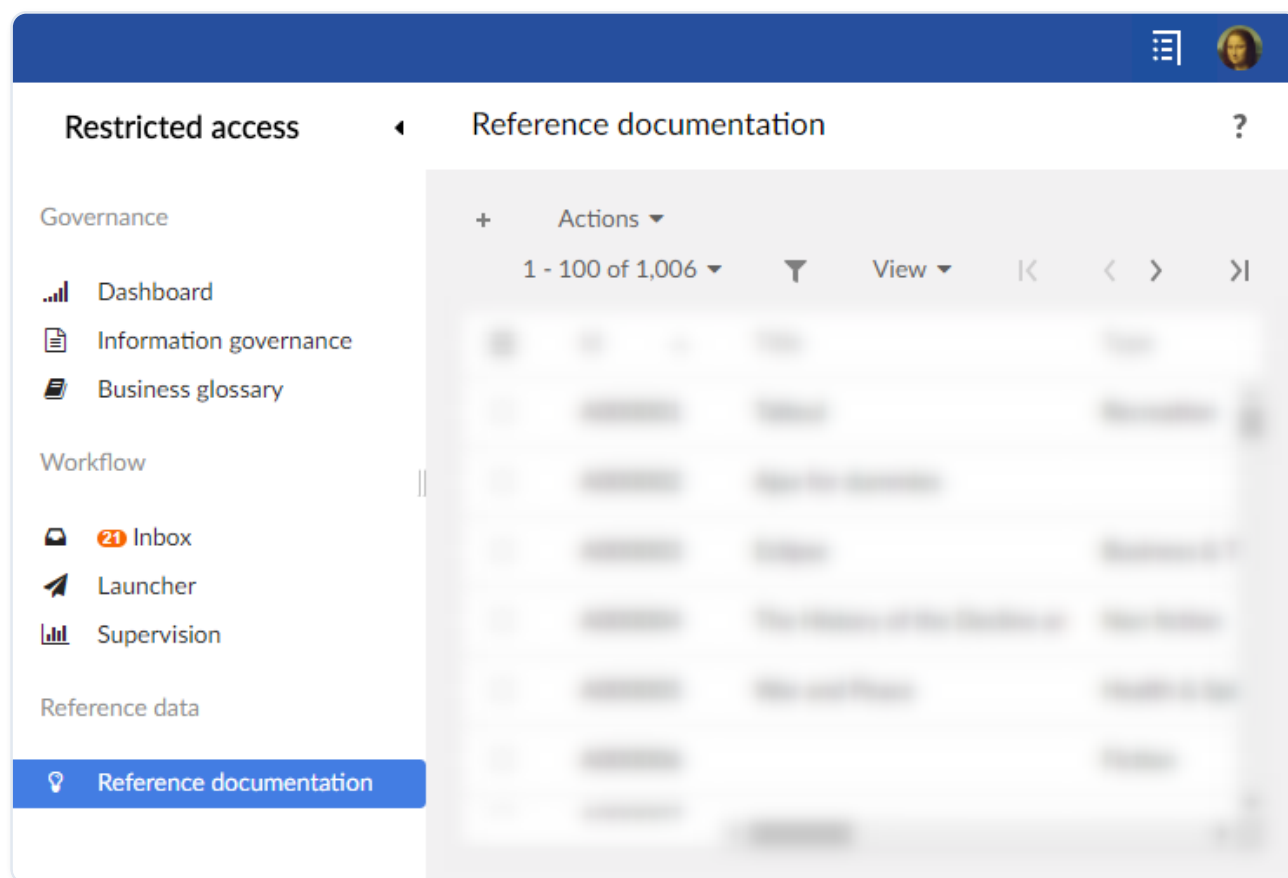
パースペクティブ

EBX® パースペクティブは、ターゲットとなるユーザーに対して高度に構成可能なビューです。パースペクティブは、ビジネスユーザーに簡素化されたユーザーインターフェイスを提供し、1つ以上のプロファイルに割り当てることができます。このビューはいくつかの一般的な領域に分割されており、ドキュメントでは次のように呼ばれています。

- ****ヘッダー：****現在ログインしているユーザーのアバターとパースペクティブセクター (複数のパースペクティブが使用可能な場合) を表示します。ユーザーのアバターをクリックすると、ユーザーペインにアクセスできます。
- ****ナビゲーションペイン：****パースペクティブ管理者によって構成された階層メニューを表示します。ユーザーのアクティビティに関連する関連エンティティやサービスにアクセスするために、展開または折りたたむことができます。
- ****ワークスペース：****インターフェイスで状況依存のメイン作業領域。

パースペクティブは、許可されたユーザーによって構成されます。パースペクティブを構成する方法の詳細については、[パースペクティブ管理](#)を参照してください。

階層メニューの例



パースペクティブモード

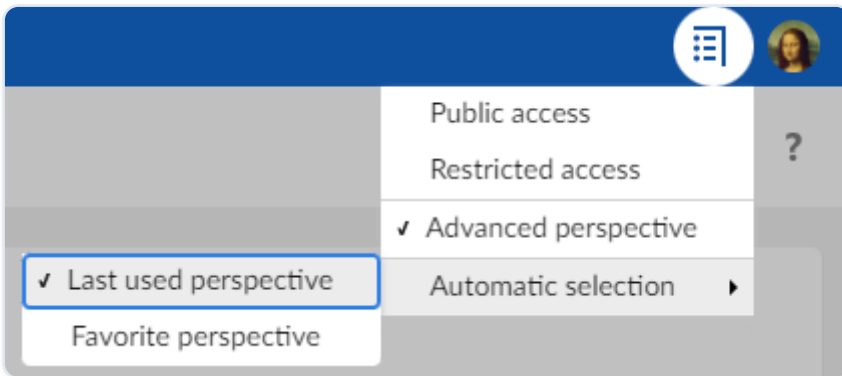
2つのパースペクティブモードから選択できるようになりました。

ユーザーが選択したモードに応じて、最後にアクセスしたパースペクティブまたはお気に入りのパースペクティブが接続時に適用されます。

最後に使用したパースペクティブ

このオプションを使用すると、ユーザーは最後に選択したパースペクティブを取得でき、次回のログイン時に適用されます。

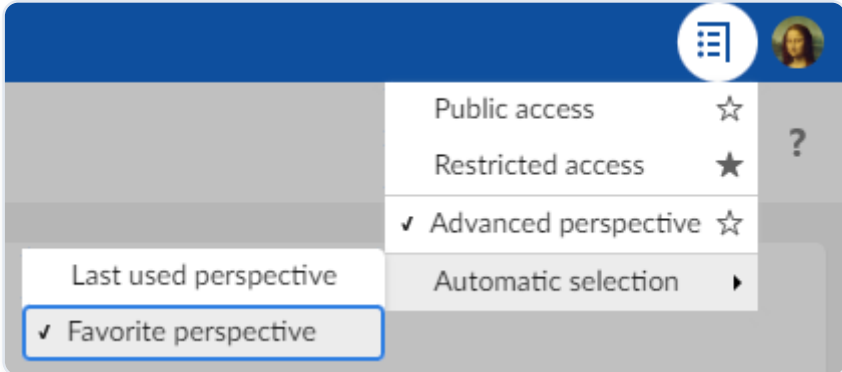
ユーザーがこのモードを選択すると、パースペクティブ名の横にある星が非表示になります。



お気に入りのパースペクティブ

ユーザーが複数のパースペクティブを使用できる場合は、1つをお気に入りのパースペクティブとして定義して、ログイン時にこのパースペクティブがデフォルトで適用されるようにすることができます。これを行うには、各パースペクティブの横にあるパースペクティブセレクターでアイコンを使用できます。

- 塗りつぶしの星は、お気に入りのパースペクティブを示します。これをクリックすると、お気に入りのパースペクティブが削除されます。
- 透明の星は、関連するパースペクティブがお気に入りではないことを示します。これをクリックすると、このパースペクティブがお気に入りのパースペクティブとして設定されます。



以下も参照してください。:

- [推奨されるパースペクティブ](#)

ユーザーペイン

一般的な EBX® 機能は、ユーザーペインにグループ化されています。任意のページの右上隅にあるアバター (またはユーザーのイニシャル) をクリックしてアクセスできます。

次に、ユーザーペインがユーザーアバターとともに表示され、プロフィール構成 (ユーザーの権限に応じて)、言語の選択、密度の選択、およびオンラインドキュメントへのアクセスが提供されます。

注意:

ログアウトボタンはユーザーペインにあります。

アバター

ユーザーごとにアバターを定義できます。アバターは、URL パスを使用して定義された画像または 2 文字 (デフォルトではユーザーのイニシャル) で構成されています。背景色は自動的に設定され、変更することはできません。使用する画像は正方形にする必要がありますが、サイズ制限はありません。

注意:

アバターは、ユーザーペイン、履歴、ワークフローのインターフェイスに表示されます。

この機能は、Java メソッド `UIComponentWriter.addUserAvatar` から利用できます。

アバターのレイアウトは、領域のセクションでカスタマイズできます。アバターのみの表示、ユーザー名のみの表示、または両方の表示を選択できます。

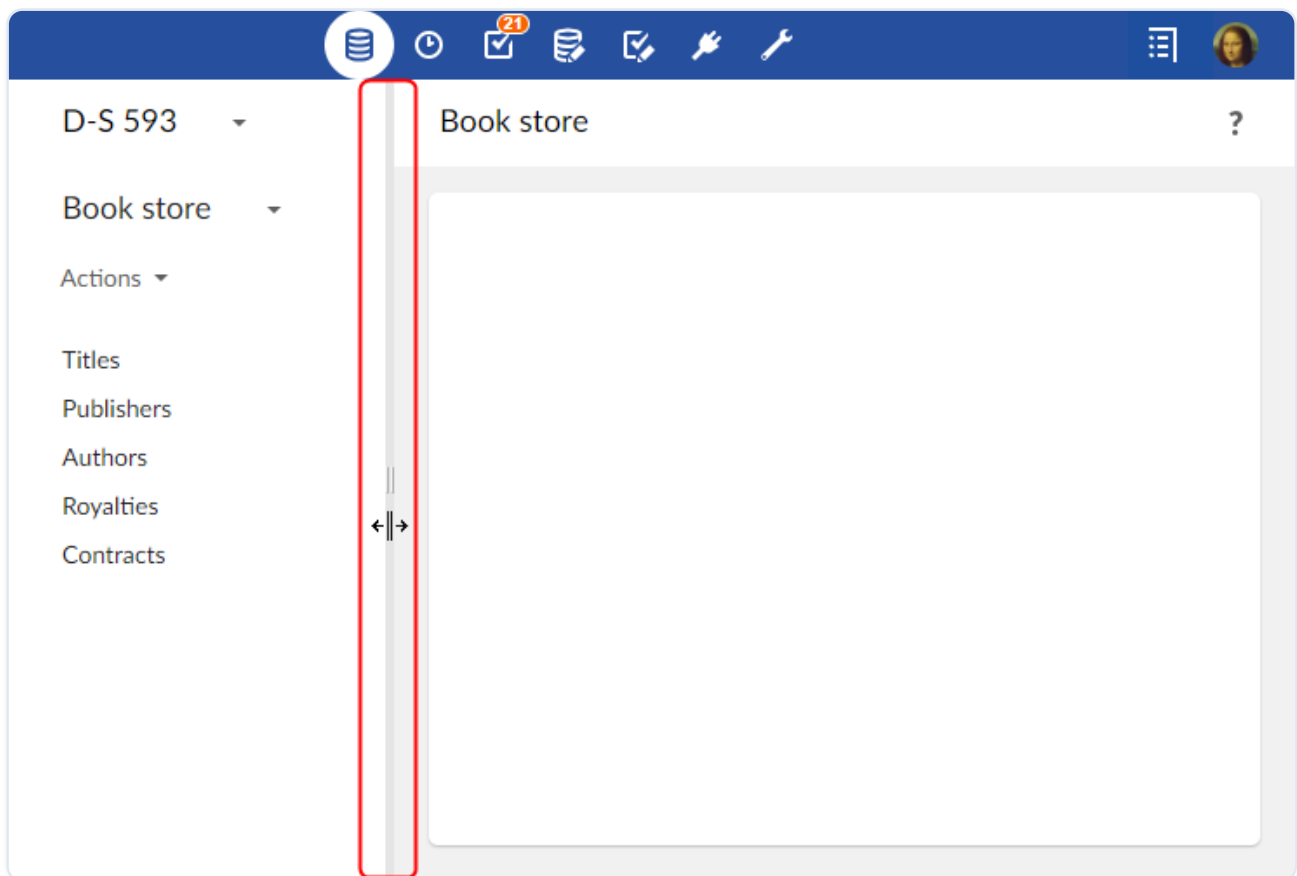
密度

ユーザーは、と から表示密度モードを選択できます。表示モードは、ユーザーペインから変更できます。

ユーザーインターフェイス機能

ナビゲーションペイン幅のリセット

ナビゲーションペインの幅のサイズを変更した後、境界線にカーソルを合わせてダブルクリックすると、デフォルトの幅に戻すことができます。



EBX® ヘルプの場所

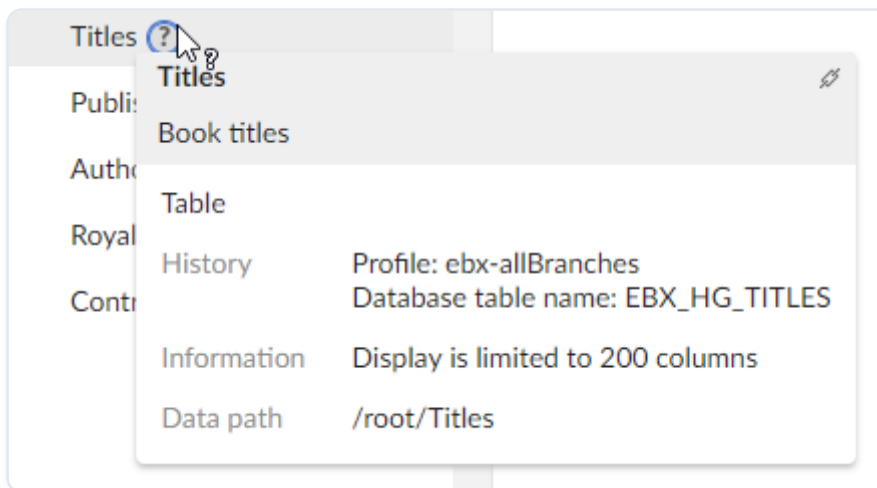
[ユーザーペイン](#)からアクセスできる完全なスタンドアロン製品ドキュメントに加えて、ヘルプにはインターフェイス内のさまざまな形式でアクセスできます。

状況依存型ヘルプ

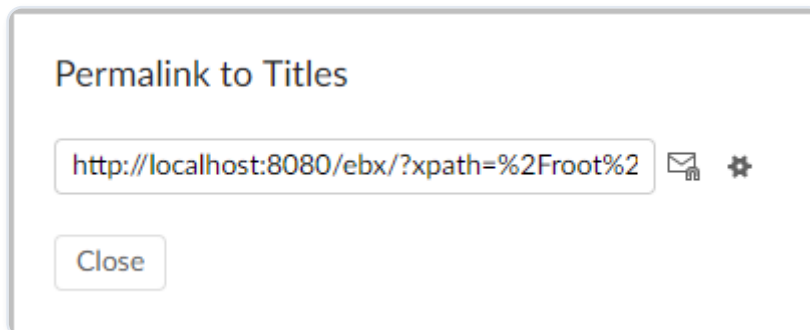
EBX® でワークスペースを参照する場合、2 番目のヘッダーの右側にある疑問符をクリックすると、状況依存のヘルプを利用できます。製品ドキュメントの対応する章が表示されます。

エレメントに関する状況依存型ヘルプ

状況依存のヘルプが定義されているエレメントにカーソルを合わせると、疑問符が表示されます。疑問符をクリックすると、エレメントに関する情報を含むパネルが開きます。



エレメントへのパーマリンクが利用可能になると、パネルの右上隅にリンクボタンが表示されます。



ユーザーガイド [目次](#)

新しいユーザーインターフェースのプレビュー

- [概要](#)
- [プレビューについて](#)
 - [フィードバックの送信](#)
- [新しいUIへのアクセス](#)
- [ナビゲーションの基本](#)
- [新しいUIの使用](#)
 - [ホームの設定](#)
 - [ビューチューニングの理解](#)
 - [ドラフトの理解](#)
- [用語集](#)

概要

この章では、プレビューとして利用可能な新しいユーザーインターフェース (UI) を理解し、使用方法について説明します。このガイドを使用して、UI の操作方法、一般的なタスクの完了方法、および主要な用語を理解してください。このコンテンツは、ビジネスユーザー (データコンシューマー、データスチュワード、コントリビューター) 向けに作成されています。管理設定については、『管理ガイド』>「新しいユーザーインターフェースのプレビュー」に記載されています。

新しい UI は、より高速なナビゲーション、より優れたコンテキストの保持、およびデータ操作のより柔軟な方法をサポートするように設計されており、特にテーブル、レコード、および進行中の作業間を移動する際に役立ちます。

注記:

新しい UI の使用は任意です。既存の UI と並行して評価できますが、既存の UI は本番環境での使用が推奨されるインターフェースです。

プレビューについて

新しい UI は、フィードバックを収集し、早期導入を可能にするためのプレビューとして提供されています。プレビュー機能は、一般提供前に変更される可能性があります。

プレビュー機能はリリース間で変更される可能性があります。現在の UI の一部の機能が欠落しているか、動作が異なる場合があります。パフォーマンスとアクセシビリティはまだ調整中です。

プレビューの使用目的:

- 今後のナビゲーションとインタラクションパターンに慣れるため
- 管理者がアクセスを有効にしている場合、日常業務を実行するため

プレビューを使用しない目的:

- 安定した本番環境対応の動作を必要とする重要な操作のため
- プレビューでまだ利用できない機能に役割が依存している場合

注記:

管理者がプレビュー UI の利用可能性を制御します。

フィードバックの送信

EBX® のあらゆる側面に関するフィードバックは常に歓迎されます。これは特に UI プレビューに当てはまります。フィードバックを送信するには、[EBX® Ideas ポータル](#)の特別なUI > UI プレビューカテゴリを使用してください。

新しいUIへのアクセス

管理者がプレビュー UI の利用可能性を制御します。管理者が UI をデプロイして有効にした後、次の方法でアクセスできます。

- メニューから、オプションを使用します。新しい UI のアーティファクトがデプロイされていない場合、オプションはデフォルトのホームページに移動します
- EBX® URL に `/ebx-hub` を追加します

ナビゲーションの基本

新しい UI は、ドック、ワークスペース、カードの 3 つの主要コンポーネントで構成されています。これらのコンポーネントにより、ナビゲーションとデータ操作にアクセスできます。

次の画像は、新しい UI の簡略化された図であり、前述のコンポーネントと以下のリストにあるそれらの説明を強調することを目的としています。



1) ドック	サイズ変更可能なドックは、画面の左側にある永続的な垂直領域です。その主な目的は、 [ホーム] 、利用可能なパースペクティブ、進行中のデータ操作、および開いているテーブルとレコード間のナビゲーションを可能にすることです。
2) ワークスペース	ワークスペースは、ドックの右側に表示される主要な領域です。新しい UI にアクセスすると、ワークスペースはデフォルトで [ホーム] になります。ドックを使用してナビゲートすると、ワークスペースのフォーカスは選択したビューまたはコンポーネントに自動的に更新されます。
3) カード	カードは、ワークスペースに表示されるコンテンツの個々の単位です。これらは、作業に関連する情報、アクション、またはショートカットを提示し、新しい UI 全体でデータと機能と対話する一貫した方法を提供します。カードを使用して深くナビゲートすると、新しいカードの列が右側に開き、選択に関連するより具体的または詳細なビューが表示されます。これにより、以前のコンテキストを表示したまま、データを段階的に探索できます。 [ホーム]に表示される一部のカードのコンテンツは構成可能です。[ホーム]から移動した後、カードに表示されるコンテンツはパースペクティブ構成によって決定されます。

新しいUIの使用

このセクションの次のトピックでは、新しい UI と対話する方法の一部を紹介します。

- **ホームの設定:** ****をパーソナライズする方法について説明します

- **ビューチューニングの理解:** データビューアクセス、フィルタリング、および検索機能の統合を強調します
- **ドラフトの理解:** データの作成または編集時のドラフトの新しい概念を概説します

ホームの設定

****は、新しい UI のパーソナライズされたランディングエリアです。最も重要なデータ、最近のアイテム、およびコンポーネントを表示するように構成できます。

****を編集すると、次のことができます。

- 次の機能を提供するカードを追加または削除
 - 頻繁にアクセスするテーブルへのショートカット
 - 開いているドラフトの表示
 - レコード評価の表示
 - ワークフロー受信トレイの表示
 - 最近作成および変更されたレコードの表示
 - 指定されたテーブル内のレコード数のカウント
- カードの再配置、サイズ変更、および構成
- ドックのレイアウトの変更。このレイアウトは、****から移動しても保持されます

****を構成するには:

1. 新しい UI に初めてアクセスする場合は、**をクリックします。または、画面の右上にある**アイコンをクリックします
2. 編集モードになったら、変更したい内容に応じて、**または**のいずれかを展開します
3. 表示コンポーネントを並べ替えてドックを編集します
4. 次の方法で****表示を編集します
 - 目的のカードをグリッドにドラッグします
 - カードがグリッド上にある場合、サイズ変更や再配置が可能です。一部のカードに表示される情報はデフォルトで設定されており、その他は構成が必要です。構成が必要なカードについては、次のステップを参照してください
5. カードの****アイコンをクリックして、カードに必要な情報を提供します
6. 変更後、保存して閉じます

ビューチューニングの理解

新しい UI では、カードの上部にある歯車アイコンをクリックして、テーブルのビューをチューニングできます。ビューチューニングは、次の機能を提供することでデータの検索を支援することを目的としています。

- 階層ビューを含む、構成済みのテーブルビューの選択と表示
- テーブルデータに対する検索の実行
- 1つ以上のフィールドを使用したレコードの並べ替え

ドラフトの理解

新しい UI では、データベースにコミットする前にデータを作成および編集するためのドラフトの概念が導入されています。

ドラフトの仕組み:

- 新しいレコードを追加するか、既存のレコードを編集すると、新しいドラフトが作成されます
- 変更は、作成時にドラフトに保存されます
- ドラフトを使用すると、部分的な変更をコミットせずに、作業を一時停止したり、別の場所に移動したり、後で戻ったりすることができます

ドラフトレコードは、変更がまだデータベースに伝播されていないため、テーブルのカードには表示されないことに注意してください。進行中のドラフトにアクセスするには、次のことができます。

- ****に移動し、**カードを使用します**
- レコードの作成または編集時に、開いているドラフトを選択します

用語集

このセクションでは、このドキュメントで使用されている用語の定義を提供します。

ドック	画面の左側にある永続的な垂直バーで、 [ホーム] 、構成済みのパースペクティブ、現在のデータ操作、開いているテーブルとレコードへのナビゲーションを提供します。
ワークスペース	ドックの右側に表示される主要な領域です。ここでは、ユーザーがタスクを完了するた

	めにデータを表示、操作、および管理します。
カード	カードは、ワークスペース内の構造化されたコンテナであり、情報、アクション、またはツールを提示します。カードは、製品全体でデータを表示および操作する一貫した方法を提供します。権限に応じて、カードは編集可能、構成可能、または読み取り専用の場合があります。
ドラフト	ドラフトは、データベースに保存する前に段階的に作業できる、コミットされていない変更のセットを表します。ドラフトは、データを追加または編集するときに自動的に作成されます。
ホーム	新しい UI のパーソナライズされたランディングエリアです。ホームには、ショートカット、ドラフト、最近のアクティビティなどのカードが表示され、作業に最も関連性の高い情報を強調するようにカスタマイズできます。
ドラフトカード	進行中のドラフトを表示し、未完了のデータ編集に戻ることができる[ホーム]カードです。
ビューチューニング	ビューの選択、検索、レコードの並べ替えなど、テーブルでのデータの表示方法を調整できるツールです。

用語集

- ガバナンス
 - repository
 - プロファイル
 - ユーザー

- ロール
- 管理者
- ユーザーとロールのディレクトリ
- ユーザーセッション
- データモデリング
 - データモデル
 - フィールド
 - 主キー
 - 外部キー
 - テーブル (データモデル内)
 - group
 - 再利用可能な型
 - 検証ルール
 - データモデルアシスタント (DMA)
- データセット
 - レコード
 - テーブル (データセット内)
 - dataset
 - inheritance
 - ビュー
 - 推奨ビュー
 - お気に入りのビュー
- データ管理のライフサイクル
 - dataspace
 - 参照データスペース
 - データスペースのマージ
 - スナップショット
- 履歴

- 履歴化
- 履歴プロフィール
- テーブル履歴ビュー
- トランザクション履歴ビュー
- ワークフローモデリング
 - ワークフローモデル
 - スクリプトタスク
 - ユーザータスク
 - ワークフローの状態
 - サブワークフローの呼び出し
 - 待機タスク
 - データコンテキスト
- データワークフロー
 - ワークフローパブリケーション
 - データワークフロー
 - 作業リスト
 - 作業項目
 - トークン
- データサービス
 - データサービス
 - リネージ
- クロスドメイン
 - node

ガバナンス

repository

TIBCO EBX® によって管理されるすべてのデータを含むバックエンドストレージエンティティ。リポジトリはデータスペースに編成されています。

[データスペース](#)も参照してください。

プロフィール

ユーザーまたはロールの総称。プロフィールは、データワークフローで、およびアクセス許可ルールを定義するために使用されます。

[ユーザー](#)、[ロール](#)も参照してください。

関連する Java APIは、`Profile` です。

ユーザー

物理ユーザーまたは外部システムが EBX® を認証してアクセスするためにリポジトリに作成されたエンティティ。ユーザーにはロールが割り当てられ、他のアカウント情報が関連付けられている場合があります。

[ユーザーとロールのディレクトリ](#)、[プロフィール](#)も参照してください。

関連する概念は、[ユーザーとロールのディレクトリ](#) です。

関連する Java API は、`UserReference` です。

ロール

ユーザーに割り当てることができる、アクセス許可ルールとデータワークフローに使用されるユーザー分類。各ユーザーは複数のロールに属している場合があります。

EBX® でロールプロフィールが指定されている場合は常に、その指定の結果として生じる動作が、そのロールのメンバーであるすべてのユーザーに適用されます。たとえば、ワークフローモデルでは、誰に作業項目を提供するかを定義するときにロールを指定できます。その結果、そのロールに属するすべてのユーザーが同じ作業項目オファーを受け取ることができます。

[ユーザーとロールのディレクトリ](#)、[プロフィール](#)も参照してください。

関連する概念は、[ユーザーとロールのディレクトリ](#) です。

関連する Java APIは、`Role` です。

管理者

EBX® の技術的な管理と構成にアクセスできる事前定義されたロール。

ユーザーとロールのディレクトリ

リポジトリにアクセスするときに認証に使用できるメソッド、使用可能なすべてのロール、およびロール割り当てを使用してリポジトリへのアクセスを許可されたユーザーを定義するディレクトリ。

[ユーザー](#)、[ロール](#)も参照してください。

関連する概念は、[ユーザーとロールのディレクトリ](#) です。

関連する Java API は、`Directory`、`DirectoryHandler` です。

ユーザーセッション

ユーザーおよびロールディレクトリに対して認証された後、ユーザーに関連付けられたリポジトリアクセスコンテキスト。

関連する概念は、[ユーザーとロールのディレクトリ](#) です。

関連する Java API は、`Session` です。

データモデリング

[メインドキュメントセクション](#) [データモデル](#)

データモデル

EBX® リポジトリで管理されるデータの構造的定義。データモデルには、編成、データ型、およびセマンティック関係の観点から、含まれているすべてのデータの詳細な説明が含まれています。データモデルの目的は、リポジトリによって管理されているデータを含むデータモデルのインスタンスであるデータセットの構造と特性を定義することです。

[データセット](#)も参照してください。

関連する概念は、[データモデル](#) です。

フィールド

名前と単純なデータ型で定義されたデータモデルエレメント。フィールドは、データモデルに直接含めることも、テーブルの列として含めることもできます。EBX® では、フィールドに長さやサイズなどの基本的な制約に加えて、計算を含むより複雑な検証ルールを割り当てることができます。フィールドの継承または他のデータに基づく計算を使用した自動値割り当ても、フィールドに対して定義できます。集約リストは、フィールドのカーディナリティを設定して、同じレコードに

複数の値を含めることで作成できます。フィールドをグループに配置して、データモデルの構造的な編成を容易にすることができます。

デフォルトでは、フィールドはアイコン  で表されます。

[レコード](#)、[グループ](#)、[テーブル \(データモデル内\)](#)、[検証ルール](#)、[継承](#)も参照ください。

関連する概念は、[構造エレメントのプロパティ](#)、[データフィールドの制御](#) です。

関連する Java API は、`SchemaNode` です。

「フィールド」の以前の名前 (バージョン 5 より前) は「属性」でした。

主キー

テーブル内のレコードを一意に識別するために使用されるフィールドまたは複数のフィールドの組み合わせ。

主キーはアイコン  で表されます。

関連する概念は、[テーブル定義](#) です。

外部キー

フィールド値が別のテーブルの主キーに対応する、1 つのテーブル内のフィールドまたは複数のフィールドの構成。外部キーは、あるテーブルのレコードを別のテーブルから参照するために使用されます。

外部キーはアイコン  で表されます。

[主キー](#)も参照してください。

関連する概念は、[外部キー](#) です。

テーブル (データモデル内)

フィールドおよび/またはフィールドのグループで構成されるデータモデルエレメント。すべてのテーブルは、レコードの一意的識別子または主キーとして機能するために、少なくとも 1 つのフィールドを定義する必要があります。データモデル内のテーブルを使用して、テーブルの構造に基づいて再利用可能な型を作成できます。次に、データモデル内の同じ構造の他のエレメントを作成するために使用できます。

テーブルはアイコン  で表されます。

[レコード](#)、[主キー](#)、[再利用可能な型](#)も参照してください。

group

データモデルの編成を容易にするために使用される分類エンティティ。グループは、フィールド、他のグループ、およびテーブルを収集するために使用できます。グループにテーブルが含まれている場合、テーブルをネストできないという制約に従う必要があるため、グループを別のテーブルに含めることはできません。グループを使用して、グループの構造に基づいて再利用可能な型を作成できます。次に、グループを使用して、データモデル内に同じ構造の他のエレメントを作成できます。

グループはアイコン  で表されます。

[再利用可能な型](#)も参照してください。

関連する Java API は、`SchemaNode` です。

再利用可能な型

データモデル内の他のエレメントを定義するために使用できる、共有された単純または複合型の定義。

検証ルール

フィールドまたはテーブルで定義された受け入れ基準。課せられたすべての検証ルールに準拠していない場合、データは無効と見なされます。

「検証ルール」の以前の名前 (バージョン 5 より前) は「制約」でした。

データモデルアシスタント (DMA)

EBX® ユーザーインターフェイスには、データモデルの実装を支援するツールが含まれています。これにより、データモデルの構造の定義、エレメントの作成と編集、およびデータモデルの構成と公開が可能になります。

[データモデル](#)も参照してください。

データセット

[メインドキュメントセクション データセット](#)

レコード

主キーによって一意に識別される、テーブル内のフィールド値のセット。レコードはテーブルの行です。各レコードは、データモデルで定義されたデータ構造に従います。データモデルは、レコードで見つかったフィールドのデータ型とカーディナリティを駆動します。

[テーブル \(データセット内\)](#)、[主キー](#)も参照してください。

「レコード」の以前の名前 (バージョン 5 より前) は「オカレンス (occurrence)」でした。

テーブル (データセット内)

データを含む同じ構造のレコード (行) のセット。各レコードは、主キーによって一意に識別されます。

テーブルはアイコン  で表されます。

[レコード](#)、[主キー](#)も参照してください。

dataset

データモデルのデータを含むインスタンス。データセットの構造と動作は、実装しているデータモデルによって提供される定義に基づいています。データモデルに応じて、データセットにはテーブル、グループ、およびフィールドの形式でデータが含まれます。

データセットはアイコン  で表されます。

[テーブル \(データセット内\)](#)、[フィールド](#)、[グループ](#)、[ビュー](#)も参照してください。

関連する概念は、[データセット](#) です。

「データセット」の以前の名前 (バージョン 5 より前) は「適応インスタンス」でした。

inheritance

あるエンティティが別のエンティティからデフォルトでデータを取得できるメカニズム。EBX® には、データセット継承とフィールド継承の 2 種類の継承があります。

有効にすると、データセットの継承により、子データセットが親データセットからデフォルトのデータ値を取得できるようになります。この機能は、親スコープで宣言されたデータが、ネストされた子スコープでデフォルトで同じ値で使われるデータモデルを設計するときに役立ちます。親から継承された値は、子によってオーバーライドできます。デフォルトでは、データセットの継承は無効になっています。データモデルの定義中に有効にすることができます。

親データセットからの継承は、アイコン  で表されます。

フィールド継承はデータモデルで定義され、別のテーブルのレコードからフィールド値を自動的にフェッチします。

継承されたフィールドは、アイコン  で表されます。

関連する概念は、[継承と値の解決](#) です。

ビュー

表示テーブルに適用できるカスタマイズ可能な表示構成。レコードを表形式、階層形式、またはタイル形式で表示するかどうかを指定したり、レコードのフィルタリング基準を設定したりするために、特定のユーザーまたはロールに対してビューを定義できます。

階層ビュー型は、テーブル内のデータのツリーベースの表現を提供します。ツリー内のノードは、フィールド値またはレコードのいずれかを表すことができます。階層ビューは、モデルデータ間の関係を示すのに役立ちます。階層形式を使用するビューを作成する場合、データの構造表現を決定するためにディメンションを選択できます。階層ビューでは、再帰的な関係をナビゲートしたり、外部キーの関係を使用して複数のテーブル間を移動したりすることができます。

以下も参照してください。:

- [ビュー](#)
- [階層](#)

推奨ビュー

推奨されるビューは、各ターゲットプロファイルのデータセット所有者が定義できます。ユーザーがビューを指定せずにログインすると、推奨されるビュー (存在する場合) が適用されます。それ以外の場合は、デフォルトのビューが適用されます。

「推奨ビューの管理」アクションを使用すると、ユーザーとロールに応じて推奨ビューの割り当てルールを定義できます。

関連する概念は、[推奨ビュー](#) です。

お気に入りのビュー

テーブルを表示する場合、ユーザーはメニューツールバーから現在のビューをお気に入りのビューとして定義することを選択できます。

お気に入りとして設定すると、このユーザーがテーブルにアクセスするたびにビューが自動的に適用されます。

関連する概念は、[\[ビュー\] メニューツールバー](#)です。

データ管理のライフサイクル

メインドキュメントセクション [データスペース](#)

dataspace

データセットで構成されるコンテナエンティティ。これは、データセットのさまざまなバージョンを分離したり、それらを整理したりするために使用されます。

子データスペースは、特定の親データスペースに基づいて作成され、親の状態で初期化されます。次に、データセットは、親データスペースから分離して、また相互に、子データスペースで変更できます。子データスペースは、後で親データスペースにマージして戻すか、他のデータスペースと比較することができます。

[継承](#)、[リポジトリ](#)、[データスペースのマージ](#)も参照してください。

関連する概念は、[データスペース](#)です。

「データスペース」の以前の名前 (バージョン 5 より前) は「ブランチ」または「スナップショット」でした。

参照データスペース

EBX® リポジトリ内のすべてのデータスペースのルート祖先データスペース。すべてのデータスペースのマージは、その親にマージする子で構成されている必要があるため、参照データスペースが別のデータスペースにマージされる資格はありません。

[データスペース](#)、[データスペースのマージ](#)、[リポジトリ](#)も参照してください。

データスペースのマージ

親データスペースへの作成以降に子データスペースで行われた変更の統合。マージが正常に完了した後、子データスペースは閉じられます。マージを実行するには、ソースデータスペースとターゲットデータスペースの間で特定されたすべての違いを確認し、競合を解決する必要があります。

たとえば、子データスペースの作成以降に親データスペースと子データスペースの両方でエレメントが変更された場合、マージの結果として保持するエレメントのバージョンを決定することにより、競合を手動で解決する必要があります。

関連する概念は、[マージ](#)です。

スナップショット

参照のために、特定の時点での状態とすべてのコンテンツをキャプチャするデータスペースの静的コピー。スナップショットは、表示、エクスポート、および他のデータスペースと比較できますが、直接変更することはできません。

スナップショットはアイコン  で表されます。

関連する概念は、[スナップショット](#)です。

「スナップショット」の以前の名前 (バージョン 5 より前) は「バージョン」または「ホーム」でした。

履歴

メインドキュメントセクション [履歴](#)

履歴化

リポジトリ内の変更を追跡するためにテーブルレベルで有効にできるメカニズム。履歴化がアクティブ化されると、テーブル履歴ビューとトランザクション履歴ビューの2つの履歴ビューを使用できます。すべての履歴ビューで、エクスポート、比較、フィルタリングなど、テーブルのほとんどの標準機能を利用できます。

履歴をアクティブ化するには、履歴プロファイルを構成する必要があります。テーブルの履歴化はデフォルトでは有効になっていません。

[テーブル履歴ビュー](#)、[トランザクション履歴ビュー](#)、[履歴プロファイル](#)も参照してください。

履歴プロファイル

どのデータスペースに変更をテーブル履歴に記録するか、および履歴化が利用できない場合にトランザクションを失敗させるかどうかを指定する一連の設定。

[履歴プロファイル](#)も参照してください。

テーブル履歴ビュー

レコードの作成、更新、削除など、特定のテーブルで行われたすべての変更のトレースを含むビュー。各エントリには、タイムスタンプやアクションを実行するユーザーなどのトランザクション情報、およびトランザクションの終了時のデータが含まれます。この情報は、レコードまたはデータセットレベルで参照することもできます。

関連するテクニカルリファレンスは、[履歴](#)です。

トランザクション履歴ビュー

トランザクションの技術データと認証データを、リポジトリレベルまたはデータスペースレベルでグローバルに表示するビュー。単一のトランザクションが複数のアクションを実行し、1つ以上のデータセット内の複数のテーブルに影響を与える可能性があるため、このビューには、各トランザクションの特定のスコープ全体で発生したすべての変更が表示されます。

関連するテクニカルリファレンスは、[履歴](#)です。

ワークフローモデリング

メインドキュメントセクション [ワークフローモデル](#)

ワークフローモデル

データに対して実行される操作の手続き上の定義。ワークフローモデルは、データが処理されるためにたどる必要のある完全なパスを記述します。これには、データの状態と、人間のユーザーおよび自動化されたスクリプトによって実行される関連アクションが含まれます。

関連する概念は、[ワークフローモデル](#)です。

「ワークフローモデル」の以前の名前 (バージョン 5 より前) は「ワークフロー定義」でした。

ワークフローモデルは、アイコン  で表されます。

スクリプトタスク

自動化されたプロセスによって実行されるデータワークフロータスク。人間の介入は必要ありません。一般的なスクリプトタスクには、データスペースの作成、データスペースのマージ、スナップショットの作成が含まれます。

スクリプトタスクは、アイコン  で表されます。

[ワークフローモデル](#)も参照してください。

ユーザータスク

人間のユーザーによって同時に実行される 1 つ以上の作業項目で構成されるデータワークフロータスク。ユーザータスクの作業項目は、ワークフローモデルに応じて、ユーザーに提供または割り当てられます。ユーザータスクを超えたデータワークフローの進行は、ワークフローモデルで定義されたタスク終了基準の充足に依存します。

ユーザータスクは、アイコン  で表されます。

[ワークフローモデル](#)も参照してください。

ワークフローの状態

データワークフローの決定ステップ。データワークフロー条件は、次に実行されるステップを決定するために使用される基準を記述します。

ワークフローの条件は、アイコン  で表されます。

サブワークフローの呼び出し

現在のデータワークフローを一時停止し、1 つ以上の他のデータワークフローを起動するデータワークフローのステップ。同じサブワークフロー呼び出しステップによって複数のサブワークフローが呼び出された場合、それらは同時に並行して実行されます。

待機タスク

現在のワークフローを一時停止し、特定のイベントを待機するデータワークフローのステップ。イベントを受信すると、ワークフローが再開され、自動的に次のステップに進みます。

データコンテキスト

ステップ間の継続性を確保するために、データワークフロー全体でステップ間で共有できるデータのセット。

データワークフロー

メインドキュメントセクション [データワークフロー](#)

ワークフローパブリケーション

適切な権限を持つユーザーが実行できるようになっているワークフローモデルのインスタンス。

「ワークフローパブリケーション」の以前の名前 (バージョン5より前) は「ワークフロー」でした。

データワークフロー

ワークフローモデルの実行されたインスタンス。ユーザータスク、スクリプトタスク、条件など、モデルで定義されているデータ処理ステップを実行します。

[ワークフローモデル](#)も参照してください。

関連する概念は、[データワークフロー](#)です。

「データワークフロー」の以前の名前 (バージョン 5 より前) は「ワークフローインスタンス」でした。

作業リスト

現在のユーザーが表示する権限を持っているすべての公開データワークフローのリスト。データワークフローを起動する権限を持つユーザーは、各自の「作業リスト」から起動します。ユーザーからのアクションを必要とするすべての未処理の作業項目は、作業リストの公開ワークフローの下に表示されます。さらに、ユーザーがデータワークフローの管理者である場合、ユーザーは「作業リスト」でそれらのデータワークフローの実行状態を表示でき、必要に応じて介入することができます。

作業項目

ユーザータスクの一部として人間のユーザーが実行する必要のあるアクション。

割り当てられた作業項目は、アイコン  で表されます。

[ユーザータスク](#)も参照してください。

トークン

トークンはデータワークフロー管理中に使用され、リポジトリ管理者に表示されます。

データサービス

[メインドキュメントセクション データサービス](#)

データサービス

EBX® は、XML Web サービスを使用して、[サービス指向アーキテクチャ \(SOA\)](#) に従ってマスターデータを共有します。すべてのデータサービスはモデルまたはビルトインサービスから直接生成されるため、ユーザーインターフェイスから利用できる機能の一部にアクセスするために使用できます。

データサービスの提供

- 通信インターフェイスを構築するための WSDL モデル駆動型のビルトインジェネレーター。これは、ユーザーインターフェイスまたはクライアントアプリケーションの HTTP(S) コネクタを介して作成できます。XML メッセージは EBX® エントリポイントに伝達されます。
- 外部システムが EBX® リポジトリと対話できるようにする SOAP メッセージ用の SOAP コネクタまたはエントリポイントコンポーネント。このコネクタは、EBX® によって生成された WSDL からの要求に応答します。このコンポーネントは、EBX® WSDL ジェネレーターに対応するすべての SOAP XMLメッセージを受け入れます。
- RESTful コネクタ、または Select 操作のエントリポイントにより、外部システムが EBX® リポジトリに問い合わせることができます。認証後、URL で定義されたリクエストを受け入れ、認証されたユーザーの権限に従って実行します。

リネージ

データサービスにアクセス権プロファイルを実装するメカニズム。次に、アクセス権プロファイルを使用して、WSDL インターフェイスを介してデータにアクセスします。

関連する概念：[リネージの WSDL の生成](#)

クロスドメイン

ノード

ノードは、ツリービューまたはグラフのエレメントです。EBX® では、「ノード」は使用状況に応じていくつかの意味を持ちます。

- [ワークフローモデル](#)のコンテキストでは、ノードはワークフローのステップまたは条件です。
- [データモデル](#)のコンテキストでは、ノードはグループ、テーブル、またはフィールドです。
- [階層](#)コンテキストでは、ノードはディメンションの値を表します。
- [適応ツリー](#)では、ノードはデータセットです。
- [データセット](#)では、ノードはデータセットまたはレコードのコンテキストで評価されたデータモデルのノードです。

[ユーザーガイド 目次](#)

データモデル

- [データモデルの概要](#)

- データモデルの実装
 - データモデルの作成
 - データモデルの構成
 - データモデル構造の実装
 - データモデルエレメントのプロパティ
 - エレメントのデータ検証コントロール
 - データモデルの拡張機能
 - 既存のデータモデルの操作
- データモデルの公開とバージョン管理
 - データモデルの公開
 - 埋め込みデータモデルのバージョン管理

データモデルの概要

- 概要
 - データモデルとは
 - データモデリングで使用される基本概念
- データモデル領域のユーザーインターフェイスの使用
 - データモデルアシスタント内のナビゲート
 - データモデルエレメントのアイコン

概要

データモデルとは

TIBCO EBX® でデータを管理するための最初のステップは、データモデルを開発することです。データモデルの目的は、組織、データ型、および意味関係の観点から、リポジトリで管理されるデータの詳細な構造定義を提供することです。

リポジトリにデータモデルを実装するには、最初に新しいデータモデルを作成し、次にそのコンポーネントテーブル、フィールド、グループエレメントの構造の詳細、およびそれらの動作とプロパティを定義します。リポジトリへのデータモデル構造の入力またはインポートを完了後、データセットで使用できるように公開します。データモデルの公開が完了すると、ユーザーはそれに基づ

づいてデータセットを作成し、EBX® リポジトリによって管理されるデータを含めることができます。

データモデリングで使用される基本概念

データモデルの作成を進めるには、次の用語の基本的な理解が必要です。

- フィールド
- 主キー
- 外部キー
- テーブル (データモデル内)
- group
- 再利用可能な型
- 検証ルール

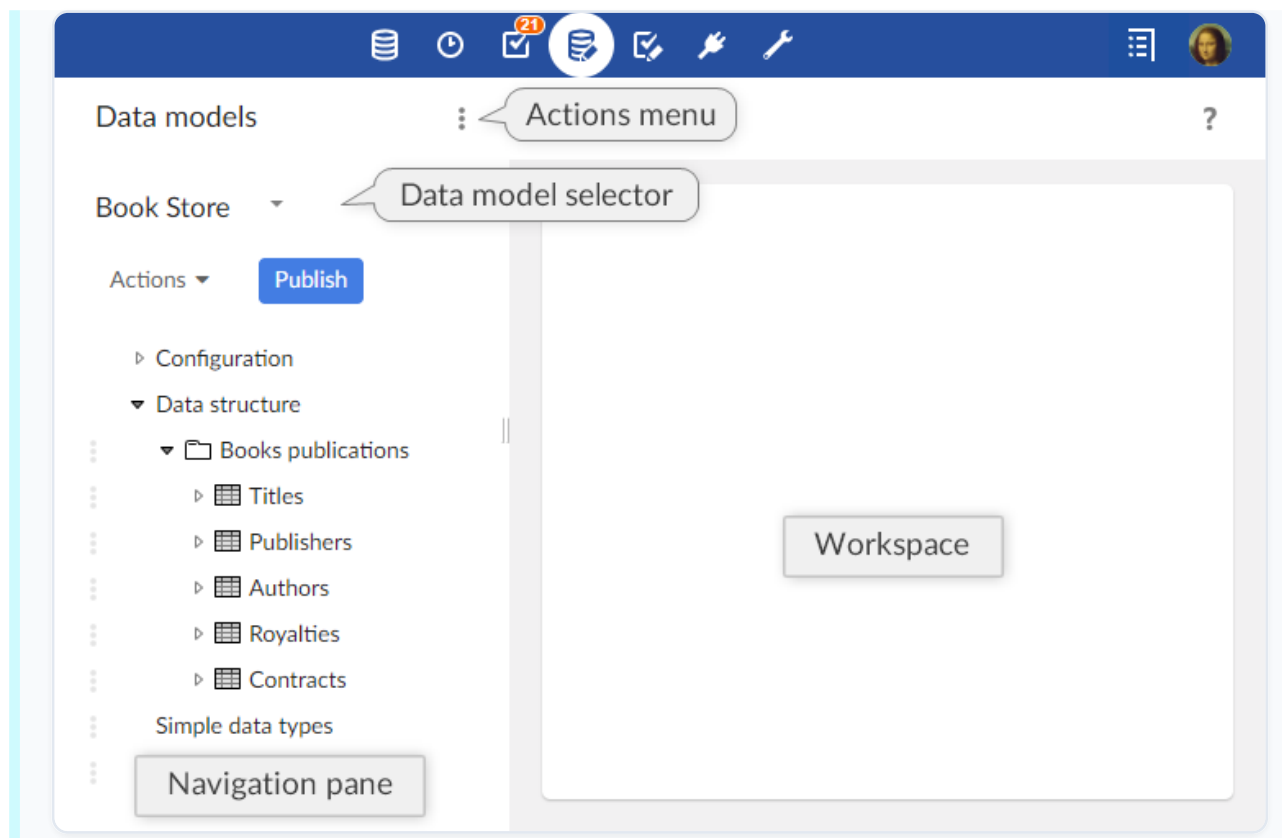
データモデル領域のユーザーインターフェイスの使用

データモデルアシスタント内のナビゲート

データモデルは、ユーザーインターフェイスのデータモデル領域で作成、編集、インポート、および公開できます。EBX® データモデルアシスタント (DMA) は、データモデルの開発を容易にします。

注意:

この領域は、で許可されたユーザーのみが使用できます。



ナビゲーションペインは、次のセクションで構成されています。

構成	データモデルの技術的構成。
グローバルプロパティ	データモデルのグローバルプロパティを定義します。
包含データモデル	現在のモデルに含まれるデータモデルを定義します。含まれているデータモデルで定義されているすべてのタイプは、現在のモデルで再利用できます。
コンポーネントライブラリ	モデルで使用可能な Java コンポーネントを定義します。これらは、プログラムによる制約、関数、UI Bean など、モデルで使用できるプログラマ的な機能を提供します。
アドオン	データモデルで使用するアドオンを指定します。これらのアドオンには、データモデルの要素にプロパティと制約を追加することにより、公開後に現在のデータモデルを強化する機能があります。

データ構造	データモデルの構造。データモデルのエレメント間の関係を定義し、各エレメントの定義へのアクセスを提供します。
単純データ型	現在のデータモデルで定義されている単純な再利用可能型。
複合データ型	現在のデータモデルで定義されている再利用可能な複合型。
包含単純データ型	含まれている外部データモデルで定義された単純な再利用可能型。
包含複合データ型	含まれている外部データモデルで定義されている再利用可能な複合型。
拡張機能	現在のデータモデルで利用可能な拡張機能。
ツールバー	データモデルで使えるツールバー。
ユーザーサービス	リリース 5.8.0 より前に利用可能な API を使用して、ユーザーサービスを宣言します。リリース 5.8.0 以降、新しい UserService API を使用することをお勧めします (これらのサービスは Java API を介して直接登録されるため、データモデルアシスタントで宣言する必要はありません)。
エンティティマッピング	データモデル内のテーブルのエンティティ名を指定します。これらの名前は、SQL リクエストのエイリアスとして、または WSDL 操作のサフィックスとして使用されます。これにより、パスの代わりに一意の名前を使用して、SQL リクエストおよびデータサービス操作でテーブルを参照できます。
レプリケーション	このテーブルは、データモデルのレプリケーションユニットを定義します。レプリケー

	シヨユニットを使用すると、リレーショナルデータベース内のソーステーブルをレプリケーションできるため、外部システムはプレーンな SQL リクエストとビューを使用してこのデータにアクセスできます。
Ajax コンポーネント	モデルで使用可能な Ajax コンポーネントを定義します。
Java バインディング	バインディングは、モデルから生成する必要がある Java タイプを指定します。

以下も参照してください。:

- データモデル構造の実装
- データモデルの構成
- 再利用可能なタイプ
- データモデルの拡張機能

データモデルエレメントアイコン

 フィールド

 主キー

 外部キー

 テーブル

 グループ

関連する概念:

- データスペース
- データセット

[ユーザーガイド 目次](#)

データモデルの実装

- データモデルの作成
- データモデルの構成
- データモデル構造の実装
- データモデルエレメントのプロパティ
- エレメントのデータ検証コントロール
- データモデルの拡張機能
- 既存のデータモデルの操作

データモデルの作成

新しいデータモデルの作成

新しいデータモデルを作成するには、ポップアップの ボタンをクリックして、ウィザードを実行します。

[ユーザーガイド 目次](#)

データモデルの構成

- データモデルの関連情報
- 権限
- データモデルのプロパティ
- 包含データモデル
- データモデルで使用するアドオン

データモデルの関連情報

データモデルの所有者とドキュメントを表示および編集するには、ナビゲーションペインでデータモデルの **[データモデルの]** メニューから を選択します。

注意:

この領域は、 で許可されたユーザーのみが使用できます。

一意名	データモデルの一意の名前。データモデルが作成されると、この名前を変更することはできません。
所有者	データモデルの情報を編集し、その権限を定義する権限を持つデータモデルの所有者を指定します。
ローカライズドキュメント	データモデルのローカライズされたラベルと説明。

権限

データモデルのユーザー権限を定義するには、ナビゲーションペインのデータモデルの **[データモデルの]** メニューから を選択します。

権限 で説明されているように、データモデルの権限の構成は、データセットの権限のオプションと同じです。

データモデルのプロパティ

ナビゲーションペインの > で、次の技術プロパティにアクセスできます。

モジュール名	このデータモデルで使用するリソースを含むモジュールを定義します。これは、モジュールに公開する場合にデータモデル公開で使用するターゲットモジュールでもあります。
モジュールパス	サーバーのファイルシステム上のモジュールの物理的な場所。
ソースロケーション	「コンポーネントライブラリ」で Java コンポーネントを構成するときに使用するソースパス。このパスが相対パスの場合、「モジュールパス」をベースパスとして使用して解決されます。

パブリケーションモード	データモデルをモジュール内の XML スキーマドキュメントとして公開するか、TIBCO EBX® リポジトリに完全に埋め込まれたパブリケーションとして公開するかを指定します。埋め込まれたデータモデルでは、パブリケーションのバージョン管理やロールバックなどの追加機能が提供されます。 詳細については、パブリケーションモードを参照してください。 モジュール内のモデルパス：データモデル生成のターゲットファイルを定義します。'/'で始まる必要があります。
データセットの継承	このデータモデルでデータセットの継承を有効にするかどうかを指定します。データセットの継承はデフォルトで無効になっています。 詳細については、データセットの継承を参照してください。
ドキュメント	Java クラスによって定義されたデータモデルのドキュメント。この Java クラスは、データモデルの要素のラベルと説明をプログラムで指定できます。この Java クラスで定義されたラベルと説明は、要素にローカルに定義されたものよりも優先して、関連付けられたデータセットに表示されます。 詳細については、動的ラベルと説明を参照してください。
特殊拡張	Java クラスのプログラムルールによって定義されたアクセス権限。

自動インクリメント確認の無効化	更新中のテーブルで見つかった「最大値」に関して、関連するデータセットの自動インクリメントされたフィールド値のチェックを無効にするかどうかを指定します。 詳細については、自動インクリメントされた値を参照してください。
ユーザーサービスの有効化 (旧 API)	リリース 5.8.0 より前に利用可能な API を使用するユーザーサービスを宣言できるかどうかを指定します。[いいえ] の場合、[構成] > [ユーザーサービス] セクションは表示されません (少なくともサービスがこのセクションですでに宣言されている場合を除く)。リリース 5.8.0 以降では、新しい UserService Java API を使用することをお勧めします (これらのサービスは Java API を通じて直接登録されるため、データモデルアシスタントでの宣言は必要ありません)。 詳細については、UserServiceDeclaration を参照してください。

包含データモデル

テーブルの > で他のデータモデルのエントリを追加することにより、別のデータモデルで定義されている現在のモデルのデータ型を使用できます。

この表に含まれているモデルのレコードにアクセスすると、タブにモデルに関する技術情報が表示されます。包含データモデルでは、たとえば Java リソースが削除されたために、最終的に検証エラーが発生する可能性があるため、このビューにはこれらの問題に関する情報が表示されません。

検証エラーがなく、埋め込みデータモデルとして定義および公開されているか、モジュールにパッケージ化されているデータモデルのみを含めることができます。

データ型の名前は、ローカルで定義された型定義と包含型定義の両方で一意にする必要があります。つまり、包含データ型の名前は、現在のデータモデルまたは他の包含データモデルで定義されているデータ型の名前と一致してはなりません。

以下も参照してください。:

- [外部データモデルの包含](#)

データモデルで使用されるアドオン

どのデータモデルでも、現在のデータモデルで使用される アドオンを指定できます。これらのアドオンは、データモデルエレメントにプロパティと制約を追加することで、公開後に現在のデータモデルを充実させる機能を持ちます。

ユーザーインターフェイスを通じてデータモデルで使用されるアドオンを定義するには、ナビゲーションウィンドウのデータモデル構成の下にある テーブルに新しいレコードを作成します。このテーブルのレコードは、次のプロパティを定義します。

名称	アドオンの公開名。
バージョン	アドオンバージョン。
有効化済み	アドオンがアクティブ化されているかどうかを示します。アドオンを使用するにはアクティブ化する必要があります。

[ユーザーガイド](#) [目次](#)

データモデル構造の実装

データモデルの構造定義を操作するには、ナビゲーションペインで操作しているデータモデルを選択します。

次に、 の下のナビゲーションペインでデータモデルの構造にアクセスして、フィールド、グループ、およびテーブルの構造を定義できます。

- [一般的なアクションとプロパティ](#)
 - [データモデルへのエレメントの追加](#)
 - [データモデルのエレメントの削除](#)
 - [既存のエレメントの複製](#)
 - [エレメントの移動](#)

- 再利用可能なタイプ
 - 再利用可能なタイプの定義
 - 再利用可能なタイプの使用
 - 他のデータモデルで定義されたデータ型を含む
- データモデルエレメント作成の詳細
 - フィールドの作成
 - テーブルの作成
 - グループの作成
 - 主キーフィールドの作成
 - 外部キーフィールドの作成または定義
 - 関連付けの作成
- 既存のエレメントの変更
 - 主キーからのフィールドの削除

一般的なアクションとプロパティ

データモデルへのエレメントの追加

データモデルの構造を説明するために、次のエレメントを使用できます。

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

既存のエントリの右側にある下矢印  をクリックし、メニューからエレメント作成オプションを選択して、データ構造内の既存のエレメントに関連する新しいエレメントを追加します。既存のエレメントがフィールド、グループ、またはテーブルのいずれであるかに応じて、既存のエレメントの子として新しいエレメントを作成するか、同じレベルの既存のエレメントの前または後に新しいエレメントを作成するかを選択できます。次に、エレメント作成ウィザードに従って、新しいエレメントを作成できます。

注意:

エレメント `root` は、データモデルの作成時に常に追加されます。このエレメントの名前を変更する必要がある場合は、削除して新しい名前で作成できます。

名前、ラベル、説明、および情報

データモデルに新しいエレメントを作成するときは常に、データ構造のレベルで一意的なフィールド「名前」の名前を指定する必要があります。この名前は、エレメントが作成されると割り当てられ、後で変更することはできません。

エレメントの一意的な名前の代わりに、ユーザーインターフェイスに表示されるローカライズされたユーザーフレンドリーなラベルと、エレメントの簡単なローカライズされた説明を提供するオプションがあります。一意的な名前とは異なり、ラベルと説明は作成後に変更できます。各ユーザーの言語設定に応じて、TIBCO EBX® は対応するローカライズされたラベルとエレメントの説明を表示します。

データモデルのエレメントの削除

エントリに対応する下矢印  を使用して、任意のエレメントをデータ構造から削除できます。

再利用可能なタイプを使用していないグループまたはテーブルを削除する場合、削除は再帰的に実行され、ネストされたエレメントがすべて削除されます。

既存のエレメントの複製

エレメントを複製するには、そのエントリに対応する下矢印  をクリックします。データ構造のレベルで一意的な複製エレメントの名前を指定する必要があります。他のすべてのプロパティは、ソースエレメントからコピーされます。

複製されたエレメントは、コピー元のエレメントと同じレベルでデータモデルに追加され、既存のエレメントの後に追加されます。他のエレメントを含むテーブルまたはグループを複製する場合、ネストされたすべてのエレメントが元の名前でコピーされます。

注意:

主キーフィールドを複製すると、フィールドのプロパティは維持されますが、新しいフィールドは主キーに自動的に追加されません。

エレメントの移動

データ構造の現在のレベル内でエレメントを並べ替えるには、そのエントリに対応する下矢印  をクリックし、 を選択します。次に、現在のエレメントを移動する *前* のフィールドに対応する左矢印ボタンを選択します。

注意:

データ構造内のレベル外の位置にエレメントを移動することはできません。

再利用可能なタイプ

再利用可能なタイプは、一度作成された共有エレメント定義であり、データモデルのさまざまな場所で再利用できます。

注意:

または セクションで再利用可能な型の定義を変更すると、その再利用可能な型に基づいてすべてのエレメントの構造が変更されます。再利用可能なタイプを使用するグループまたはテーブルの構造は、読み取り専用として表示されます。関連する再利用可能な型の構造を編集するには、または セクションから型にアクセスする必要があります。

再利用可能なタイプの定義

ナビゲーションペインの と の下矢印  メニューから、同じ構造の定義とプロパティを共有するエレメントをさらに作成するために使用できる単純型および複合型の再利用可能な型を定義できます。または、対応する下矢印  メニューを使用して、既存のテーブルとグループを再利用可能なタイプに変換することもできます。

各データ型の特定のページで、ナビゲーションペインの と の下にある を選択すると、再利用可能な型を使用しているエレメントを確認できます。次に、このタイプに基づくすべてのエレメントがテーブルに表示されます。データ型がどのエレメントでも使用されていない場合は、下矢印  メニューから を選択して再利用可能なタイプを削除できます。

再利用可能なタイプの使用

新しいエレメントの構造は、再利用可能な型を使用して定義できます。これを行うには、エレメント作成フォームで既存の再利用可能なタイプを選択します。作成されたエレメントは、再利用可能な型の定義を共有します。

他のデータモデルで定義されたデータ型を含む

複数のデータモデル間で再利用可能な型を共有することもできます。外部データモデルの包含を構成することにより、そのデータモデルで定義されたデータ型を使用して、ローカルで定義された再利用可能な型を使用するのと同じ方法でデータ構造に新しいエレメントを作成できます。

注意:

データ型の名前は、ローカルで定義されたすべての型と含まれているすべての型で一意である必要があるため、包含データモデルのデータ型と同じ名前で新しい再利用可能な型を作成することはできません。同様に、ローカルで定義された再利用可能なタイプと同じ名前のデータ型を定義する外部データモデル、または包含される別のデータモデルのデータ型を含めることはできません。

包含データ型は、ナビゲーションパネルの および のセクションに表示されます。これらの包含される再利用可能なタイプの詳細を表示できます。ただし、元のデータモデルでのみローカルで編集できます。

詳細については、[包含データモデル](#) を参照してください。

データモデルエレメント作成の詳細

フィールドの作成

新しいフィールドを作成するときは、そのデータ型を選択する必要があります。これにより、このフィールドに基づいて値のデータ型が定義されます。フィールドが作成されると、フィールドのデータ型を変更することはできません。

フィールドを作成するときに、外部キー、必須フィールド、およびテーブルの下に作成された場合は主キーとして指定することもできます。

テーブルの作成

テーブルの作成中に、既存の再利用可能なタイプに基づいて新しいテーブルを作成するオプションがあります。詳細については、[再利用可能なタイプ](#) を参照してください。

すべてのテーブルには、ナビゲーションペインからテーブルの子エレメントとして作成できる少なくとも 1 つの主キーフィールドを指定する必要があります。

グループの作成

グループの作成中に、既存の再利用可能なタイプに基づいて新しいグループを作成するオプションがあります。詳細については、[再利用可能なタイプ](#) を参照してください。

主キーフィールドの作成

すべてのテーブルに少なくとも 1 つの主キーが必要です。「データ構造」ツリーのテーブルのエントリの下に子エレメントとしてテーブルを作成することにより、テーブルの主キーフィールドを作成できます。

主キーとして直接新しいフィールドを作成することに加えて、テーブルの の タブの主キーの定義に、テーブルの既存の子フィールドを追加できます。

外部キーフィールドの作成または定義

外部キーフィールドのデータ型は「文字列」です。「データ構造」ツリーのテーブルのエントリの下に子エレメントとして作成することにより、テーブルの外部キーフィールドを作成できます。

「文字列」型の既存のフィールドを外部キーに変換することもできます。「文字列」型の既存のフィールドを外部キーに変換するには、フィールドの を有効にし、関連するパラメーターを定義します。

外部キーを直接作成する場合でも、既存のフィールドから作成する場合でも、参照するレコードを含むテーブルを定義する必要があります。

関連付けの作成

関連付けにより、テーブル間のセマンティックリンクを定義できます。「データ構造」ツリーのテーブルのエントリの下に子エレメントとして関連付けを作成し、フォームで を選択して新しいエレメントを作成することにより、関連付けを作成できます。関連付けは、テーブル内でのみ定義できます。既存のフィールドを関連付けに変換することはできません。

関連付けを作成するときは、関連付けのタイプを指定する必要があります。次のように、いくつかのオプションが利用可能です。

- **外部キー** の逆の関係。この場合、関連付けエレメントは ソーステーブル で定義され、ターゲットテーブル を参照します。これは、ターゲットテーブルで定義され、ソーステーブルを参照する外部キーフィールドの対応物です。関連付けの親テーブルを参照する外部キーを定義する必要があります。
- **リンクテーブル** を介して。この場合、関連付けエレメントは ソーステーブル で定義され、リンクテーブル から推測される ターゲットテーブル を参照します。このリンクテーブルは、2 つの外部キーを定義します。1 つはソーステーブルを参照し、もう 1 つはターゲットテーブル

を参照します。リンクテーブルの主キーは、自動インクリメントされたフィールドや、関連付けのソーステーブルまたはターゲットテーブルへの外部キーも参照する必要があります。リンクテーブルとこれら 2 つの外部キーを定義する必要があります。

- XPath 述語を使用します。この場合、関連付け要素はソーステーブルで定義され、パスを使用して指定されたターゲットテーブルを参照します。XPath 式も定義され、現在のテーブルのレコードをターゲットテーブルのレコードに関連付けるために使用される基準を指定します。ターゲットテーブルと XPath 式を定義する必要があります。

すべてのタイプの関連付けで、ソーステーブルのレコードにセマンティックにリンクされているターゲットテーブルのレコードを **関連付けられたレコード** と呼びます。

関連付けを作成後、追加のプロパティを指定できます。関連付けの場合、次のことが可能です。

- 追加の XPath フィルターを指定して、関連付けられたレコードにフィルターを設定します。XPath フィルターを定義する場合は、ソーステーブルとターゲットテーブルのフィールドのみを使用できます。つまり、リンクテーブル以外の関連付けである場合、XPath フィルターでリンクテーブルのフィールドを使用することはできません。使用可能なウィザードを使用して、XPath フィルターで使用するフィールドを選択できます。
- 表形式のビューを構成して、関連するテーブルに表示する必要があるフィールドを定義します。関連付けのターゲットテーブルが存在しない場合、既存の表形式ビューを構成または変更することはできません。表形式のビューが定義されていない場合、付与されたアクセス権に従ってユーザーが表示できるすべての列が表示されます。
- 関連するレコードをフォームにレンダリングする方法を定義します。関連するレコードをフォームに直接レンダリングするか、特定のタブにレンダリングするかを指定できます。デフォルトでは、関連付けられたレコードは、親テーブルの関連付けと同じ位置にフォームでレンダリングされます。
- データサービスの「Select」操作で関連レコードを非表示/表示します。デフォルトでは、関連付けられたレコードはデータサービスの「選択」操作で非表示になります。
- 必要な関連レコードの最小数と最大数を指定します。関連付けられたデータセットでは、関連付けが必要な最小数または最大数の関連付けられたレコードに準拠していない場合、指定された重大度の検証メッセージが追加されます。デフォルトでは、関連するレコードの必要な最小数と最大数は制限されていません。
- XPath 述語を使用して検証制約を追加し、関連するレコードを制限します。XPath 述語を定義する場合は、ソーステーブルとターゲットテーブルのフィールドのみを使用できます。つまり、リンクテーブルを介した関連付けである場合、XPath 述語でリンクテーブルのフィールドを使用することはできません。使用可能なウィザードを使用して、XPath 述語で使用する

フィールドを選択できます。関連するデータセットでは、関連するレコードが指定された制約に準拠していない場合、指定された重大度の検証メッセージが追加されます。

既存の要素の変更

主キーからのフィールドの削除

主キーに属するフィールドはすべて、テーブルの の タブで主キーから削除できます。

用語集の [主キー](#) を参照してください。

[ユーザーガイド 目次](#)

データモデル要素のプロパティ

要素を最初に作成した後、その定義を完了するために追加のプロパティを設定できます。

以下も参照してください。:

- [要素のデータ検証コントロール](#)
- [基本的な要素のプロパティ](#)
 - [一般的な基本プロパティ](#)
 - [フィールドの基本プロパティ](#)
- [要素の詳細プロパティ](#)
 - [一般的な詳細プロパティ](#)
 - [フィールドの詳細プロパティ](#)
 - [テーブルの詳細プロパティ](#)
 - [グループの詳細プロパティ](#)

基本的な要素のプロパティ

一般的な基本プロパティ

次の基本プロパティは、いくつかのタイプの要素で共有されます。

情報	エレメントに関連付けられた追加の非国際化情報。
値の最小数	エレメントの値の最小数。 主キーは複数値にすることができないため、このプロパティを [1] または [未定義] に設定する必要があります。フィールドを選択ノードとして定義する場合、値の最小数は自動的に [0] に設定されます。
値の最大数	エレメントの値の最大数。[1] より大きい値に設定すると、エレメントは複数値になります。 主キーは複数値にすることができないため、このプロパティを [1] または [未定義] に設定する必要があります。 テーブルの場合、値の最大数は作成時に自動的に [無制限] に設定されます。フィールドを選択ノードとして定義する場合、値の最大数は自動的に [0] に設定されます。
検証ルール	このプロパティは、[パスワード] フィールド、再利用可能型、複雑な再利用可能型のフィールド、および選択ノードを除く、テーブルおよびテーブル内のフィールドで使用できます。提供されている XPath 1.0 基準エディターを使用して、強力な複雑な検証ルールを定義するために使用されます。 詳細については、基準エディターを参照してください。 これは、値の検証が複雑な基準または他のフィールドの値に依存する場合に役立ちます。

特定の状況下で必須の値の検証がルールで定義されていることを示すこともできます。この場合、ルールが満たされない場合、値は必須です。詳細については、[「null」値の制約](#)を参照してください。

関連するウィザードを使用して、検証ルールのローカライズされたラベルを定義したり、基準が満たされていない場合に表示される重大度のあるローカライズされたメッセージを定義したりできます。

検証メッセージの重大度を定義するときに、フォームの送信時に検証ルールに違反する入力が拒否されるかどうかを示すことができます。エラー管理ポリシーは、フィールドで定義された検証ルールで、重大度が「エラー」に設定されている場合にのみ使用できます。検証ルールを有効なままにする必要がある場合、ルールに違反する入力はすべて拒否され、値は変更されません。エラーが許可されている場合、ルールに違反する入力はすべて受け入れられ、値が変更されます。指定しない場合、検証ルールはデフォルトでフォーム送信時に常にエラーをブロックします。

検証ルールがテーブルで定義されている場合、それは「テーブルの制約」と見なされ、テーブルの各レコードは実行時にそれに対して評価されます。詳細については、[テーブルの制約](#)を参照してください。

フィールドの基本プロパティ

次の基本プロパティは、フィールドに固有です。

デフォルト値	このフィールドに割り当てられたデフォルト値。新しいデータ作成フォームでは、デフォルト値がユーザー入力フィールドに自動
--------	--

	的に表示されます。デフォルト値は、フィールドの定義されたタイプに準拠している必要があります。 詳細については、デフォルト値を参照してください。
変換エラーメッセージ	このフィールドのデータ型に対して無効な値をユーザーが入力したときにユーザーに表示する国際化されたメッセージ。
計算ルール	このプロパティは、再利用可能型を除いて、テーブルのフィールドで使用できます。提供されている XPath 1.0 エディターを使用してフィールドの値を計算するためのルールを定義します。 基準エディターを参照してください。 これは、値が同じレコード内の他の値に依存しているが、プログラムによる計算を必要としない場合に役立ちます。 計算ルールには、次の制限があります。 • 計算ルールは、テーブル内の単純なフィールドでのみ定義できます。 • タイプ <code>OResource</code> または <code>Password</code> のフィールドに計算ルールを定義することはできません。 • 選択ノードと主キーフィールドで計算ルールを定義することはできません。 • 検証レポートからエレメントにアクセスする場合、計算ルールを定義することはできません。

エレメントの詳細プロパティ

一般的な詳細プロパティ

次の詳細プロパティは、いくつかのタイプのエレメントで共有されます。

デフォルトビューおよびツール > 可視性

このエレメントをデータセットのデフォルトビュー、データセットのテキスト検索、またはデータサービスの「選択」操作に表示するかどうかを指定します。

- モデル駆動ビュー

現在のエレメントがテーブルの場合に、テーブルのデフォルト表形式ビュー、テーブルのデフォルトレコード形式、データセットのデフォルトビューに表示するかどうかを指定します。データモデルの構造から生成されたデータセットのデフォルトビュー、表形式のビュー、デフォルトレコード形式。現在のエレメントがテーブルの中にある場合、プロパティを [非表示] に設定すると、特定のアクセス権限を定義しなくても、テーブルのデフォルトの表形式のビューとデフォルトのレコードフォームからエレメントが隠されます。現在のエレメントは、このエレメントを表示するカスタムビューを作成できるように、ビュー構成ウィザードに表示されます。現在のエレメントがテーブルの場合、このプロパティを [非表示] に設定すると、特定のアクセス権限を定義しなくても、データセットのデフォルトビューからテーブルが非表示になります。このプロパティは、テーブルでもテーブル内でもないエレメントに設定された場合は無視されます。

- すべてのビュー

現在のエレメントをデータセット内のテーブルのすべてのビューに表示するかどうかを指定します。プロパティを [すべてのビューで非表示] に設定すると、特定のアクセス権限を定義しなくても、表形式 (デフォルトの表形式のビューを含む) または階層のいずれの場合でも、テーブルのすべてのビューでエレメントが非表示になります。現在のエレメントは、ビュー構成ウィザードでも非表示になります。つまり、このエレメントを表示するカスタムビューを作成することはできません。このプロパティは、テーブルにないエレメントに設定されている場合は無視されます。このプロパティはフォームには適用されません。つまり、プロパティを [すべてのビューで非表示] に設定すると、レコード形式ではエレメントが非表示にならず、ビューでのみ非表示になります。

- 構造化検索ツール

現在のエレメントをデータセット構造化検索ツールに表示するかどうかを指定します。プロパティを [構造化検索で非表示] に設定すると、データセットの構造化検索ツールでエレメントが非表示になります。エレメントは、クイック検索ツールで引き続き検索できます。このプロパティがテーブルにないエレメントに設定されている場合、このプロパティは無視されます。

- クイック検索

現在のエレメントをクイック検索ツールに含めるかどうかを指定します。プロパティを [クイック検索から除外] に設定す

	ると、データセットのクイック検索ツールからエレメントが除外されます。エレメントは構造化検索ツールで引き続き検索可能です。このプロパティは、テーブルにないエレメントに設定されている場合は無視されます。 • データサービス 現在のエレメントをデータサービスの Select 操作に表示するかどうかを指定します。プロパティを [データサービスから除外] に設定すると、データサービスの「Select」操作でエレメントが非表示になります。このプロパティは、テーブルにないエレメントに設定されている場合は無視されます。 『開発者ガイド』のデフォルトビューを参照してください。
デフォルトのビューとツール > ウィジェット	使用するウィジェットを定義します。ウィジェットは、関連するデータセットのフォームに表示される入力コンポーネントです。未定義の場合、現在のエレメントのタイプとプロパティに応じて、関連付けられたデータセットにデフォルトのウィジェットが表示されます。ビルトインウィジェットまたはカスタムウィジェットを使用することができます。カスタムウィジェットは、Java API を使用して定義され、フィールドまたはグループの豊富なユーザーインターフェイスコンポーネントの開発を可能にします。ビルトインウィジェットとカスタムウィジェットは、テーブルまたは関連付けで定義できません。カスタムウィジェットと UI Bean の両方を定義することは禁止されています。外部キーフィールドにカスタムウィ

	ジェットとコンボボックスセクターの両方を定義することは禁止されています。 詳細については、 <code>UIWidgetFactory</code> を参照してください。
デフォルトのビューとツール > コンボボックスセクター	外部キーのコンボボックス選択で使用される公開ビューの名前を指定します。ドロップダウンリストの右下隅に選択ボタンが表示されます。外部キーを定義する場合、この機能を使用すると、[セクター] ボタンを使用して高度な選択ビューにアクセスできます。このボタンから詳細選択ビューが開き、そこから並べ替えと検索のオプションを使用できます。公開されたビューが定義されていない場合、詳細選択ビューは無効になります。参照されるテーブルのパスが絶対である場合、このテーブルに対応する公開されたビューのみが表示されます。参照されるテーブルのパスが相対である場合、ターゲットテーブルを含むデータモデルに関連付けられているすべての公開されたビューが表示されます。このプロパティは、カスタムウィジェットが定義されていない場合にのみ設定できます。 『開発者ガイド』の外部キーのコンボボックスセクターのビューの定義を参照してください。
UI bean	注意: TIBCO EBX® バージョン 5.8.0 以降では、UI Bean の代わりにウィジェットを使用することをお勧めします。ウィジェットは UI Bean よりも多くの機能を提供し、UI Bean でそれ以上のアップデートは行われません。詳細については、 ウィジェット を参照してください。

	このプロパティは、テーブルと関連付けを除くすべてのエレメントで使用できます。データセット内のこのエレメントに関連するユーザーインターフェイスをカスタマイズするための Java クラスを指定します。UI Bean は、Java API の <code>UIBeanEditor</code> クラスを拡張することによって、エレメントを別の方法で表示したり、その値を変更したりすることができます。
エクスポート時の変換	このプロパティは、フィールドおよびターミナルノードであるグループで使用できます。関連するデータセットをアーカイブとしてエクスポートするときに実行される変換操作を定義する Java クラスを指定します。変換の入力は、このエレメントの値です。 詳細については、<code>NodeDataTransformer</code> を参照してください。
アクセスプロパティ	現在のエレメントのアクセスモードを定義します。つまり、そのデータを読み書きできるかどうかを定義します。 • [読み取りと書き込み] は、データモデル XSD のモード <code>RW</code> に対応します。 • [読み取り専用] は、データモデル XSD のモード <code>R-</code> に対応します。 • [非データセットノード] は、データモデル XSD のモード <code>CC</code> に対応します。 • [非ターミナルノード] は、データモデル XSD のモード <code>--</code> に対応します。 『開発者ガイド』のアクセスプロパティを参照してください。

比較モード	エレメントに関連付けられた比較モードを定義します。これは、データセットでその違いを検出する方法を制御します。 • [デフォルト] は、関連データを比較するときにエレメントが表示されることを意味します。 • [無視] は、変更されたコンテンツ (レコードまたはデータセット) の 2 つのバージョンを比較するときに変更が検出されないことを意味します。 マージ中、コンテンツが変更されていても、無視されたエレメントのデータ値はマージされません。ただし、操作中に作成された無視されたデータセットまたはレコードの値はマージされます。 アーカイブのインポート中、コンテンツが変更されている場合、無視されたエレメントの値はインポートされません。ただし、操作中に作成された無視されたデータセットまたはレコードの値はインポートされます。 『開発者ガイド』の比較モードを参照してください。
最終変更ポリシーの適用	このエレメントをサービスから除外して、レコードに対して実行された最後の変更を同じテーブルの他のレコードに適用できるようにする必要があるかどうかを定義します。 • [デフォルト] は、このエレメントの最後の変更を他のレコードに適用できることを意味します。 • [無視] は、このエレメントの最後の変更を他のレコードに適用できないことを意

	味します。このエレメントは、最終変更の適用サービスでは表示されません。 『開発者ガイド』の最終変更ポリシーの適用を参照してください。
ノードカテゴリ	このエレメントのカテゴリを定義します。カテゴリを使用すると、データセット内のデータのユーザーへの可視性を制御できます。 [非表示] カテゴリのノードは、デフォルトで非表示になっています。制約事項：[非表示] 以外のカテゴリ指定は、テーブルレコードノードには適用されません。 『開発者ガイド』のカテゴリを参照してください。

フィールドの詳細プロパティ

次の詳細プロパティは、フィールドに固有です。

null 入力の確認

プロパティ `osd:checkNullInput` を実装します。このプロパティは、ユーザー入力時に `null` の制約をアクティブにしてチェックするために使用されます。

デフォルトでは、一時的に不完全な入力を許可するために、必須エレメントのチェックはユーザー入力では実行されず、データセットの検証時にのみ実行されます。ユーザー入力の直後に必須エレメントをチェックする必要がある場合は、このプロパティを `true` に設定します。

注意:

プロパティが `true` 以上に設定されている場合、値は必須と見なされます。ターミナルエレメントの場合、必須値はアクティブ化されたデータセットでのみチェックされます。非ターミナルエレメントの場合、データセットがアクティブ化されているかどうかに関係なく、値がチェックされます。

『開発者ガイド』の[制約](#)、[トリガー](#)、[関数](#)を参照してください。

空白の削除

空白の削除

プロパティ `osd:trim` を実装します。このプロパティは、ユーザー入力時に先頭と末尾の空白を削除する必要があるかどうかを示すために使用されます。このプロパティが設定されていない場合、ユーザー入力時に先頭と末尾の空白が削除されます。

『開発者ガイド』の[ユーザー入力時の空白の処理](#)を参照してください。

UI bean

[一般的な詳細プロパティ](#)を参照してください。

関数 (計算値)

このプロパティは、非主キーフィールドで使用できます。このフィールドの値をプログラムで計算する Java クラスを指定します。これは、フィールドの値がリポジトリ内の他の値に依存している場合、または値の計算でサードパーティのシステムからデータを取得する必要がある場合に役立ちます。

関数は、`ValueFunction` インターフェイスを実装することで作成できます。

検証の無効化

フィールドに定義された制約を無効にする必要があるかどうかを指定します。このプロパティは、関数フィールドでのみ定義できます。true の場合、関連するデータセットを検証するときに、フィールドで定義されたカーディナリティ、単純および高度な制約はチェックされません。

エクスポート時の変換

[一般的な詳細プロパティ](#)を参照してください。

アクセスプロパティ

[一般的な詳細プロパティ](#)を参照してください。

自動インクリメント

このプロパティは、テーブルに含まれるタイプ「整数」のフィールドでのみ使用できます。設定すると、新しいレコードが作成されるときにフィールドの値が自動的に計算されます。これは、レコードごとに一意の識別子を生成するため、主キーに役立ちます。2つの属性を指定できます。

開始値	自動インクリメントを開始するための値。 この属性が指定されていない場合、デフォルト値は [1] です。
インクリメントステップ	値は、自動インクリメントの前の値に基づいて増加されます。この属性が指定されていない場合、デフォルトの値は [1] です。
自動インクリメント確認の無効化	更新されるテーブルの最大値に対して、関連付けられたデータセットの自動インクリメント後のフィールド値のチェックを無効にするかどうかを指定します。

自動インクリメントされた値の動作は次のとおりです。

- フィールド値の計算と割り当ては、新しいレコードが挿入され、フィールド値がまだ定義されていない場合に実行されます。
- プログラムによる挿入で `null` 以外の値がすでに指定されている場合、割り当ては実行されません。したがって、非表示モードまたは上書きモードでのレコード挿入に対しては、割り当ては実行されません。
- アーカイブインポートで値が指定されている場合は、インポートされた値が優先されます。
- 可能な場合は常に、新しく割り当てられた値はリポジトリのスコープ内で一意です。

つまり、割り当ての一意性は、リポジトリ内の任意のデータスペースで、このデータモデルに基づくすべてのデータセットにまたがります。異なるデータスペース間の一意性により、子データスペースと親データスペースのマージが容易になり、レコードの主キーに自動インクリメントされた値が含まれている場合の競合が合理的に回避されます。

このポリシーにもかかわらず、特定の値を割り当てる一括更新トランザクションが、同じフィールドに自動インクリメントされた値を割り当てるトランザクションと同時に実行される場合、特定の制限が存在します。異なるデータスペース間にロックがないため、後者のトランザクションが前者のトランザクションですでに設定されている値を割り当てる可能性があります。

『開発者ガイド』の[自動インクリメント値](#)を参照してください。

デフォルトビュー

一般的な詳細[プロパティ](#)を参照してください。

ノードカテゴリ

一般的な詳細プロパティを参照してください。

継承フィールド

フィールド値を自動的にフェッチするために、現在のフィールドから別のテーブルのフィールドへの関係を定義します。

ソースレコード	現在のエレメントからこのフィールドの値を継承するレコードにつながる外部キーまたは空白で区切られた外部キーのシーケンス。このプロパティが指定されていない場合、現在のレコードが継承のソースレコードとして使用されます。
ソースエレメント	このフィールドの値を継承するソースレコード内のエレメントの XPath。ソースエレメントはターミナルであり、「ソースレコード」で記述されたレコードに属している必要があり、そのタイプはこのフィールドのタイプと一致する必要があります。フィールド継承を使用する場合、このプロパティは必須です。

用語集の[継承](#)を参照してください。

詳細については、[継承フィールド](#)も参照してください。

テーブルの詳細プロパティ

次の詳細プロパティは、テーブルに固有です。

テーブル

主キー	テーブルの主キーを構成するテーブル内のフィールドのリスト。[データ構造] ビューのように、ここで主キーフィールドを追加または削除できます。
-----	---

	各主キーフィールドは、テーブルのルートエレメントの下から始まる絶対 XPath 表記で示されます。 主キーに複数のエレメントがある場合、リストは空白で区切られます。たとえば、「/name /startDate」です。
プレゼンテーション	データセット内のこのテーブルのユーザーインターフェイスでレコードを表示する方法を指定します。
プレゼンテーション > レコードラベリング	テーブル内のレコードのデフォルトおよびローカライズされたラベルを提供するフィールドを定義します。 Java クラスを指定して、プログラムでラベルを設定したり、階層にラベルを設定したりすることもできます。この Java クラスは、<code>UILabelRenderer</code> インターフェイスまたは <code>UILabelRendererForHierarchy</code> インターフェイスのいずれかを実装する必要があります。 注意：レコードラベルを表示する場合、関連するデータセットで定義されたアクセス権限は適用されません。アクセス権限の制限のために通常非表示になっているフィールドは、ラベルに表示されます。
プレゼンテーション > フォーム内のグループに対するデフォルトレンダリング	このテーブルに含まれるグループのデフォルトの表示レンダリングモードを指定します。ここで何も定義されていない場合、管理領域で設定されたデフォルトのポリシーがグループの表示に使用されます。 『管理ガイド』のレコードフォーム：ノードのレンダリングモードを参照してください。

グループに対するレンダリングを有効にする

常に使用可能なモードの [展開済み] および [折りたたみ済み] に加えて、テーブル内のグループに対して有効にする表示レンダリングモードを指定します。グループをタブとして表示するオプションを使用するには、テーブルでタブを有効にする必要があります。同様に、グループをリンクとして表示するオプションを使用するには、リンクを有効にする必要があります。

グループに対するデフォルトレンダリング

このテーブルに含まれるグループに使用するデフォルトの表示レンダリングモードを指定します。グループがデフォルトモードを指定しない場合、このテーブルに定義されているデフォルトモードが使用されます。デフォルトのレンダリングモードを [リンク] として定義するには、リンクを有効にする必要があります。ネットワークとブラウザのパフォーマンスに応じてレンダリングモードを選択します。リンクモードは、コンテンツが同じページに表示されないため軽量ですが、展開モードと折りたたみモードは重くなります。

注意：テーブルでタブビューが有効になっている場合、リンクを使用していたグループは自動的に折りたたみモードに変換されます。これは、同じユーザーインターフェイスに共存するリンクとタブから生じる固有の表示レイアウトの複雑さを回避するために行われます。

プレゼンテーション> フォームの特定のレンダリング	データセットのレコードフォームをカスタマイズするための特定のレンダリングを定義します。 詳細については、 <code>UIForm</code> および <code>UserServiceRecordFormFactory</code> を参照してください。
ツールバー	このテーブルで使用するツールバーを定義します。 ツールバーは、[構成] > [ツールバー] セクションで編集できます。 表形式ビューの上部：デフォルトテーブルビューの上部で使用するツールバーを定義します。 表形式ビューの行：デフォルトテーブルビューの各行で使用するツールバーを定義します。 タイルビュー項目：タイルビューの各項目で使用するツールバーを定義します。 レコード上部：レコードフォームで使用するツールバーを定義します。 階層上部：テーブルのデフォルト階層ビューで使用するツールバーを定義します。 詳細については、ツールバーを参照してください。
履歴	履歴化をいつ実行するか、および要求される保証のレベルを指定します。使用可能な履歴プロファイルは、[管理] > [履歴とログ] で編集できます。 詳細については、リポジトリ内の履歴構成を参照してください。

特定のフィルター	テーブルにレコード表示フィルターを定義します。
アクション	関連するデータセットのテーブルで許可されるアクションを指定します。デフォルトでは、特定のアクセス権限がデータセットで定義されていない限り、すべてのアクションが許可されます。

一意性の制約

テーブル全体で一意である必要があるフィールドまたはフィールドのセットを示します。

トリガー

レコードの作成、更新、削除など、テーブルに対して変更が実行されたときに自動的に実行されるメソッドを定義する Java クラスを指定します。

データワークフローを開始するためのビルトイントリガーがデフォルトで含まれています。

トリガーのスクリプトを定義するためのビルトイントリガーもデフォルトで含まれています。

『開発者ガイド』の[トリガー](#)を参照してください。

アクセスプロパティ

[一般的な詳細プロパティ](#)を参照してください。

デフォルトビュー

[一般的な詳細プロパティ](#)を参照してください。

ノードカテゴリ

[一般的な詳細プロパティ](#)を参照してください。

グループの詳細プロパティ

次の詳細プロパティは、グループに固有です。

値コンテナクラス (JavaBean)

このグループの子の値を保持する Java クラスを指定します。Java クラスは、JavaBean 標準プロトコルに準拠している必要があります。つまり、グループの各子はクラス内の JavaBean プロパ

ティに対応している必要があり、すべてのプロパティにはゲッターアクセサーとセッターアクセサーが定義されている必要があります。

UI bean

一般的な詳細プロパティを参照してください。

エクスポート時の変換

一般的な詳細プロパティを参照してください。

アクセスプロパティ

一般的な詳細プロパティを参照してください。

デフォルトビュー

可視性	一般的な詳細プロパティを参照してください。
フォーム内のレンダリング	このグループのレンダリングモードを定義します。このプロパティが設定されていない場合、コンテナテーブルで指定されたグループのデフォルトビューが使用されます。[タブ]と[リンク]はそれぞれ、コンテナテーブルで有効になっている場合にのみ使用できます。 タブ位置 この属性は、モデルで定義されているすべてのタブに対するタブの位置を指定します。この位置は、タブの順序を決定するために使用されます。位置が指定されていない場合、タブはデータモデル内のグループの位置に従って表示されます。

ノードカテゴリ

一般的な詳細プロパティを参照してください。

関連する概念:

- エレメントのデータ検証コントロール

ユーザーガイド [目次](#)

エレメントのデータ検証コントロール

エレメントを最初に作成した後、その定義を完了するために追加のコントロールを設定できます。

以下も参照してください。:

- データモデルエレメントのプロパティ
- 単純なコンテンツ検証
- 高度なコンテンツ検証
 - 検証プロパティ

単純なコンテンツ検証

フィールドで次の静的コントロールを使用すると、そのコンテンツに単純な検証ルールを確立できます。特定のフィールドで利用できるコントロールは、そのデータ型によって異なります。

固定長	このフィールドに必要な正確な文字数。
最小の長さ	このフィールドに許可される最小文字数。
最大長	このフィールドに許可される最大文字数。
パターン	フィールドの値が一致する必要がある正規表現パターン。フィールドとそのデータ型の両方のパターンを同時に定義することはできません。
小数点以下の桁数	このフィールドに許可される小数点以下の最大桁数。

最大桁数	この整数または 10 進フィールドに許可される最大合計桁数。
列挙型	このフィールドに事前定義された可能な値のリストを定義します。列挙型がフィールドとそのタイプの両方で定義されている場合、関連するデータセット内のこのフィールドの列挙型は、これら2つの列挙型の共通部分に置き換えられます。
より大きい [定数]	このフィールドに許可される最小値を定義します。
より小さい [定数]	このフィールドに許可される最大値を定義します。

XML スキーマでサポートされるファセットを参照してください。

高度なコンテンツ検証

エレメントに対して次の動的およびコンテキストコントロールを使用して、そのコンテンツの高度な検証ルールを確立できます。特定のエレメントで利用できるコントロールは、エレメントのタイプとそのデータ型 (存在する場合) によって異なります。

以下も参照してください。:

- 動的制約

外部キー制約	
テーブル	外部キーによって参照されるテーブルを定義します。外部キーは、デフォルトで同じデータセット内のテーブルを参照します。また、同じデータスペース内の別のデータセット内のテーブル、または別

	のデータセット内のデータセットを参照することもできます。
モード	外部キーによって参照されるテーブルの場所。 「デフォルト」：現在のデータモデル。 「その他のデータセット」：同じデータスペース内の異なるデータセット。 「その他のデータスペース」：別のデータスペースのデータセット。
参照テーブル	テーブルのを場所記述する XPath 式。たとえば、<code>/root/MyTable</code> です。
参照データセット	テーブルが別のデータセットにある場合は必須です。参照されるテーブルを含むデータセットの一意の名前。
参照データスペース	テーブルが別のデータスペースにある場合は必須です。参照されるテーブルを含むデータスペースの一意の名前。
ラベル	テーブル内のレコードのデフォルトおよびローカライズされたラベルを提供するフィールドを定義します。階層リンクで指定されたラベルの表示をカスタマイズすることもできます。 [XPath 式] が [いいえ] に設定されている場合、プログラムでラベルを設定する Java クラスを指定することもできます。この Java クラスは、JavaAPI の <code>TableRefDisplay</code> インターフェイスを実装する必要があります。

	注意：レコードラベルを表示する場合、関連するデータセットで定義されたアクセス権限は適用されません。アクセス権限の制限のために通常非表示になっているフィールドは、ラベルに表示されます。
フィルター	XPath 式を使用して外部キーフィルターを定義します。 JavaAPI の <code>TableRefFilter</code> インターフェイスを実装する Java クラスを指定することもできます。
より大きい [動的]	このフィールドに許可される最小値を提供するフィールドを定義します。
より小さい [動的]	このフィールドに許可される最大値を提供するフィールドを定義します。
固定長 [動的]	このフィールドに必要な正確な文字数を提供するフィールドを定義します。
最小の長さ [動的]	このフィールドに許可される最小文字数を提供するフィールドを定義します。
最大長 [動的]	このフィールドに許可される最大文字数を提供するフィールドを定義します。
除外された値	このフィールドで許可されていない値のリストを定義します。
除外されたセグメント	このフィールドで許可されていない値の包括的範囲を定義します。 最小除外値：このフィールドに許可されていない最小値。

	最大除外値：このフィールドに許可されていない最大値。
特定の制約 (コンポーネント)	Java API の Constraint インターフェイスを実装する 1 つ以上の Java クラスを指定します。詳細については、プログラムによる制約 を参照してください。
特定の列挙 (コンポーネント)	列挙を定義する Java クラスを指定します。クラスは、Java API の ConstraintEnumeration インターフェイスを実装して、順序付けられた値のリストを定義する必要があります。
別のノードより入力される列挙型	別のリストまたは列挙型エレメントへの参照を使用して、この列挙型の可能な値を定義します。
データスペースセットの構成	タイプデータスペース識別子 (osd : dataspaceKey) のフィールドで参照できるデータスペースを定義します。構成が設定されていない場合、デフォルトでは、開いているブランチのみをこのフィールドで参照できます。 • 包含 このフィールドで参照できるデータスペースを指定します。 パターン：データスペースをフィルタリングするパターンを指定します。パターンは、データスペースの名前と照合されます。 タイプ：このフィールドで参照できるデータスペースのタイプを指定しま

す。定義されていない場合、この制限はブランチに適用されます。

子孫を含める：指定されたパターンに一致するデータスペースの子または子孫をセットに含めるかどうかを指定します。定義されていない場合、この制限は子データスペースには適用されません。[なし] の場合、指定されたパターンに一致するデータスペースの子も子孫も含まれません。[すべての子孫] の場合、指定されたパターンに一致するデータスペースのすべての子孫が含まれます。[すべての子孫ブランチ] の場合、指定されたパターンに一致するデータスペースのすべての子孫ブランチが含まれます。[すべての子孫スナップショット] の場合、指定されたパターンに一致するデータスペースのすべての子孫スナップショットが含まれます。[子ブランチ] の場合、指定されたパターンに一致するデータスペースの直接ブランチのみが含まれます。現在のデータスペースがバージョンの場合、このバージョンの直接の子であるブランチが含まれます。現在のデータスペースがブランチの場合、このブランチの子であるバージョンの直接の子であるブランチが含まれます。[子スナップショット] の場合、指定されたパターンに一致するデータスペースの直接スナップショットのみが含まれます。現在のデータスペースがブランチの場合、このブランチの直接の子であるスナップショットが含まれます。現在のデータスペースがバージョ

ンの場合、このバージョンの子であるブランチの直接の子であるバージョンが含まれます。

- 除外

このフィールドで参照できないデータスペースを指定します。包含が定義されていない場合、除外は無視されます。

パターン：データスペースをフィルタリングするパターンを指定します。パターンは、データスペースの名前と照合されます。

タイプ：このフィールドで参照できるデータスペースのタイプを指定します。定義されていない場合、この制限はブランチに適用されます。

子孫を含める：指定されたパターンに一致するデータスペースの子または子孫をセットに含めるかどうかを指定します。定義されていない場合、この制限は子データスペースには適用されません。[なし] の場合、指定されたパターンに一致するデータスペースの子も子孫も含まれません。[すべての子孫] の場合、指定されたパターンに一致するデータスペースのすべての子孫が含まれます。[すべての子孫ブランチ] の場合、指定されたパターンに一致するデータスペースのすべての子孫ブランチが含まれます。[すべての子孫スナップショット] の場合、指定されたパターンに一致するデータスペースのすべての子孫スナップショットが

含まれます。[子ブランチ] の場合、指定されたパターンに一致するデータスペースの直接ブランチのみが含まれます。現在のデータスペースがバージョンの場合、このバージョンの直接の子であるブランチが含まれます。現在のデータスペースがブランチの場合、このブランチの子であるバージョンの直接の子であるブランチが含まれます。[子スナップショット] の場合、指定されたパターンに一致するデータスペースの直接スナップショットのみが含まれます。現在のデータスペースがブランチの場合、このブランチの直接の子であるスナップショットが含まれます。現在のデータスペースがバージョンの場合、このバージョンの子であるブランチの直接の子であるバージョンが含まれます。

- データスペースフィルター

データセットまたはレコードのコンテンツでデータスペースを受け入れるか拒否するかを指定するフィルターを指定します。このフィルターは、このフィールドに関連付けられている専用の入力コンポーネントでのみ使用されます。つまり、このフィルターは、このフィールドを検証するときには使用されません。特定の制約を使用して、このフィールドで特定の制御を実行できます。フィルターは、Java API の `DataspaceSetFilter` インターフェイスを実装する Java クラスによって定義されます。

データセットセットの構成

データセット識別子 (osd : datasetName) タイプのフィールドで参照できるデータセットを定義します。

- 包含

このフィールドで参照できるデータセットを指定します。

パターン：データセットをフィルタリングするパターンを指定します。パターンは、データセットの名前と照合されます。

子孫を含める：指定されたパターンに一致するデータセットの子または子孫をセットに含めるかどうかを指定します。

- 除外

このフィールドで参照できないデータセットを指定します。包含が定義されていない場合、除外は無視されます。

パターン：データセットをフィルタリングするパターンを指定します。パターンは、データセットの名前と照合されます。

子孫を含める：指定されたパターンに一致するデータセットの子または子孫をセットに含めるかどうかを指定します。

- フィルター

データセットまたはレコードのコンテキストでデータセットを受け入れるか拒否するかを指定するフィルターを指

定します。このフィルターは、このフィールドに関連付けられている専用の入力コンポーネントでのみ使用されます。つまり、このフィルターはこのフィールドを検証するときには使用されません。特定の制約を使用して、このフィールドで特定のコントロールを実行できます。フィルターは、Java API の `DatasetSetFilter` インターフェイスを実装する Java クラスによって定義されます。

検証プロパティ

特定の Java クラスを使用しない各制約は、次のプロパティを使用して、重大度のあるローカライズされた検証メッセージを定義できます。

検証	ユーザー定義の重大度レベルでローカライズされた検証メッセージを定義します。
重大度	検証メッセージの重大度レベルを定義します。可能な値は、[エラー]、[警告]、[情報] です。
エラー管理ポリシー	検証エラーが発生したときの制約の動作を指定します。操作 (データセットの更新、レコードの作成、更新または削除) の後、またはユーザーがフォームを送信したときに、制約が常に有効である必要があることを指定できます。この場合、制約に違反する入力または操作は拒否され、値は変更されません。指定しない場合、制約は、デフォルトですべての操作でエラーが防止されるリレーショナルデータモデルの外部キー制約を除いて、デフォルトでフォーム送信時にエラーをブロックするだけです。このオプションは、静的制御、値の除外、セグメントおよ

	び外部キーの制約の除外でのみ使用できます。外部キー制約では、エラー管理ポリシーはフィルターに関係しません。つまり、参照レコードが存在するが外部キーフィルタを満たさない場合、外部キー制約はブロックされません。この場合、更新は拒否されず、検証エラーが発生します。テーブル履歴またはレプリケーションがアクティブ化されている場合、リレーショナルデータモデルで定義されている構造上の制約にエラー管理ポリシーを指定することはできません。つまり、このプロパティを固定長、最大長、最大桁数、および小数点以下の桁数の制約に設定すると、基になる RDBMS ブロッキング制約検証ポリシーのために、データモデルのコンパイル時にエラーが発生します。アーカイブをインポートする場合、このプロパティは無効です。つまり、アーカイブをインポートするときは、構造上の制約を除くすべてのブロック制約が常に無効になります。
メッセージ	データセット内のこのフィールドの値がこの制約に準拠していない場合に表示するメッセージを定義します。カスタムメッセージを指定する場合は、オプションでローカライズされたバリエーションを提供できます。

関連する概念:

- データモデルエレメントのプロパティ

ユーザーガイド [目次](#)

データモデルの拡張機能

- データモデルで使用する拡張機能

- インデックス作成と検索戦略
- ビジネスオブジェクト
 - ビジネスオブジェクトモデルの構造の定義
- ツールバー
 - ツールバーの構造の定義
 - エレメントの削除
 - 既存のエレメントの複製
 - エレメントの移動
 - 既存のテーブルへの関連付け
 - ツールバーのエクスポート
 - ツールバーのインポート
- エンティティマッピング
- リレーショナルテーブルへのデータのレプリケーション
- ノードの機密性

データモデルで使用される拡張機能

任意のデータモデルに対して、使用する拡張機能を指定できます。これらの拡張機能には、データモデルの周囲に追加機能を定義し、データモデルエレメントにプロパティと制約を追加することにより、公開後に現在のデータモデルを強化する機能があります。データモデルの拡張機能は、ナビゲーションペインの セクションに表示されます。

次の拡張機能は、新しいデータモデルで自動的に有効になります。

- ツールバー
- エンティティマッピング
- レプリケーション
- インデックス作成と検索戦略
- ビジネスオブジェクト
- カスタムフォーム
- スクリプト
- レコードの権限
- ノードの機密性

一部の拡張機能はオプションであり、専用のデータセットで定義されています。これらの拡張機能は、ユーザーインターフェイスを介して無効または有効にできます。

オプションのデータモデル拡張機能は無効にするには、拡張機能の左側にあるメニュー  からアクションを選択します。

注意:

拡張機能が無効にすると、その専用データセットも削除され、削除後に復元することはできません。

ナビゲーションペインの エントリの左側にあるメニュー  から、アクションを選択して拡張機能を有効にできます。このオプションは、いくつかの拡張機能が利用可能な場合にのみ表示されます。このアクションを選択すると、使用可能なデータモデル拡張機能がフォームに表示されます。

データモデル拡張機能がアクティブ化されると、ナビゲーションペインの エントリの下に表示され、通常のデータセットとして編集できます。つまり、拡張機能の構成は、データセットで利用可能なすべての EBX® 機能の利点を活用する専用のデータセットに埋め込まれています。

インデックス作成と検索戦略

この機能は、[検索](#)の章に記載されています。

ビジネスオブジェクト

ビジネスオブジェクトは、同じデータモデル内の複数のテーブルのレコードを操作できる抽象レイヤーです。ビジネスオブジェクトモデルと呼ばれるビジネスオブジェクトの定義は、セクションからデータモデルで実行できます。ビジネスオブジェクトモデルはメインテーブルで定義され、このメインテーブルの関係を使用して定義された子エンティティが含まれます。

ナビゲーションペインの > セクションから、 の左側にあるメニュー  をクリックし、オプションを選択して、ビジネスオブジェクトモデルを作成します。作成ウィザードに従って、ビジネスオブジェクトモデルを作成します。ビジネスオブジェクトモデルは、次の情報を定義します。

ビジネスオブジェクト名

ビジネスオブジェクトモデルの名前。名前は、ビジネスオブジェクトモデルを一意に識別するために使用されるため、データモデルのコンテキスト内で一意にする必要が

	あります。つまり、同じ名前で複数のビジネスオブジェクトを作成することはできません。
ラベルおよび説明	エンドユーザーに表示される国際化されたラベルと説明。
メインテーブル	このビジネスオブジェクトモデルのメインテーブルを定義します。指定するパスは絶対パス (「/」で始まる) にする必要があります。このビジネスオブジェクトモデルのメインテーブルは、すでに別のビジネスオブジェクトモデルの一部である場合は使用できません。つまり、メインテーブルがすでにメインテーブルとして、または別のビジネスオブジェクトモデルの子エンティティとして定義されている場合、メインテーブルは使用できません。

ビジネスオブジェクトモデルの構造の定義

ビジネスオブジェクトモデルは、メインテーブルと子エンティティで構成されます。子エンティティは、メインテーブルに含まれる関係を使用して定義されます。

ビジネスオブジェクトモデルの下に子エンティティを追加するには、既存のビジネスオブジェクトの左側にあるメニュー  をクリックし、オプションを選択します。次に、作成ウィザードに従って子エンティティを作成します。次の情報を設定する必要があります。

テーブルとの関係	このフィールドは、子エンティティとして追加するテーブルとの関係を定義します。関係へのパスは絶対パス (「/」で始まる) にする必要があります。逆外部キーによる外部キー制約と関連付けのみがサポートされています。また、これらの関係は、次の制約に準拠する必要があります。外部キー制約は、リストまたはリストの下にすることはできません。外部キー制約を定義するフィールドは一意として宣言する必要があります。つまり、ブロック検証モードで一意性制約を定義
----------	---

するか、コンテナテーブルの唯一の主キーフィールドにする必要があります。逆外部キーによる関連付けは、リストまたはリストの下にある外部キーフィールドを参照する必要があります。また、関係のターゲットテーブルは、このビジネスオブジェクトモデルまたは別のビジネスオブジェクトモデルによってすでに使用されているテーブルを参照できません。つまり、ターゲットテーブルがすでにメインテーブルまたはビジネスオブジェクトモデルの子エンティティとして定義されている場合、そのテーブルは使用できません。

ツールバー

ツールバーを使用すると、データセット内のテーブルまたはレコードを表示するときに表示されるボタンとメニューをカスタマイズできます。ツールバーのカスタマイズは、セクションのデータモデルで実行できます。

の左側にあるメニュー  をクリックして、ナビゲーションペインのセクションからツールバーを追加します。オプションを選択します。作成ウィザードに従ってツールバーを作成します。ツールバーでは次の情報が定義されます。

名称	ツールバーの名前。ツールバーの名前は、データモデルのコンテキストで一意にする必要があります。つまり、同じ名前で複数のツールバーを作成することは許可されていません。
ラベルおよび説明	エンドユーザーに表示される国際化されたラベルと説明。
デフォルトテンプレート	デフォルトのツールバーの構造でツールバーを作成できます。
位置	関連するデータセットでツールバーを使用できる場所を指定します。

表形式ビューの上部：デフォルトテーブルビューの上部で使用するツールバーを定義します。

表形式ビューの行：デフォルトテーブルビューの各行で使用するツールバーを定義します。

タイルビュー項目：タイルビューの各項目で使用するツールバーを定義します。

レコード上部：レコードフォームで使用するツールバーを定義します。

階層上部：テーブルのデフォルト階層ビューで使用するツールバーを定義します。

既知の制限事項

[タイルビュー項目] として使用するよう定義されたツールバーは、6 つを超える項目を持つことはできず、アクション項目のみを定義できます。これらのアクションボタンの表示は、常に [アイコンのみ] モードです。

ツールバーの構造の定義

ツールバーは、次のエレメントを定義できます。

- アクションボタン
- メニューボタン
- 区切り文字
- メニューグループ
- アクションメニュー項目
- サブメニュー項目

これらのエレメントのいずれかをツールバーの下に追加するか、既存のエレメントの左側にあるメニュー  をクリックし、メニューで作成オプションを選択して既存のエレメントに追加します。次に、作成ウィザードに従ってエレメントを作成します。

アクションボタン

このタイプのエレメントを使用すると、アクションをツールバーのボタンに関連付けることができます。ユーザーがツールバーの 1 つにある関連するボタンをクリックすると、アクションが実行されます。タイプのエレメントは、次の情報を定義します。

サービス	ユーザーがボタンをクリックしたときに実行されるサービスを定義します。ビルトインサービス、またはモジュールまたは現在のデータモデルで定義されているユーザーサービスを選択できます。[Web コンポーネント] ターゲットが選択されている場合、サービスはツールバーの Web コンポーネントとして利用可能であると宣言する必要があります。
ラベルおよび説明	エンドユーザーに表示される国際化されたラベルと説明。
レイアウト	ツールバーを使用して、このエレメントがデータセットにどのように表示されるかを定義します。アイコンのみ、テキストのみ、左側にアイコンが付いたテキスト、または右側にアイコンが付いたテキストを表示できます。
アイコン	表示するアイコン。アイコンを使用して、提案されたアイコンのセットから選択したり、URL を使用してアイコンを参照したりすることができます。
リリーフ	このボタンの表示方法を定義します。ボタンはエンボスまたはフラットとして表示できます。
強調表示	ボタンをデフォルトで強調表示する必要があるかどうかを示します。

注意:

タイプのエレメントは、タイプのエレメントの下でのみ作成できます。

メニューボタン

このタイプのエレメントを使用すると、ユーザーがツールバーの関連するボタンをクリックしたときに表示されるメニューを定義できます。タイプのエレメントは、次の情報を定義します。

ラベルおよび説明	エンドユーザーに表示される国際化されたラベルと説明。
レイアウト	ツールバーを使用して、このエレメントがデータセットにどのように表示されるかを定義します。アイコンのみ、テキストのみ、左側にアイコンが付いたテキスト、または右側にアイコンが付いたテキストを表示できます。
アイコン	表示するアイコン。アイコンを使用して、提案されたアイコンのセットから選択したり、URL を使用してアイコンを参照したりすることができます。
リリーフ	このボタンの表示方法を定義します。ボタンはエンボスまたはフラットとして表示できます。
強調表示	ボタンをデフォルトで強調表示する必要があるかどうかを示します。

注意:

タイプのエレメントは、タイプのエレメントの下でのみ作成できます。

区切り文字

このタイプのエレメントを使用すると、ツールバーの2つのエレメントの間に間隔の形でセパレーターを挿入できます。

注意:

タイプのエレメントは、タイプのエレメントの下でのみ作成できます。

メニューグループ

このタイプのエレメントを使用すると、メニューでエレメントのグループを定義できます。タイプのエレメントは、次の情報を定義します。

ラベルおよび説明	エンドユーザーに表示される国際化されたラベルと説明。
グループタイプ	作成するメニューグループのタイプを指定します。 - [ローカル] を使用すると、完全にカスタマイズ可能な空のメニューグループを作成できます。 - [サービスグループ] を使用すると、既存のサービスグループをこのメニューグループに割り当てることができます。 - [メニュービルダー] を使用すると、事前定義されたメニューコンテンツをこのメニューグループに割り当てることができます。一度作成すると、このメニューグループのタイプを変更することはできません。
サービスグループ名	再利用する既存のサービスグループを指定します。グループはモジュールで宣言され、他のサービスグループを含めることができます。このグループに含まれるすべてのサービスは、関連するデータセットのエンドユーザーに表示されます。
メニュービルダー名	このメニューグループに割り当てる定義済みのメニューコンテンツを指定します。 - [デ

	フォルトメニュー [アクション]] には、デフォルトツールバーの [アクション] メニューと同じコンテンツがあります。標準サービスとカスタムサービスは区別なく表示されます。 - [デフォルトメニュー [アクション] (区切り文字付き)] のメニューコンテンツは上記と同じですが、標準サービスとカスタムサービスが分離されているため、表示が異なります (最初に標準サービス、次にカスタムサービス)。
除外されるサービス	再利用されたサービスのグループから除外するサービスを示します。これらのサービスは、関連するデータセットのエンドユーザーには表示されません。
除外されるサービスグループ	再利用するサービスのグループから除外するグループを示します。除外されたグループのサービスは、関連するデータセットのエンドユーザーには表示されません。
フィルタリングポリシー	[スマートフィルタリング] の場合、直接アクセス、つまりアクションボタンまたはアクションメニュー項目を介して構成されたサービスは、このグループの自動生成から削除されます。

注意:

タイプのエレメントは、次のエレメントの下でのみ作成できます。

- メニューボタン
- サブメニュー項目

アクションメニュー項目

このタイプのエレメントを使用すると、アクションをツールバーのメニュー項目に関連付けることができます。ユーザーがメニュー内の対応する項目をクリックすると、アクションが実行されます。タイプのエレメントは、次の情報を定義します。

ラベルおよび説明	エンドユーザーに表示される国際化されたラベルと説明。
サービス	ユーザーがボタンをクリックしたときに実行されるサービスを定義します。ビルトインサービス、またはモジュールまたは現在のデータモデルで定義されているユーザーサービスを選択できます。[Web コンポーネント] ターゲットが選択されている場合、サービスはツールバーの Web コンポーネントとして利用可能であると宣言する必要があります。

注意:

タイプのエレメントは、タイプのエレメントの下でのみ作成できます。

サブメニュー項目

このタイプのエレメントを使用すると、ツールバーメニューにサブメニューを追加できます。は、次の情報を定義します。

ラベルおよび説明	エンドユーザーに表示される国際化されたラベルと説明。
----------	----------------------------

注意:

タイプのエレメントは、タイプのエレメントの下でのみ作成できます。

エレメントの削除

ツールバーのエレメントはすべて、削除するエレメントの左側にあるメニュー  を使用して削除できます。

他のエレメントを含むエレメントが削除されると、削除されたエレメントの下にあるすべてのエレメントに対して再帰的に削除が実行されます。

既存のエレメントの複製

エレメントを複製するには、複製するエレメントの左側にあるメニュー  をクリックします。複製したエレメントの名前とプロパティを指定します。すべてのソースエレメントのプロパティが複製されます。

複製されたエレメントは、元のエレメントと同じレベルの最終位置に追加されます。他のエレメントを含むエレメントが複製されると、すべてのサブエレメントがそれらのプロパティとともに複製されます。

エレメントの移動

エレメントを移動するには、メニュー  をクリックし、使用する移動オプションを選択します。

既存のテーブルへの関連付け

ツールバーを既存のテーブルに関連付けるには、ツールバーの左側にあるメニュー  をクリックし、を選択します。このサービスを使用すると、ツールバーに複数のテーブルのデフォルトのツールバーを一度に設定できます。これを行うには、ツールバーのターゲットの場所を指定し、ツールバーに関連付けるテーブルまたはテーブルのプロパティを定義する複合データ型を選択します。

ツールバーのエクスポート

モデルで定義されたツールバーを XML ドキュメントにエクスポートすることができます。これを行うには、セクションのメニューにあるオプションを選択します。ウィザードに従って、ツールバーをエクスポートします。

注意:

選択したツールバーは、セクションでエクスポートするツールバーを選択し、メニューで利用できるオプションを選択することでエクスポートできます。ツールバーは、データモデルエクスポートサービスを使用してエクスポートすることもできます。これは、ナビゲーションペインの  メニューにあります。

以下も参照してください。:

- [データモデルのインポートとエクスポート](#)

ツールバーのインポート

XML ドキュメントから既存のツールバーをインポートすることができます。これを行うには、セクションのメニューにあるオプションを選択します。次に、ウィザードに従ってツールバーをインポートします。

注意:

ツールバーは、ナビゲーションペインの [🔗](#) メニューからアクセスできるデータモデルインポートサービスを使用してインポートすることもできます。

以下も参照してください。:

- [データモデルのインポートとエクスポート](#)

以下も参照してください。:

- [ツールバーの使用](#)

エンティティマッピング

パスの代わりに一意の名前を使用して、SQL リクエストおよびデータサービス操作でテーブルを参照できます。これを行うには、エンティティマッピングを定義して、テーブルパスと名前の関連付けを作成する必要があります。テーブルを参照するときは、この名前をデータサービス操作の WSDL サフィックスとして使用し、SQL リクエストのエイリアスとして使用します。

注意:

EBX® は、データモデルで定義されたテーブルのエンティティ名を自動的に作成します。テーブルパスの最後のステップは、デフォルトのエンティティ名として使用されます。エンティティ名はデータモデル内で一意にする必要があるため、モデルに同じ名前のテーブルが含まれていると、コンパイル中にエラーが発生するようになりました。この場合のエラーを防ぐには、エンティティ名を明示的に設定する必要があります。

ユーザーインターフェイスを介してエンティティマッピングを定義するには、データモデルを表示しているときにナビゲーションペインで > に移動し、新しいレコードを作成します。次のプロパティを定義できます。

テーブルパス	SQL リクエストまたは WSDL 操作でこの名前によって参照される、現在のデータモデル内のテーブルのパスを指定します。
エンティティ名	この名前は、関連するテーブルのすべての WSDL 操作名のサフィックスとして、また SQL リクエストのエイリアスとして使用されます。特定のテーブルで定義されていない場合は、代わりにテーブルパスの最後のエレメントが使用されます。この名前は、このデータモデルのコンテキスト内で一意にする必要があります。

以下も参照してください。:

- データサービス

リレーショナルテーブルへのデータのレプリケーション

どのデータモデルでも、リポジトリ内のデータをリレーショナルデータベース内の専用テーブルにミラーリングするためのレプリケーションユニットを定義できます。これらのテーブルにより、SQL リクエストとビューによるデータへの直接アクセスが可能になります。

ユーザーインターフェイスを介してレプリケーションユニットを定義するには、ナビゲーションページの セクションの テーブルに新しいレコードを作成します。各レプリケーションユニットレコードは、特定のデータスペース内の特定のデータセットに関連付けられています。同じデータセット内にある限り、単一のレプリケーションユニットで複数のテーブルをカバーできます。レプリケーションユニットは、次の情報を定義します。

名称	レプリケーションユニットの名前。この名前は、現在のデータモデルのレプリケーションユニットを識別します。一意にする必要があります。
データスペース	このレプリケーションユニットに関連するデータスペースを指定します。スナップショットまたはリレーショナルデータスペースにすることはできません。
データセット	このレプリケーションユニットに関連するデータセットを指定します。
更新ポリシー	データ同期ポリシーを指定します。可能なポリシーは次のとおりです。 • コミット時：データベース内のレプリケートされたテーブルのコンテンツは、ソーステーブルに関して常に最新です。EBX® ソーステーブルを更新するすべてのトランザクションは、レプリケートされたテーブルで対応する挿入、更新、および削除ステートメントをトリガーします。 • オンデマンド：データベース内のレプリケートされたテーブルは、明示的な Update 操作が実行された場合にのみ更新されます。
テーブル	データベースに複製されるデータモデルのテーブルを指定します。

	テーブルパス：データベースにレプリケートされる現在のデータモデル内のテーブルのパスを指定します。 データベース内のテーブル名：データの複製先となるデータベース内のテーブルの名前を指定します。この名前は、すべてのレプリケーションユニット間で一意にする必要があります。
集約リスト	データベースに複製されるテーブル内の集約リストのプロパティを指定します。 パス：データベースに複製されるテーブル内の集約リストのパスを指定します。 データベース内のテーブル名：集約リストのデータが複製されるデータベース内のテーブルの名前を指定します。この名前は、すべてのレプリケーションユニット間で一意にする必要があります。

以下も参照してください。：

- レプリケーション

ノードの機密性

ノードの機密性は、特定のユーザーに対して非表示の場合に、クエリまたはリクエストでノードを使用できるかどうかを定義します。デフォルトでは、すべてのノードが機密です。

これは、ノードフォームエディターの タブを使用して、DMA で構成できます。次のように、いくつかのレベルで機密性を定義することが可能です。

- データモデルレベル (ルートノード)：データモデルターミナルノードのデフォルトの機密性を定義します。ただし、より具体的な定義が (テーブルまたはノード自体に) 設定されている場合を除きます。

- テーブルレベル (テーブルノード)：より具体的な定義が (ノード自体に) 設定されている場合を除き、テーブルターミナルノードのデフォルトの機密性を定義します。
- ターミナルノードレベル：ノードの機密性を定義します。

詳細については、[機密性の定義](#)を参照してください。

[ユーザーガイド 目次](#)

既存のデータモデルの操作

データモデルが作成されると、ワークスペースの **[データモデルの]** メニューから利用できるいくつかのアクションを実行できます。

- [データモデルの検証](#)
- [データモデルのインポートとエクスポート](#)
- [データモデルの複製](#)
- [データモデルの削除](#)

データモデルの検証

どの時点でも、データモデルを検証するには、ナビゲーションペインから **>** を選択します。生成されたレポートは、検証の結果を提供します。検証レポートから、ボタンをクリックして報告された検証ステータスを更新するか、ボタンをクリックしてデータモデルに関連付けられている現在の検証レポートをクリアし、最初から完全な検証を再実行できるようにするオプションがあります。

注意:

検証プロセスは基本的なデータモデルの整合性をチェックしますが、より複雑なチェックは公開時にのみ実行されます。データモデルを公開しようとする、さらに多くのメッセージが報告される場合があります。

増分データ検証の詳細については、[検証](#) を参照してください。

データモデルのインポートとエクスポート

TIBCO EBX® には、XML スキーマドキュメント (XSD) ファイルまたはアーカイブファイル (zip) との間でインポートおよびエクスポートするためのビルトインのデータモデルサービスが含まれています。インポートとエクスポートは、ナビゲーションペインのターゲットデータモデルの **データモデルの** メニューから実行できます。インポートまたはエクスポートは、常に単一のデー

タモデルのコンテキストで実行されます。つまり、インポート中に、ターゲットデータモデルの構造は、インポートされたデータモデルのコンテンツ (XSD またはアーカイブ) に完全に置き換えられます。同様に、エクスポート中、データモデル全体が XSD またはアーカイブファイルに含まれます。

データモデルをインポートする場合、対応する XSD ファイルは整形形式であり、EBX® 検証規則に準拠している必要があります。

このドキュメントでモジュール内にあるリソースを宣言する場合、そのモジュールはデータモデルの構成でも宣言する必要があります。モジュールが宣言されていない場合、データモデルをインポートすることはできません。モジュールの宣言の詳細については、[データモデルのプロパティ](#)を参照してください。

インポートを実行するには、インポート先のデータモデルの **[データモデルの]** メニューから 選択します。

ローカルファイルシステムからXMLスキーマドキュメント (XSD) またはアーカイブ (zipファイル) をインポートできます。これを行うには、インポートウィザードの ボタンをクリックして、次の手順に従います。

- **ドキュメント名：**インポートする XSD またはアーカイブファイルのローカルファイルシステム上のパス。

注意:

アーカイブをインポートするときは、ルートに XML スキーマドキュメント (XSD) が1つだけ含まれている必要があります。データモデル拡張に関連する XML ドキュメントも、アーカイブのルートに配置する必要があります。これらの XML ドキュメントは、EBX® でサポートおよび登録されている拡張機能の名前に準拠している場合、自動的にインポートされます。

モジュールにパッケージ化された XSD のデータモデルをインポートすることもできます。モジュールから XSD 形式のデータモデルをインポートする場合は、次のプロパティを使用します。

モジュール	データモデルがパッケージ化されるモジュール。
モジュールパス	データモデルを含むモジュールへのパス。

ソースパス	ビジネスオブジェクトとルールを構成するために使用される Java ソースへのパス。 インポートされるデータモデルでプログラムエレメントが定義されている場合は、このプロパティが必要です。
モデル	インポートするモジュール内のデータモデル。

注意:

インポートされた XSD ファイルは「UTF-8」でエンコードする必要があります。エクスポートされた XSD ファイルは常に「UTF-8」でエンコードされます。

エクスポートを実行するには、エクスポートするデータモデルの **[データモデルの]** メニューから **を** 選択します。

データモデルの複製

データモデルを複製するには、そのデータモデルの **[データモデルの]** メニューから **を** 選択します。新しいデータモデルには、リポジトリ内で一意の名前を付ける必要があります。

データモデルの削除

データモデルを削除するには、そのデータモデルの **[データモデルの]** メニューから **を** 選択します。データモデルを削除すると、既存のすべてのパブリケーションが残り、関連するデータセットに引き続きアクセスできます。以前に削除されたものと同じ名前で新しいデータモデルを再作成すると、新しいデータモデルは、リポジトリ内の既存のすべてのパブリケーションに再関連付けされます。新しいデータモデルの公開時に、既存の公開の置き換えを確認する機会があります。

注意:

領域で削除されたデータモデルのパブリケーションをクリーンアップできるのは、管理者のみです。

公開プロセスの詳細については、**データモデルの公開** を参照してください。

データモデルの公開とバージョン管理

- データモデルの公開
- 埋め込みデータモデルのバージョン管理

データモデルの公開

- パブリケーションについて
- パブリケーションモード
- 埋め込みパブリケーションモード
 - 公開の表示と作成

パブリケーションについて

TIBCO EBX® リポジトリの埋め込みデータモデルに基づく各データセットは、データモデル自体に直接ではなく、データモデルの公開に関連付けられています。ナビゲーションペインの ボタンを使用してデータモデルを初めて公開すると、公開が作成されます。パブリケーションが存在すると、それに基づいてデータセットを作成できます。

注意:

ボタンは、データモデルを公開する権限を持つユーザーにのみ表示されます。詳細については、[データモデルの権限](#) を参照してください。

データセットはパブリケーションに基づいているため、データモデルに加えた変更は、それらのデータセットに関連付けられたパブリケーションに再公開した場合にのみ、既存のデータセットに有効になります。データモデルを既存のパブリケーションに再公開すると、その特定のパブリケーションに関連付けられているすべての既存のデータセットが更新されます。

パブリケーションモード

「埋め込み」モードまたは「モジュール内」モードのいずれかを使用して、データモデルを公開できます。「埋め込み」パブリケーションモードは、EBX® リポジトリ内で管理および永続化されるパブリケーションを生成するため、バージョン管理およびロールバック機能が関連付けられ

ています。「モジュール内」パブリケーションモードは、リポジトリ内で管理またはバージョン管理されていないモジュールに含まれる XML スキーマドキュメントを作成します。

データモデルの構成に応じて、EBX® は、ナビゲーションペインの ボタンをクリックすると、使用するパブリケーションプロセスを自動的に決定します。データモデルがパブリケーションモード「モジュール内」を指定し、生成されるターゲット XSD を提供する場合、パブリケーションプロセスは、構成で定義されたモジュールに含まれる XSD ファイルを生成します。

埋め込みパブリケーションモード

特定の埋め込みデータモデルを初めて公開すると、データモデルと同じ名前の新しい公開がリポジトリに自動的に作成されます。このモデルに対して複数のパブリケーションがすでに作成されている場合は、このプロセスのターゲットパブリケーションを選択する必要があります。

さまざまな公開の使用の詳細については、[公開の表示と作成](#) を参照してください。

データモデルが以前に公開されている場合は、パブリケーションプロセス中に、現在の公開によって導入されている構造上の違いを並べて比較ビューで確認する機会があります。

パブリケーションプロセスには、参照用にデータモデルの現在の状態の読み取り専用スナップショットを作成するオプションもあります。このスナップショットは、他の変更が行われた後にデータモデルを現在の状態にロールバックする必要がある場合に役立ちます。

注意:

データモデルの状態の静的アーカイブであるスナップショットを、データモデルの並列進化ブランチとして機能するデータモデル バージョンと混同しないでください。データモデルのバージョンの詳細については、[埋め込みデータモデルのバージョン管理](#) を参照してください。

公開の表示と作成

現在のデータモデルに存在するパブリケーションにアクセスするには、ナビゲーションペインの **[データモデルの]** メニューから を選択します。そこから、パブリケーションの詳細を表示し、新しい公開を作成できます。

場合によっては、データセットがそのデータモデルの異なる状態に基づくことを可能にするために、同じデータモデルの複数のパブリケーションを採用する必要があるかもしれません。複数のパブリケーションが存在する場合、ユーザーはパブリッシング時にターゲットとする利用可能なパブリケーションを選択するように求められるため、複数のパブリケーションは慎重に処理する

必要があります。新しいパブリケーションを作成するアクションは、「管理者」のロールに属するユーザーのみが使用できます。

新しいパブリケーションを作成するには、ナビゲーションペインのデータモデルの **[データモデルの]** メニューから **を選択し、** ボタンをクリックします。パブリケーションに付ける名前は、リポジトリ内で一意にする必要があります。

ユーザーガイド [目次](#)

埋め込みデータモデルのバージョン管理

- [バージョンについて](#)
- [バージョンへのアクセス](#)
- [バージョンの操作](#)
- [データモデルのバージョン管理に関する既知の制限](#)

バージョンについて

並行して進化するデータモデルの **バージョン** を作成できます。バージョンをデータモデルのスナップショットと混同しないでください。データモデルのスナップショットは、公開時に取得され、履歴の読み取り専用参照のために厳密に保持されます。

バージョンへのアクセス

データモデルの既存のバージョンを表示するには、データモデルの **[データモデルの]** メニューから **を選択します。**

既存のバージョンは、親子関係に従ってツリー形式で表されます。すべてのデータモデルには、デフォルトで、データモデルと同じ名前のルートバージョンがあります。

バージョンの操作

ワークスペースで、各バージョンの横にある下矢印  メニューを使用して、次のアクションを実行できます。

アクセスデータモデルバージョン	対応するバージョンのデータモデルに移動します。
バージョンの作成	選択したバージョンの内容に基づいて、新しいバージョンを作成します。新しいバージョンは、選択したバージョンの子として追加さ

	れますが、その内容は、作成後の親バージョンの内容とは関係ありません。
デフォルトバージョンとして設定	選択したバージョンを、ユーザーがデータモデルにアクセスしたときに開くデフォルトバージョンとして設定します。
アーカイブのエクスポート	選択したデータモデルバージョンを、そのアクセス許可と情報を含むバージョンのコンテンツを含むアーカイブにエクスポートします。エクスポートされたアーカイブは、リポジトリ管理者がアクセスできるアーカイブディレクトリにあります。既存のアーカイブ名にエクスポートすると、既存のファイルが上書きされます。 詳細については、アーカイブディレクトリを参照してください。
アーカイブのインポート	アーカイブのコンテンツを選択したバージョンにインポートします。インポートするアーカイブには、バージョンに関連付けられているデータモデルと同じ名前のデータモデルが含まれている必要があります。

バージョンは、エントリの右側にある ボタンをクリックして削除できます。パブリケーションにリンクされているバージョン、または子バージョンがあるバージョンは削除できません。データモデルのルートバージョンも削除できません。

同じデータモデルの 2 つのバージョンをワークスペースで比較するには、チェックボックスを選択してから、> を選択します。並べて比較すると、データモデルのバージョン間の構造の違いが示され、左側に古いバージョン、右側に新しいバージョンがあります。

データモデルのバージョン管理に関する既知の制限

- 2 つのバージョンのデータモデルをマージすることはできません。
- 比較インターフェイスは、フィールドの更新を表示せず、追加と削除のみを表示します。
- モジュールにパッケージ化されたデータモデルのバージョン管理はサポートされていません。

- 埋め込みデータモデルで使用されるモジュールにパッケージ化されたリソースは、バージョンの作成時にバージョン管理されません。つまり、バージョンの作成中にリソースの参照のみが保存され、参照されるリソースのコンテンツがそれらを使用している可能性のあるすべてのバージョンと互換性があることを確認するのは開発者の責任です。

ユーザーガイド [目次](#)

データスペース

- [データスペースの概要](#)
- [データスペースの作成](#)
- [既存のデータスペースの操作](#)
- [スナップショット](#)

データスペースの概要

- [概要](#)
 - [データスペースとは](#)
 - [データスペースに関連する基本概念](#)
- [データスペース領域のユーザーインターフェイスの使用](#)

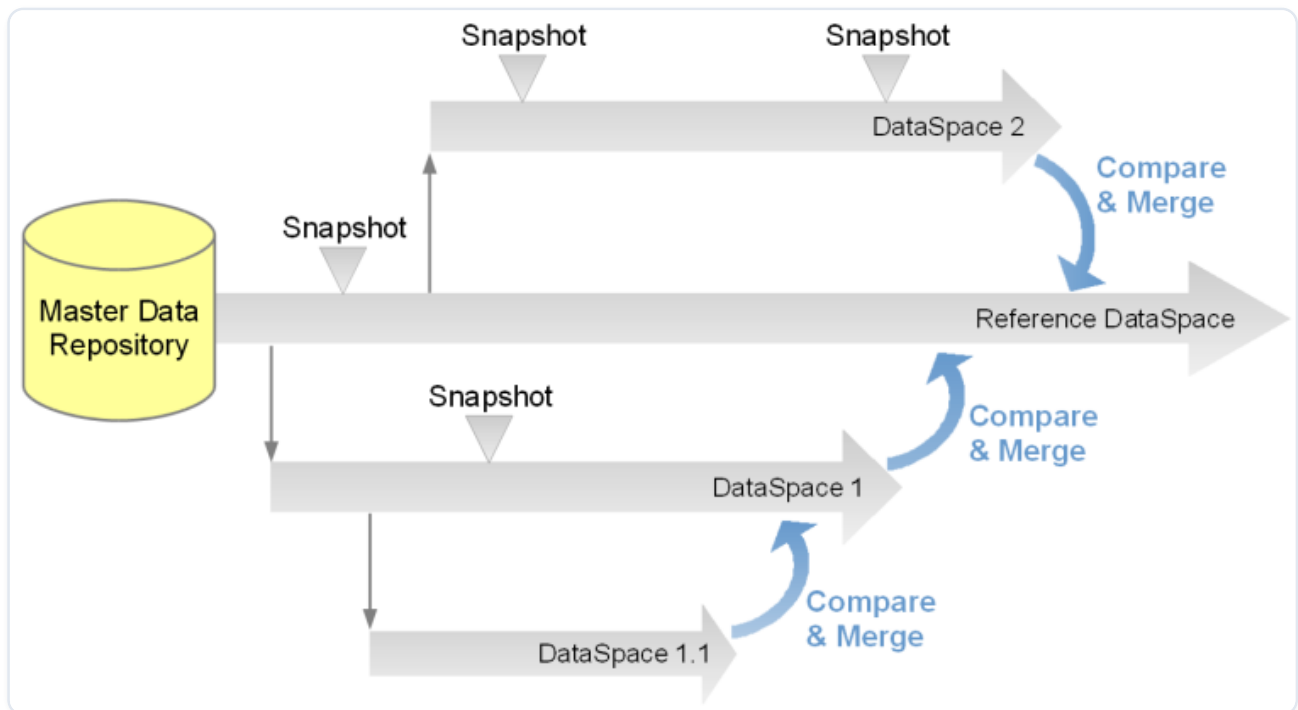
概要

データスペースとは

データのライフサイクルは複雑になる可能性があります。途中でさまざまな状態のトレースを保持するなど、将来統合されるいくつかの同時更新に取り組んでいる間、データの現在のバージョンを管理する必要がある場合があります。TIBCO EBX® では、これはデータスペースとスナップショットを使用することで可能です。

データスペースは、さまざまなバージョンのデータセットを分離して整理するコンテナです。データスペースは、親の状態で自動的に初期化される子データスペースを作成することで分岐できます。したがって、変更は、親または他のデータスペースに影響を与えることなく、子データスペースで分離して行うことができます。子データスペースでの変更が完了すると、そのデータスペースを親データスペースと比較してマージして戻すことができます。

スナップショットは、特定の時点でのデータスペースの状態の静的な読み取り専用キャプチャであり、参照目的で取得できます。必要に応じて、スナップショットを使用して、後でデータスペースのコンテンツを元に戻すことができます。



データスペースに関連する基本概念

次の用語の基本的な理解は、データスペースを操作するときに役立ちます。

- データスペース
- スナップショット
- データセット
- データスペースのマージ
- 参照データスペース

データスペース領域のユーザーインターフェイスの使用

データスペースは、領域で作成、アクセス、および変更できます。

注意:

この領域は、で許可されたユーザーのみが使用できます。

ナビゲーションペインには既存のすべてのデータスペースが表示され、ワークスペースには選択したデータスペースに関する情報とそのスナップショットが一覧表示されます。

The screenshot shows the TIBCO EBX user interface. On the left is a navigation pane with a tree structure: 'Dataspace' is expanded, showing 'Master Data - Reference' (which includes 'Samples' and 'D-S 593'), 'Workflow', 'Dynamic data hierarchy', and 'Dynamic data hierarchy - re'. 'D-S 593' is selected and highlighted in blue. The main area on the right displays the details for 'D-S 593'. At the top of this area is a header with 'D-S 593' and a help icon. Below the header are two tabs: 'Create a dataspace' and 'Actions'. The 'Actions' tab is active, showing a list of properties and their values:

Identifier	Tutorial
Type	Dataspace
Creation	on 07/10/2013 at 12:32:34 by Doc. Adm
Status	Open
History purge	Marked for history purge by Doc. Admini
Owner	Doc. Administrator (admin)
Loading strategy	On-demand loading and unloading
Child merge policy	Allows validation errors in result
Child dataspace sort policy	By creation date (asc)

以下も参照してください。:

- [データスペースの作成](#)
- [スナップショット](#)

関連する概念:

- [データセット](#)

[ユーザーガイド 目次](#)

データスペースの作成

- [概要](#)

- プロパティ

概要

新しいデータスペースを作成するには、その基準となる既存のデータスペースを選択し、ワークスペースの ボタンをクリックします。

注意:

この領域は、 で許可されたユーザーのみが使用できます。

新しいデータスペースは、作成元のデータスペースの子データスペースになります。作成時に親のすべてのコンテンツで初期化され、この状態の初期スナップショットが取得されます。

リポジトリ内のすべてのデータスペースのルートである参照データスペースを除いて、データスペースは常に別のデータスペースの子です。

プロパティ

新しいデータスペースを作成するには、次の情報が必要です。

識別子	データスペースの一意の識別子。
所有者	データスペースの所有者。デフォルトでは、その情報と権限を変更できます。所有者は必ずしもデータスペースの作成者である必要はありません。
ラベル	データスペースに関連付けられたローカライズされたラベルと説明。

[ユーザーガイド 目次](#)

既存のデータスペースの操作

- データスペース情報
- データスペースの権限

- 一般的な権限
- 許容されるアクション
- データスペースのマージ
 - マージの開始
 - 変更の確認と承認
 - マージの完了
 - マージの中止
- データスペースの比較
- データスペースの検証
- データスペースアーカイブ
 - エクスポート
 - インポート
- データスペースを閉じる

データスペース情報

データスペースに関連付けられている特定のプロパティは、領域のナビゲーションパネルから > を選択することで変更できます。

ドキュメント	データスペースに関連付けられたローカライズされたラベルと説明。
子のマージポリシー	このマージポリシーは、ユーザーが開始するマージプロセスにのみ適用されます。ワークフロースクリプトタスクによって実行されるようなプログラムによるマージには適用されません。 使用可能なマージポリシーは次のとおりです。 • 結果の検証エラーを許可する：検証結果に関係なく、子データスペースをマージ

	できます。これがデフォルトのポリシーです。 • マージの事前検証：子データスペースは、結果が有効である場合にのみマージできます。
現在の所有者	データスペースの所有者。デフォルトでは、その情報と権限を変更できます。所有者は必ずしもデータスペースの作成者である必要はありません。
子データスペースの並べ替えポリシー	データスペースツリー内の子データスペースの表示順序を定義します。定義されていない場合、親データスペースのポリシーが適用されます。デフォルトは 'ラベル' です。
所有者を変更	データスペースの現在の所有者が、その「現在の所有者」プロパティを変更することにより、別のプロファイルに所有権を与えることが許可されているかどうか。値が「禁止」の場合、管理者のみが所有者を変更できます。
権限の変更	データスペースの現在の所有者がそのアクセス許可を変更できるかどうか。値が「禁止」の場合、管理者のみがデータスペースの権限を変更できます。

データスペースの権限

一般的な権限

データスペース ID	権限が適用されるデータスペース。
プロファイルの選択	ルールが適用されるプロファイル。
制限ポリシー	これらのアクセス許可が、他のプロファイルに定義されたポリシーを通じて特定のユー

	ザーに割り当てられたアクセス許可を制限するかどうか。 制限ポリシー を参照してください。
データスペースへのアクセス	データスペースに対するグローバルアクセス許可。 Read-only • 権限に応じて、データスペースとそのスナップショット、および子データスペースを表示できます。 • 権限に応じてデータスペースの内容を表示できます。変更することはできません。 Write • 権限に応じて、データスペースとそのスナップショット、および子データスペースを表示できます。 • 権限に応じて、データスペースの内容を変更できます。 Hidden • データスペースもそのスナップショットも直接見ることはできません。 • 子データスペースからは、現在のデータスペースを表示できますが、選択することはできません。 • データスペースの内容にアクセスできません。 • データスペースに対してアクションを実行できません。

許容されるアクション

ユーザーは、次のアクションを実行できます。

子データスペースの作成	プロファイルが子データスペースを作成できるかどうか。
スナップショットの作成	プロファイルがデータスペースからスナップショットを作成できるかどうか。
マージの開始	プロファイルがデータスペースをその親とマージできるかどうか。
アーカイブのエクスポート	プロファイルがエクスポートを実行できるかどうか。
アーカイブのインポート	プロファイルがインポートを実行できるかどうか。
データスペースを閉じる	プロファイルがデータスペースを閉じることができるかどうか。
スナップショットを閉じる	プロファイルがデータスペースのスナップショットを閉じることができるかどうか。
サービスの権限	サービスのアクセス許可を指定します。
作成時の子データスペースの権限	現在のデータスペースから作成される子データスペースのデフォルトのアクセス許可を指定します。

データスペースのマージ

特定のデータスペースでの作業が完了したら、データスペースを作成元のデータスペースに一方向にマージして戻すことができます。マージプロセスは次のとおりです。

1. 親と子の両方のデータスペースは、マージを開始したユーザーと管理者ユーザーを除くすべてのユーザーにロックされます。これらのロックは、マージ操作の間保持されます。ロックすると、データスペースの内容を読み取ることはできますが、変更することはできません。

****注意：****親データスペースに対するこの制限は、直接の変更をブロックすることに加えて、進行中のマージが完了するまで、他の子データスペースをマージできないことを意味します。

2. 子データスペースの作成以降に行われた変更は、親データスペースに統合されます。

3. 子データスペースは閉じられています。
4. 親データスペースのロックが解除されます。

マージの開始

データスペースをその親データスペースにマージするには、次の手順を実行します。

1. 領域のナビゲーションペインでそのデータスペースを選択します。
2. ワークスペースで、メニューから **マージ** を選択します。

変更の確認と承認

データスペースのマージを開始した後、子 (ソース) データスペースの作成以降に行われた変更を確認して、親 (ターゲット) データスペースに適用する変更を決定する必要があります。

注意:

データサービスまたはプログラムを使用してマージを実行する場合、この変更セットのレビューと承認の段階はバイパスされます。自動マージの場合、子データスペース内のすべての変更が親データスペースのデータを上書きします。

変更の受け入れプロセスでは、レビューが必要な変更を要約するサイドバイサイドの比較インターフェイスを使用します。次のデータスペース状態の比較から関連する変更を取得することにより、2 つの **変更セット** 列が取得されます。

- 最初のスナップショットと比較した現在の子データスペース。
- 子データスペースの初期スナップショットと比較した親データスペース。

デフォルトでは、検出されたすべての変更がマージ対象として選択されています。マージから除外する変更の選択を解除できます。ナビゲーションペインでエレメントを選択すると、データモデルのさまざまなスコープに関連する変更を表示できます。

データは現在のデータスペースとその親の両方で変更される可能性が高いため、競合を検出するために、マージには現在のデータスペース、その初期スナップショット、および親データスペースが含まれます。

マージプロセスは、データスペース内のテーブルのアクセス許可の変更も処理します。他の変更と同様に、アクセス制御の変更は、マージに含めるために確認する必要があります。

特定のスコープでマージする変更を決定したら、ボタンをクリックして、そのスコープ内のすべての変更をレビューしたことを示す必要があります。マージを続行するには、すべての変更を確認する必要があります。

変更の種類

マージプロセスでは、次の変更をレビュー対象の変更と見なします。

- レコードとデータセットの作成
- 既存のデータへの変更
- レコード、データセット、または値の削除
- テーブルのアクセス許可への変更

競合の種類

このレビューインターフェイスには、検出された競合も表示されます。同じスコープにソースデータスペースとターゲットデータスペースの両方に変更が含まれている場合、競合が発生する可能性があります。

競合は次のように分類されます。

- レコードまたはデータセットの作成の競合
- エンティティ変更の競合
- レコードまたはデータセットの削除の競合
- 他のすべての競合

マージの完了

すべての変更を確認し、マージ結果に含める変更を決定したら、ナビゲーションペインの ボタンをクリックします。

マージの親データスペースで構成されている子マージポリシーに応じて、後続のプロセスが異なる場合があります。デフォルトでは、結果に検証エラーが含まれている場合でも、マージは完了します。マージ内の親データスペースの管理者は、結果に検証エラーが含まれない場合にのみ子のマージが完了するように、子マージポリシーを設定できます。

ただし、親データスペースの管理者が子マージポリシーを「事前検証マージ」に設定している場合は、マージの結果を保持するための専用データスペースが最初に作成されます。結果が有効な場合、マージ結果を含むこの専用データスペースは自動的に親データスペースにマージされ、それ以上のアクションは必要ありません。

専用のマージデータスペースで検証エラーが検出された場合は、元の親データスペースと、「< 子データスペースの名前>」という名前のマージ結果を含むデータスペースにのみアクセスできます。次のオプションは、ワークスペースの > メニューから使用できます。

- **キャンセル。** マージを中止し、マージ前の状態で子データスペースを回復します。
- **続行。** これを使用して、専用のマージデータスペースに必要な修正を加えた後、マージを再試行できます。

データスペースの子マージポリシーの設定

データスペースの管理者は、マージによって検証エラーが発生するデータスペースが発生する場合に、ユーザーインターフェイスを介して子データスペースのマージのファイナライズをブロックできます。これを行うには、親データスペースのワークスペースから**> **を選択します。データスペースの情報ページで、を に設定します。このポリシーは、すべての子データスペースのこの親データスペースへのマージに適用されます。

注意:

マージが Web コンポーネントを介して実行される場合、子マージポリシーの動作は説明したものと同じです。親データスペースで定義されたポリシーは、子データスペースをマージするときに自動的に適用されます。ただし、この設定は、データワークフローのスクリプトタスクを含むプログラムによるマージでは無視されます。

以下も参照してください。:

- [子のマージポリシー](#)

マージの中止

マージはユーザーセッションのコンテキストで実行され、1 回の操作で完了する必要があります。開始したマージを続行しない場合は、ボタンをクリックして操作を中止できます。

マージを開始した後に別のページに移動すると、マージは破棄されますが、親データスペースと子データスペースのロックは、領域でロックを解除するまで残ります。

ナビゲーションペインでデータスペースを選択し、ワークスペースの ボタンをクリックすると、データスペースのロックを解除できます。子データスペースからロック解除を実行すると、子デー

タスペースと親データスペースの両方のロックが解除されます。親データスペースからロック解除を実行すると、親データスペースのロックが解除されるだけなので、子データスペースのロックを個別に解除する必要があります。

データスペースの比較

データスペースの内容を、リポジトリ内の別のデータスペースまたはスナップショットの内容と比較できます。比較を実行するには、ナビゲーションペインでデータスペースを選択し、ワークスペースから > を選択します。

比較ウィザードでは、現在のデータスペースを比較するデータスペースまたはスナップショットを選択するように求められます。

継承された値と計算された値を持つフィールドを無視するより高速な比較のために、フィルター「永続化された値のみ」を選択します。

以下も参照してください。:

- [コンテンツの比較](#)

データスペースの検証

データスペースのコンテンツのグローバル検証を実行するには、ナビゲーションパネルでそのデータスペースを選択し、ワークスペースで > を選択します。

注意:

このサービスは、現在のデータスペースに含まれるすべてのデータセットを検証する権限がある場合にのみ、ユーザーインターフェイスで使用できます。

データスペースアーカイブ

データスペースのコンテンツは、アーカイブにエクスポートすることも、アーカイブからインポートすることもできます。

エクスポート

データスペースをアーカイブにエクスポートするには、ナビゲーションパネルでそのデータスペースを選択し、ワークスペースで > を選択します。エクスポートされると、アーカイブファイルはサーバーのファイルシステムに保存され、管理者のみがファイルを取得できます。

注意:

詳細については、『管理ガイド』の [アーカイブディレクトリ](#) を参照してください。

アーカイブをエクスポートするには、次の情報を指定する必要があります。

作成するアーカイブの名前	エクスポートされるアーカイブの名前
エクスポートポリシー	必須。 デフォルトのエクスポートポリシーは [データスペースのコンテンツ全体] で、選択したすべてのデータをアーカイブにエクスポートします。 変更セットを使用して、データスペースとその初期スナップショットの違いのみをエクスポートすると便利な場合があります。変更セットを含む 2 つの異なるエクスポートオプションがあります。[更新とコンテンツ全体] と [更新のみ] です。最初のオプションは、現在のすべてのデータと、現在の状態と初期スナップショットの違いを含む変更セットをエクスポートします。2 番目のオプションは、変更セットのみをエクスポートします。どちらのオプションでも比較ページが表示され、この変更セットに含める差異を選択できます。差異はテーブルレベルで検出されます。
エクスポートするデータセット	このデータスペースからエクスポートするデータセット。データセットごとに、その

データ値、権限、および/または情報をエクスポートできます。

インポート

アーカイブからデータスペースにコンテンツをインポートするには、ナビゲーションパネルでそのデータスペースを選択し、ワークスペースで > を選択します。

選択したアーカイブに変更セットが含まれていない場合、データスペースの現在の状態はアーカイブのコンテンツに置き換えられます。

選択したアーカイブにコンテンツ全体と変更セットが含まれている場合は、変更セットの違いを現在の状態とマージするために、変更セットを適用することを選択できます。変更セットを適用すると、比較画面が表示されます。ここで、変更セットの違いを選択してマージできます。

選択したアーカイブに変更セットのみが含まれている場合は、変更セットの違いを選択して、比較画面でマージできます。

データスペースを閉じる

データスペースが不要になった場合は、閉じることができます。閉じられると、データスペースはユーザーインターフェイスの領域に表示されなくなり、アクセスできなくなります。

管理者は、リポジトリからクリーンアップされていない限り、閉じたデータスペースを再度開くことができます。

データスペースを閉じるには、> を選択します。

以下も参照してください。:

- [未使用のデータスペースとスナップショットを閉じる](#)

[ユーザーガイド](#) [目次](#)

スナップショット

- [スナップショットの概要](#)
- [スナップショットの作成](#)
- [スナップショットの内容の表示](#)

- スナップショット情報
- スナップショットの比較
- スナップショットの検証
- エクスポート
- スナップショットを閉じる

スナップショットの概要

スナップショットは、データスペースの読み取り専用コピーです。スナップショットは、特定の時点でのデータスペースの状態と内容の記録として存在します。

以下も参照してください。:

- スナップショット

スナップショットの作成

スナップショットは、領域のナビゲーションペインでそのデータスペースを選択し、ワークスペースで > を選択することにより、データスペースから作成できます。

次の情報が必要です。

識別子	スナップショットの一意の識別子。
ラベル	スナップショットに関連付けられたローカライズされたラベルと説明。

スナップショットの内容の表示

スナップショットの内容を表示するには、スナップショットを選択し、ワークスペースから > を選択します。

スナップショット情報

> を選択すると、スナップショットに関連付けられている情報を変更できます。

ドキュメント	スナップショットに関連付けられたローカライズされたラベルと説明。
現在の所有者	スナップショットの所有者。デフォルトでは、スナップショットの情報と権限を変更できます。所有者は、必ずしもスナップショットの作成者である必要はありません。
所有者を変更	スナップショットの現在の所有者が、その「現在の所有者」プロパティを変更することにより、別のプロファイルに所有権を与えることが許可されているかどうか。値が「禁止」の場合、管理者のみが所有者を変更できます。

スナップショットの比較

スナップショットの内容を、リポジトリ内の別のスナップショットまたはデータスペースの内容と比較できます。比較を実行するには、スナップショットを選択してから、ワークスペースから**> **を選択します。

比較ウィザードでは、現在のスナップショットを比較するデータスペースまたはスナップショットを選択するように求められます。

継承された値と計算された値を持つフィールドを無視するより高速な比較のために、フィルター「永続化された値のみ」を選択します。

以下も参照してください。:

- [コンテンツの比較](#)

スナップショットの検証

スナップショットの内容のグローバル検証を実行するには、ワークスペースで>を選択します。

注意:

このサービスを使用するには、スナップショットに含まれるすべてのデータセットを検証する権限が必要です。

エクスポート

スナップショットをアーカイブにエクスポートするには、そのスナップショットを開き、ワークスペースで > を選択します。エクスポートされると、管理者のみがアーカイブを取得できます。

注意:

詳細については、『管理ガイド』の [アーカイブディレクトリ](#) を参照してください。

アーカイブをエクスポートするには、次の情報を指定する必要があります。

作成するアーカイブの名前	エクスポートされるアーカイブの名前
エクスポートするデータセット	このスナップショットからエクスポートするデータセット。データセットごとに、そのデータ値、権限、および情報をエクスポートするかどうかを選択できます。

スナップショットを閉じる

スナップショットが不要になった場合は、閉じることができます。スナップショットを閉じると、領域の関連するデータスペースの下にスナップショットが表示されなくなり、アクセスできなくなります。

管理者は、リポジトリからクリーンアップされていない限り、閉じたデータスペースを再度開くことができます。

スナップショットを閉じるには、> を選択します。

以下も参照してください。:

- [未使用のデータスペースとスナップショットを閉じる](#)

データセット

- データセットの概要
- データセットの作成
- データの表示
- データの編集
- ファイルのインポートおよびエクスポートサービス
 - XML のインポートとエクスポート
 - CSV のインポートとエクスポート
- 既存のデータセットの操作
- データセットの継承

データセットの概要

- 概要
 - データセットとは
 - データセットに関連する基本概念
- データユーザーインターフェイスの使用

概要

データセットとは

データセットは、基礎となるデータモデルによって提供される構造定義に基づくデータのコンテナです。データモデルが公開されると、その定義に基づいてデータセットを作成することができます。そのデータモデルが後で変更されて再公開されると、関連するすべてのデータセットが自動的に更新されて一致します。

データセットでは、実際のデータ値を調べて操作できます。テーブルに適用されるビューにより、データのネイチャーとデータへのアクセス方法に最も適した方法でデータを表すことができます。検索とフィルターを使用して、データを絞り込んで検索することもできます。

データセットレベルでアクセスを制御するために、さまざまな役割にさまざまな権限を付与することもできます。したがって、カスタマイズされたアクセス許可を使用すると、特定のユーザーが

データを表示および変更できるようにし、他のユーザーにはデータを非表示にすることができます。

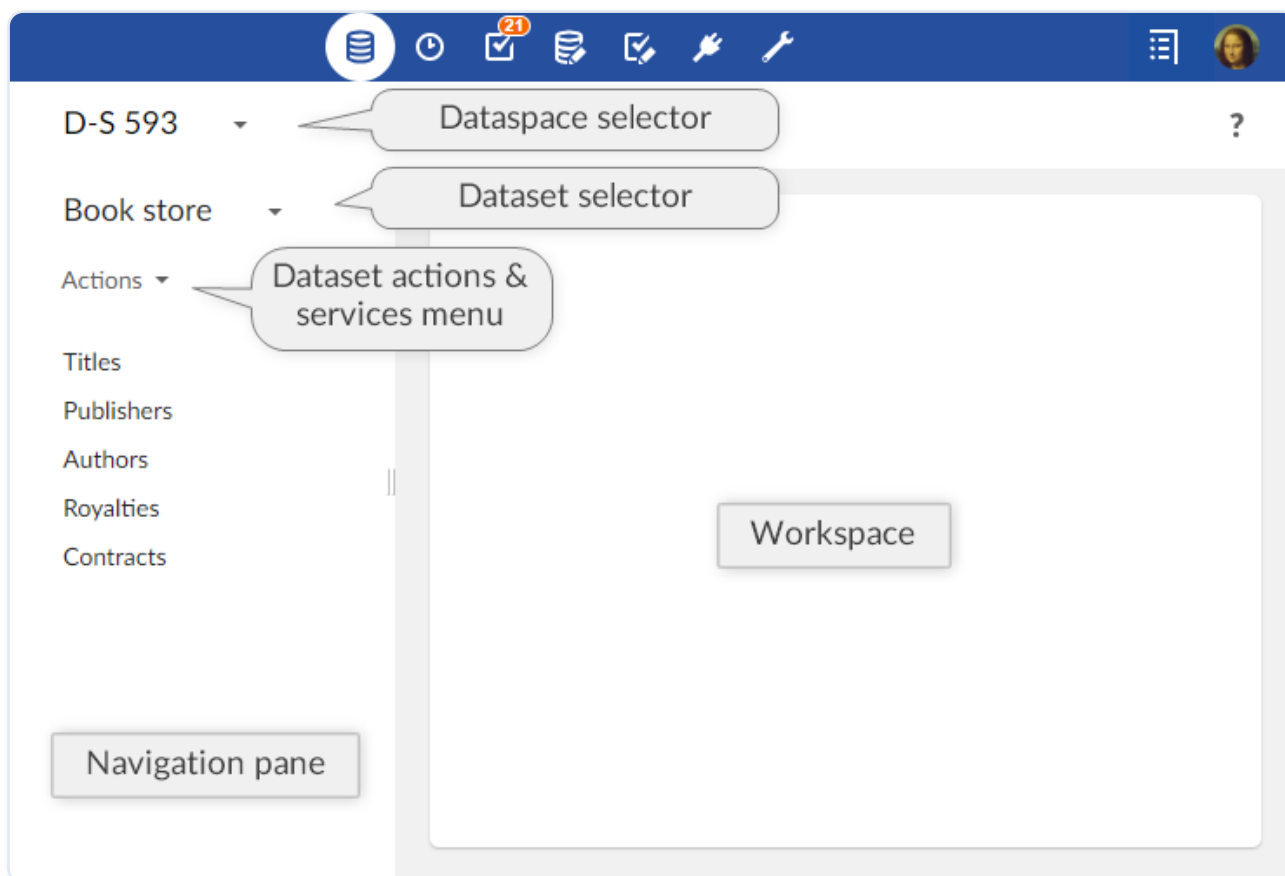
データセットに関連する基本的な概念

次の用語の基本的な理解は、データセットを操作するときに役立ちます。

- [dataspace](#)
- [dataset](#)
- レコード
- フィールド
- 主キー
- 外部キー
- テーブル (データセット内)
- [group](#)

データユーザーインターフェイスの使用

データセットは、[\[高度なパースペクティブ\]](#)を使用して、または特別に構成されたパースペクティブから、領域で作成、アクセス、および変更できます。許可されたユーザーのみがこれらのインターフェイスにアクセスできます。



ナビゲーションペインのメニューを使用して、データセットを選択または作成します。次に、データセットのデータ構造がナビゲーションペインに表示され、レコードフォームとテーブルビューがワークスペースに表示されます。

ワークスペースでデータセットのテーブルを表示しているときに、ボタン  に、表示されるレコードを絞り込むために適用できる検索とフィルターが表示されます。

データセットレベルでの操作は、ナビゲーションペインのメニューにあります (サービスはリストの下部にあります)。

以下も参照してください。:

- データセットの作成
- クイック検索
- ユーザーインターフェイスでのレコードの操作
- 継承

関連する概念:

- データモデル
- データスペース

ユーザーガイド 目次

データセットの作成

- ルートデータセットの作成
- 継承する子データセットの作成

ルートデータセットの作成

新しいルートデータセット、つまり親データセットから継承しないルートデータセットを作成するには、[userdata_intro.html#ui](#)  メニューを選択します。ナビゲーションペインで、ポップアップの ボタンをクリックし、ウィザードを実行します。

注意:

この領域は、または特別に構成されたパースペクティブから許可されたユーザーのみが使用できます。

ウィザードでは、新しいデータセットの基礎となる 3 つのデータモデルパッケージモード (パッケージ、埋め込み、または外部) のいずれかを選択できます。

- パッケージデータモデルは、Web アプリケーションであるモジュール内にあるデータモデルです。
- 埋め込みデータモデルは、TIBCO EBX® リポジトリ内で完全に管理されるデータモデルです。

データセットのベースとなるデータモデルを見つけたら、スペースや特殊文字を使用せずに一意の名前を指定する必要があります。オプションで、データセットのローカライズされたラベルを提供できます。これは、ユーザーの言語設定に応じて、ユーザーインターフェイスでユーザーに表示されます。

注意:

データセットを複製する場合、テーブルの内容はコピーされません。

継承する子データセットの作成

継承メカニズムにより、データセットに親子関係を持たせることができます。これにより、デフォルト値が子孫によって祖先から継承されます。子データセットを作成できるようにするには、基になるデータモデルでデータセットの継承を有効にする必要があります。

子データセットを作成するには、ナビゲーションペインで [userdata_intro.html#ui](#) を選択し、目的の親データセットの横にある  ボタンをクリックします。

データセットは自動的に親データセットと同じデータモデルに基づくため、提供する必要のある情報は一意の名前と、オプションでローカライズされたラベルのみです。

以下も参照してください。:

- データセットの継承

ユーザーガイド [目次](#)

データの表示

TIBCO EBX® は、レコードを一覧表示するさまざまな方法を提供します。この章では、「ビュー」の概念に至るまで、さまざまな方法で、さまざまなユーザープロファイルに従って、レコードを並べ替え、検索、および表示する方法について説明します。

- 「ビュー」メニュー
- データの並べ替え
 - 列で並べ替え
- クイック検索
 - 特殊文字
 - クイック検索結果のハイライト表示
- データの検索とフィルタリング
 - フィールドの検索

- 検証フィルター
- 複雑なフィルターの作成
- カスタムテーブル検索
- 技術的な制限事項
- ビュー
 - ビューの説明
 - 単純な表形式のビュー
 - 階層ビュー
 - タイルビュー
 - ビューの共有
 - ビューパブリケーション
- ビューの管理
 - 推奨ビューの管理
 - 「ビュー」メニューツールバー
- グリッド編集
 - コピー/貼り付け
- History

「ビュー」メニュー

「ビュー」ドロップダウンメニューを使用すると、使用可能なすべてのビューと管理機能にアクセスできます。

ビューは、リストされている各ビューで利用できる「ビュー」メニューツールバーで直接管理されます: 「ビュー」メニューツールバー。

ビューをグループ化することもできます。管理者は、「ビューグループ」テーブルの下の「ビューの構成」でグループを事前に定義する必要があります。エンドユーザーは、ビューの作成または変更時に「ビューグループ」フィールドを使用して、ビューをグループに属するものとして設定できます。詳細については、「[ビューの説明](#)」を参照してください。

データの並べ替え

ソート基準は、レコードが表示される順序を制御します。

テーブルの左上にある ボタンを使用して、特定の並べ替え条件を定義します。

並べ替えには 2 つのタイプがあります。

- クイック検索で使用される関連性による並べ替え
- 列で並べ替えます。

列で並べ替え

ダイアログボックスには、次のものがあります。

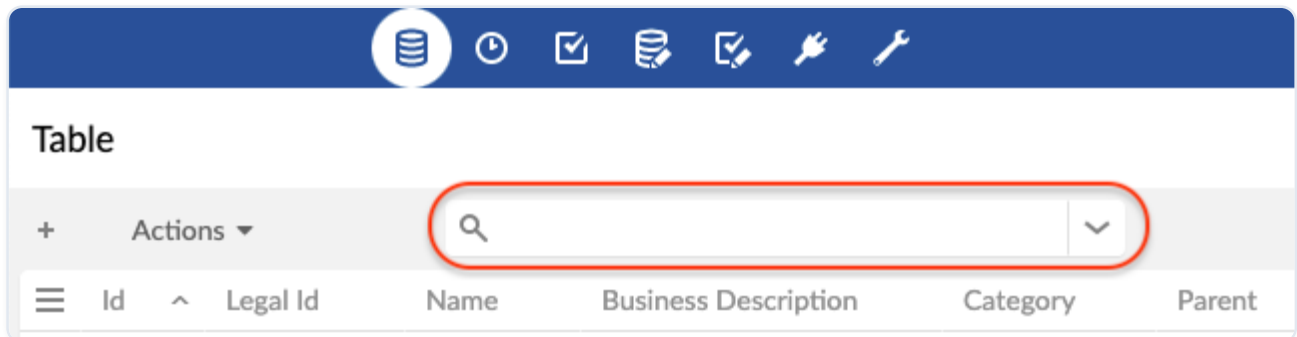
- 左側のソートされた列のリスト
- 右側のソートされていない列のリスト

← → 矢印ボタンを使用して、列をあるリストから別のリストに切り替えます。

↑ ↓ 矢印ボタンを使用して、列の優先度を変更します。

列の並べ替え順序を変更するには、マウスオーバーに表示される (昇順) または (降順) ボタンを使用します。

クイック検索



クイック検索を使用すると、表形式、階層、またはタイルビューで結果を簡単に見つけることができます。

大文字と小文字は区別されません。複数の用語を一度に (スペースで区切って) 検索できます。デフォルトでは、見つかったレコードは関連性によって並べ替えられます。この検索には、表示されない列も含め、検索可能なすべての列が含まれます。列のクイック検索ができない場合は、列ヘッダーにマウスポインターを移動するとアイコンが表示されます。

以下も参照してください。:

- 検索

特殊文字

クイック検索では、文字列の検索を絞り込むための特殊文字が提供されます。文字列ベースではない列はこの構文をサポートしておらず、「コード」などの特定の検索戦略もサポートしていないことに注意してください。

+ ... 必須	結果に単語が含まれている必要があります。検索エンジンがこの単語を検索から除外しないようにします。 注意: +" phrase " または +(group) で動作します。 <i>例:</i> +flute +bach └, 必須の <i>flute</i> と必須の <i>bach</i> の結果を検索します。<i>flute</i> のみまたは <i>bach</i> のみの結果は無視されます。
- ... 含まない	結果から単語を除外します。 注意: -" phrase " または -(group) で動作します。 <i>例:</i> bach -flute └, <i>bach</i> を使用して結果を検索しますが、<i>flute</i> は使用しません。
... ~ あいまい検索	検索で単語の 2 文字を変更して検索できることを指定します。

	1 文字を変更できるようにするには、~1 を使用します。 注意: " phrase "~ と連携してフレーズ内の 2 つの単語を変更します。 例: handel~ ↳ <i>handel</i> で、1 文字または 2 文字が異なる単語で結果を検索します。
? シングルジョーカー	不明な文字を置き換えます。 例: ?uttle ↳ 任意の文字で始まり、<i>uttle</i> で終わる単語で結果を検索します。
* 拡張ジョーカー	いくつかの不明な文字を置き換えます。 例: rachmanino* ↳ <i>rachmanino</i> で始まる単語で結果を検索します。
"... " フレーズ	フレーズの完全一致を見つけます。 注意: + - ~ で囲むことができます。 例: "Johann Sebastian Bach" ↳ 正確に <i>Johann Sebastian Bach</i> を含む結果を検索します。

(...) group

単語をグループ化して、特殊文字 **+** または **-** を単語に適用できるようにします。

注意:

グループのグループを作成できます。

例:

bach +(flute piano)

↳ bach に加えて、必須の flute または piano で結果を検索します。

注意:

これらの特殊文字は、ドキュメント検索エンジンでも使用できます。

クイック検索結果のハイライト表示

クイック検索結果はハイライト表示され、検索クエリに一致するデータの部分を視覚的に識別することでユーザーエクスペリエンスを向上させ、結果内の関連レコードを見つけやすくします。ハイライト表示は表形式ビューでのみ有効であり、非表示の列には適用されません。

既知の制限事項

計算されたラベルを使用して表示される外部キーが検索に一致する場合、ハイライト表示が期待どおりに表示されないことがあります。これは、ハイライト表示メカニズムがラベルではなく生の値で動作するため、表示されるラベルがクエリに一致した生の値に置き換えられる可能性があるためです。

既知の制限事項

カスタムウィジェットを使用するフィールドでは、ハイライト表示は表示されません。

データの検索とフィルタリング

検索ペインはデフォルトで非表示になっており、表形式、階層、またはタイルビューのツールバーのクイック検索の右側にある  (/content/dam/onebx/jp/ja/ebx/webhelp/6-2-3/resources/images/symbols/Open Search Pane.svg) アイコンからアクセスできます。

クイック検索と基準行を組み合わせて検索を絞り込みます (結果をより少ないレコードに制限します)。

チェックをオフにすると、基準行を非アクティブ化できます。非アクティブ化された基準は、保存中に保持されません。

各基準の行末にあるゴミ箱ボタン  は、基準を完全に削除します。

検索に適用されたフィルターを保存するには、ボタンを使用します。保存では、クイック検索とすべてのアクティブな条件が考慮されます。

保存したフィルターを呼び出すには、ボタンを使用します。読み取りは、クイック検索と条件パネル全体を置き換えます。ボタンをクリックして、新しい検索を開始します。

ビューが適用されると、その構成に従って表示されるようになります。したがって、検索パネルの既存の基準はすべて削除されます。ビューには、ビューと同時に適用される一連の検索条件を含めることができます。

一部の演算子 (「text-search」など) では、Lucene 正規表現を使用できます。詳細については、[Lucene 正規表現パターンの技術仕様](#) を参照してください。

フィールドの検索

列挙として定義されていないタイプ `osd:locale` のフィールドと、タイプ `osd:resource` のフィールドを除く、すべての検索可能なフィールドが使用可能です。

検証フィルター

フィールド選択では、検証基準は最後に実行された検証の時点のレコードを表示します。

注意:

このフィルタリングは、ワークスペースからテーブルレベルで、またはナビゲーションペインからデータセットレベルで > を選択することによって少なくとも 1 回検証されたテーブルのレコードにのみ適用されます。

(検証メッセージとは関係なく) 検証の重大度レベルでフィルタリングするには、検証基準を使用します。使用可能なレベルは、、、およびです。

(検証の重大度レベルとは関係なく) 検証メッセージをフィルタリングするには、検証基準を使用します。

複雑なフィルターの作成

論理演算子でグループ化された複数の条件を使用して、複雑なフィルターを作成します。グループはツリービューで表示され、さまざまな機能を利用して、最も複雑なフィルターの構築も合理化します。

たとえば、Bob と Smith が、それぞれ フィールドと フィールドのフィルターに渡される値であるレコードを考えます。**contains** 条件が両方のフィールドに使用されている場合、論理演算子は次のようにレコードフィルタリングに影響を与えます。

- AND：レコードを表示するには、このグループの条件のすべてを満たす必要があるため、検索が制限されます。Bob Smith を含むレコードのみが結果に表示されます。
- OR：レコードを表示するには、このグループ内の基準の 1 つを満たすだけでよいいため、検索範囲が拡張されます。フィールドに Bob が含まれるレコード、または フィールドに Smith が含まれるレコードが表示されます。たとえば、結果には Jon Smith や Bob Jones が含まれる場合があります。
- NOT：レコードを表示するには、指定された条件とは逆の条件が満たされるように、グループまたは基準を反転します。
 - NOT AND：Bob Smith ではないすべてのレコードが返されます。
 - NOT OR：Bob または Smith を含まないレコードのみが返されます。たとえば、Peter Jones と Mary Stewart が返されます。
 - 単一の基準に適用される NOT：その基準のみの条件を反転します。たとえば、グループの条件が AND で、フィールドに NOT が適用された場合、フィールドに Smith が含まれないレコードのみが返されるため、レコード Bob Smith は結果から除外されます。

下表は、グループを操作するときを使用できるアクションを示しています。

演算子の選択	AND 演算子と OR 演算子を切り替えます。
マウスホバー	演算子に対して、以下を選択できます。 • NOT - グループを反転します。 • ☐ をクリックしてグループを複製します。 • ☐ をクリックして基準のグループ化を解除します。

	個々の基準上にマウスポインターを移動して [NOT] を選択し、その基準に適用する条件を反転します。
ドラッグアンドドロップ	基準に接続された線をドラッグします。 • 上/下に移動して、含まれるグループ内で順序を変更します。 • 同じグループ内の別の基準に対して、新しいサブグループを作成します。 • 別の演算子に対して、基準をそのグループに移動します。
グループ基準ボタン	このボタンを使用して、グループの編集を有効にします。グループ化する基準の横にあるチェックボックスを選択します。このアクションにより、選択した条件が新しいグループに移動されることに注意してください。

カスタムテーブル検索

下位互換性のために、レコードのカスタム検索とフィルタリングの機能は引き続き機能し、ワークスペースのアイコン  からアクセスできます。アイコンと機能は、少なくとも 1 つのカスタムフィルターが存在する場合にのみ使用できます。

データモデルのテーブルごとに、追加のカスタムフィルターを指定できます。

以下も参照してください。:

- [テーブルフィルターのカスタマイズ](#)

技術的な制限事項

最適化されたテキスト検索フィルターは、OR 演算子を使用して、最適化されていないフィルターや別のテーブルを含むフィルターと組み合わせることはできません。

この状況の典型的な例としては、次の OR フィルターを使用したテキスト検索が考えられます。

- 継承されたフィールド。
- ローカル依存関係のない計算値。
- 別のテーブルのフィールド (リンクされたフィールドなど)。

この技術的制限は、「テキスト検索」演算子が「OR」グループ内にあるため、リクエストを解決できません」(技術的制限)として表示されます。

ビュー

ターゲットユーザーに応じて、EBX® のテーブルの表示をカスタマイズすることができます。

ビューには、**表形式**、**階層**、**タイル**の3種類があります。

ワークスペースで>を選択すると、ビューを作成できます。ビューを適用するには、* > *ビューの名前**でビューを選択します。

次の3種類のビューを作成できます。

- 「単純表形式ビュー」：表示されたレコードを並べ替えてフィルタリングするためのテーブルビュー。
- 「階層ビュー」：関係に基づいて異なるテーブルのデータをリンクするツリービュー。
- 「タイルビュー」：行と列に配置されたカードの形式でレコードを表示するビュー。このビューでは、目を引く形式で情報を表示したり、表示されたレコードを並べ替えたり、フィルター処理したりできます。

ビューの説明

ビューを作成または更新する場合、最初のページでビューに関連する一般情報を指定できます。

ドキュメント	ビューに関連付けられたローカライズされたラベルと説明。
所有者	ビューの所有者の名前。このユーザーはそれを管理および変更できます (管理者とデータセット所有者のみが利用可能)。
共有	「ビュー」メニューからこのビューを使用できる他のプロファイル。 注意:

	権限が必要です。 ビューの権限 を参照してください。
ビューモード	単純表形式ビュー、階層ビュー、またはタイルビュー。
ビューグループ	このビューが属するグループ (存在する場合)。

単純な表形式のビュー

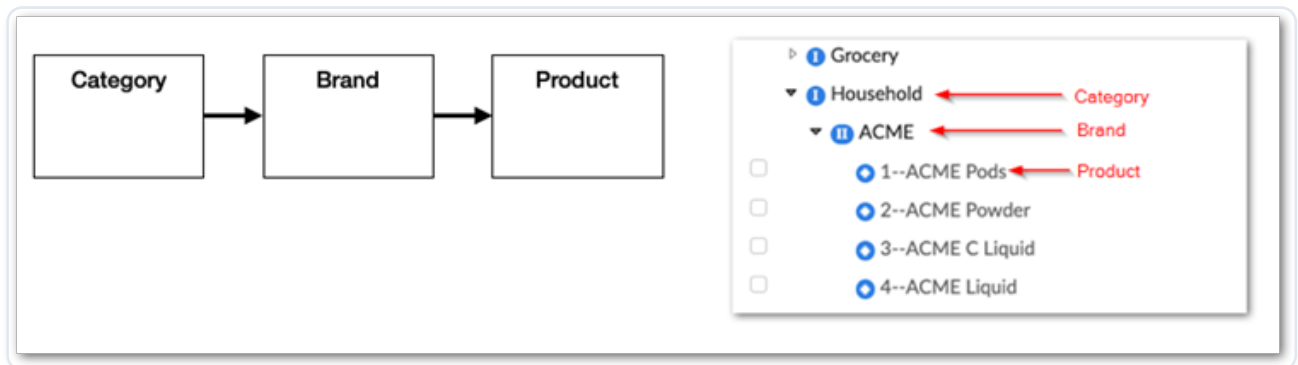
単純な表形式のビューでは、レコードをフィルタリングするための基準を定義したり、テーブルに表示される列を選択したりすることができます。

表示列	テーブルに表示される列を指定します。
並べ替え列	テーブル内のレコードの並べ替え順序を指定します。 データの並べ替え を参照してください。
フィルター	このテーブルビューに表示されるレコードのフィルターを定義します。 基準エディター を参照してください。
最初のリクエスト	[手動] に設定すると、このビューのユーザーは、データを表示する前にフィルターまたは並べ替え基準を適用する必要があります。
ページ数の制限	表示可能なレコードの数に制限を設定します。
グリッド編集	有効にすると、このビューのユーザーはグリッド編集に切り替えることができるため、表形式のビューから直接レコードを編集できます。

作成と複製を無効化	[はい] の場合、このビューのユーザーは、グリッド編集からレコードを作成または複製できません。
テーブルの上部のツールバー	このテーブルビューの上で使用するツールバーを定義します。
テーブル行上のツールバー	このテーブルビューの各行で使用するツールバーを定義します。

階層ビュー

階層ビューでは、データがツリー状に表示されます。階層ビューを使用して、データ間のさまざまな関係を視覚化したり、共通の値をグループ化したりします。階層ビューを作成するには、含めるデータとその表示方法を決定するディメンションを定義します。次の例と画像について考えてみましょう。ディメンションは --> --> で、カテゴリごとにブランドごとの製品が表示されます。



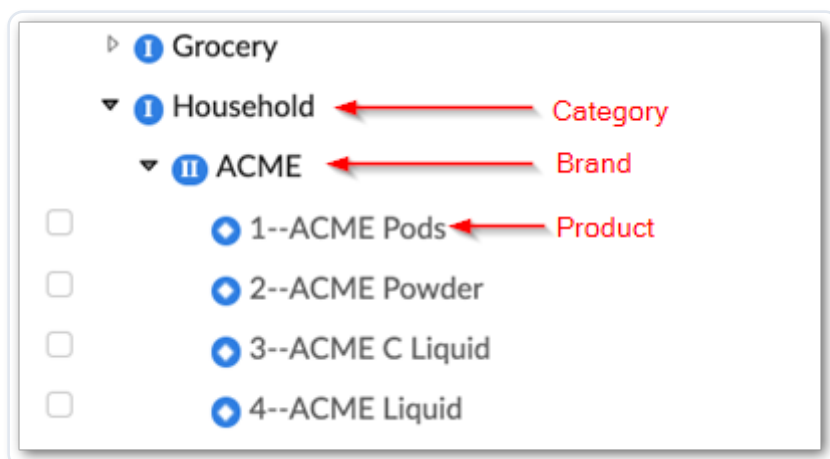
階層ディメンションとメンバー

各階層は、そのディメンションによって定義されます。ディメンションはディメンションレベルに分割されます。メンバーは階層ノードであり、レコードまたはフィールド値を表します。メンバーは、他のメンバーの親になることも、子を持たないリーフノードになることもできます。階層ビューを作成するテーブルは、ターゲットテーブルと呼ばれます。ターゲットテーブルのメンバーは、再帰的な階層の場合を除き、最下位レベルのリーフです。

以下は、次の図の説明です。

- Grocery と Household は、Category ディメンションレベルのメンバーです。これは最上位のディメンションレベルです。
- ACME は Brand ディメンションレベルのメンバーであり、中間レベルにあります。ディメンションは、多くの中間レベルを持つことができます。

- Product ディメンションレベルでリストされている個々の製品は、ターゲットテーブルのリーフメンバーです。これは最下位ディメンションレベルです。



階層ビューの作成

このトピックでは、階層ビューを作成するための高度な手順について説明します。特定の階層タイプのビューの作成に関する追加の詳細については、次のトピックを参照してください。

- 不規則な階層
- 階層によるグループ化
- 時間ベースの階層ディメンション

階層ビューを作成するには、次の手順を実行します。

1. ターゲットテーブルのメニューから、を選択します。
2. 次の情報を指定し、をクリックします。
 - ラベルや説明など、ビュー関連のドキュメントを追加します。
 - アクセス権限を設定します。
 - プロパティで、を選択します。
3. ボックスを使用して、ディメンションレベルを逆方向に操作して階層を定義します。
 - a. 最下位：このレベルのメンバーは、ターゲットテーブルに基づいて事前に決定されます。
 - b. 中間レベル：次に高いディメンションレベルに対応する属性から開始し、最も高いレベルに移動するまで展開を続けます。
 - c. 最上位：最上位ディメンションレベルに対応する属性を選択します。
4. 必要なオプションを確認し、必要な変更を加えてから、をクリックします。

5. 各ディメンションレベルのメンバーのラベル、フィルター、および並べ替えオプションを指定します。
 - 画面上部の階層リンクでディメンションレベルを選択し、ディメンションレベル間を移動します。
 - 必要に応じて、ラベル、フィルター、並べ替え方法を調整します。
6. をクリックして、階層ビューを表示します。

不規則な階層

不規則な階層には、親が 1 レベルをスキップするメンバーが含まれます。このタイプの階層の例は、Country、State、および City ディメンションを使用して示されています。一部の国には州がありませんが、すべての国には都市があります。後者の場合、都市のメンバーは国ディメンションレベルに直接ロールアップされます。

注意:

不規則な階層では、1 レベルのスキップのみがサポートされます。

EBX® で不規則な階層を表示する場合、次のことが適用されます。

- データモデルでは、ターゲットテーブルからのすべての外部キー関係を定義する必要があります。上記の例を使用すると、City テーブルは State テーブルと Country テーブルへの外部キーを保持する必要があります。もちろん、州のない都市の場合、このフィールドは未定義のままです。
- 階層ビューを構成するときは、 で オプションを有効にします。このプロパティは、下図で強調表示されています。

Dimension

- Store
 - CityFK (City)
 - CityName
 - Creator
 - Last updater
 - CityFK (City)
 - CountryFK (Country)

Display records in a new window ☐ Yes ☒ No

Prune hierarchy ☐ Yes ☒ No

Display orphans ☒ Yes ☐ No

Display root node ☒ Yes ☐ No

Root node label English (United States)

Toolbar on top of hierarchy [not defined]

Advanced properties

Remove recursive root leaves ☐ Yes ☒ No

Detect cycle ☐ Yes ☒ No

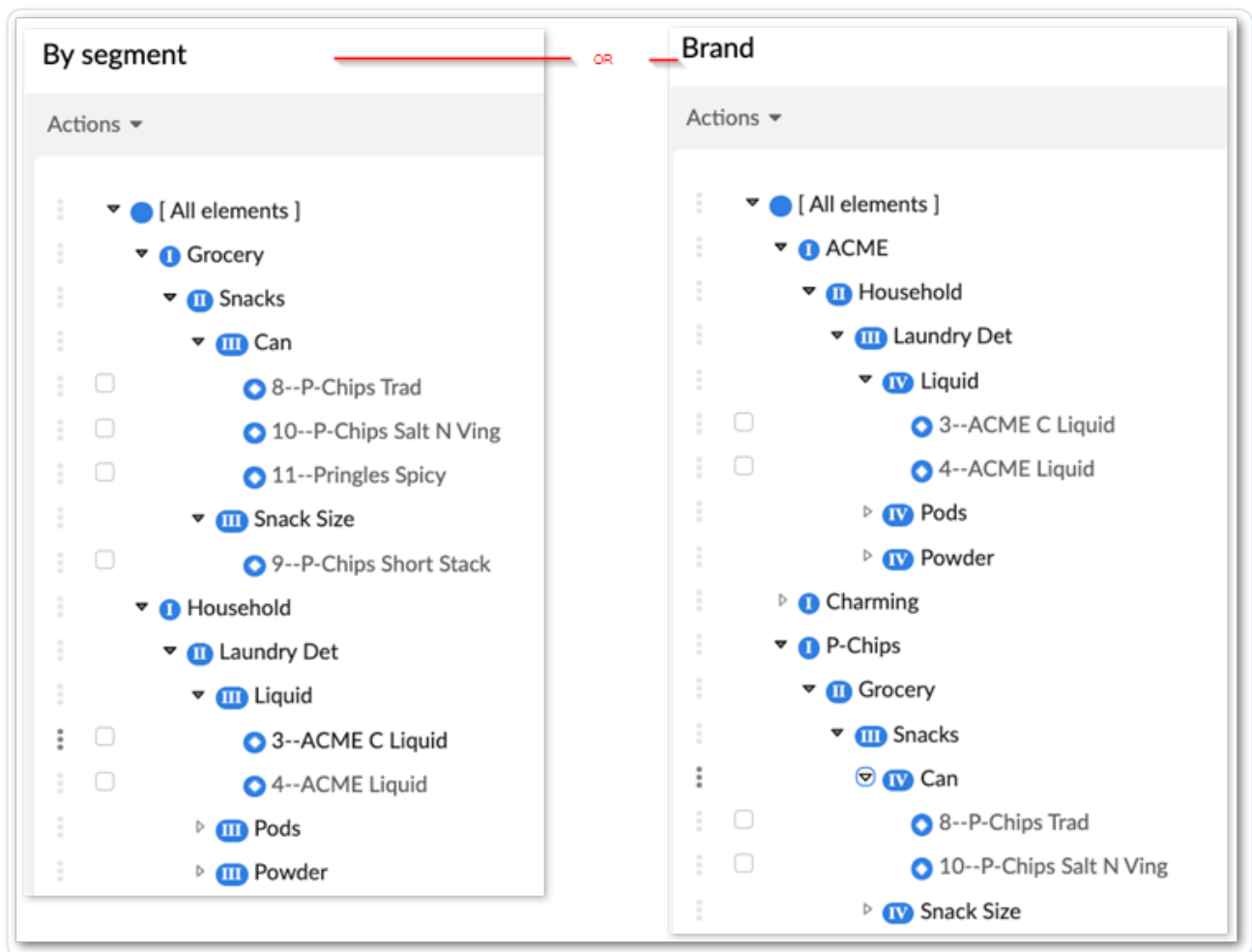
Allow skipping level ☒ Yes ☐ No

Detect leaf ☒ Yes ☐ No

Allow children count ☐ Yes ☒ No

階層によるグループ化

Group By 階層のディメンションは、同じテーブルにある列を使用して定義されます。許可される列のデータ型は、外部キーまたは列挙可能な型 (文字列、整数、列挙型、ブール値) です。これは、共通の値をグループ化する階層ビューを定義するための強力なツールを提供します。階層別グループは、ディメンションレベルの順序を設定する際にも柔軟性を提供します。たとえば、Product テーブルに Brand 列、Segment 列、Type 列、Packaging 列が含まれる場合、これらの列を任意の順序で使用して、Segment --> Type --> Packaging、または Brand --> Segment --> Type --> Packaging で Product を表示します。

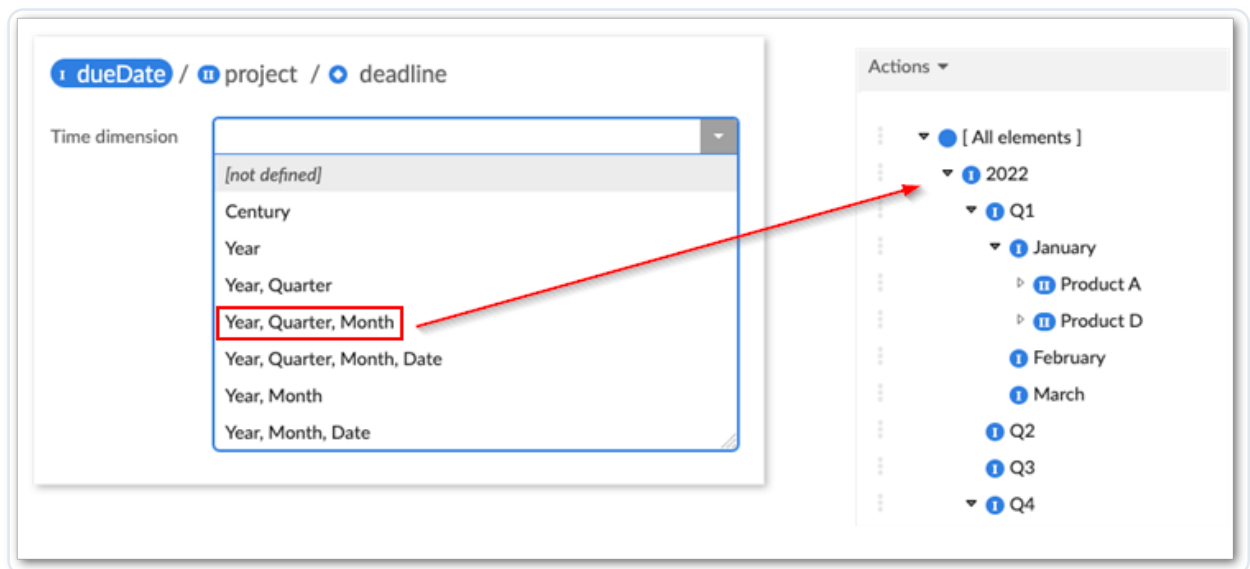


時間ベースの階層ディメンション

日付データ型のフィールドを使用して、時間ベースの階層ディメンションを定義します。ビューを構成するときは、いくつかの事前定義された表示オプションから選択します。一部のオプションは、時間をより小さな単位に分割します。たとえば、プロジェクトのフィールドを使用して、2022年の第1四半期にすべてのプロジェクトの期日を月別に表示するように指定できます。

階層ビューで時間ベースのディメンションを使用するには、次の手順を実行します。

1. 新しい階層ビュー構成を作成します。
 - a. テーブルを表示して、メニューを開き、を選択します。
 - b. 名前を入力し、必要に応じて共有オプションを指定します。
 - c. でを選択し、をクリックします。
2. 階層のルートディメンションに使用されるフィールドが日付データ型であることを確認し、をクリックします。
3. 階層構造の階層リンク画面上部にあるフィールドをクリックし、メニューを使用して、ディメンションの表示方法を指定します。



4. 階層を保存して表示します。

階層ビュー構成オプション

このフォームでは、階層ビューオプションを構成できます。

新しいウィンドウ内にレコードを表示	[はい] の場合、レコードを含む新しいウィンドウが開きます。それ以外の場合は、同じウィンドウの新しいページに表示されます。
階層の整理	[はい] の場合、子がなく、ターゲットテーブルに属していない階層ノードは表示されません。
親なしの表示	[はい] の場合、親のない階層ノードが表示されます。
ルートノードの表示	[いいえ] の場合、階層のルートノードはビューに表示されません。
ルートノードラベル	階層ルートノードのローカライズされたラベル。
階層の上部のツールバー	階層の上部にツールバーを設定できます。
サイクルの検出	再帰的な場合にサイクルの検出と表示を許可すると、最も古いノードレコードがサイクルルートとして選択されます。制限事項：検索

	モードまたは整理モードでは機能しません。
リーフの検出	メンバーがリーフであるかどうかを検出できます。リーフの検出は、大量のデータに対して非常にコストがかかります。したがって、階層ビューを表示するためにクエリ応答が遅延する場合は、このオプションを無効にすることをお勧めします。このプロパティは、孤立した親メンバーに対しては、常に無効になっています。

ラベル

別のテーブルを参照するディメンションレベルごとに、階層内の対応するノードのローカライズされたラベルを定義できます。ビルトインのウィザードを使用して、ラベルを取得するフィールドを選択できます。

フィルター

基準エディターを使用すると、ビューのレコードフィルターを作成できます。

以下も参照してください。:

- [基準エディター](#)

並べ替え戦略

ディメンションレベルごとに、次の並べ替え戦略のいずれかを選択できます。

デフォルト	ノードはラベルのアルファベット順にソートされます。
カラムで並べ替え	ノードは、選択した列で並べ替えられます。方向 (昇順/降順) は、列ごとに選択できます。

順序フィールドで並べ替え

ノードは非表示の数値フィールドでソートされます。これにより、ユーザーは階層ビューで兄弟ノードの順序を動的に変更できます。この戦略は、テーブルに少なくとも1つの「非表示」数値フィールドがある場合にのみ使用できます。

このオプションを有効にするには、階層ビューが適用されるテーブルで定義されている適格な順序フィールドを指定する必要があります。順序付けフィールドは、「整数」データ型であり、データモデル定義の詳細プロパティで、デフォルトビューモードの「非表示」に設定されている必要があります。

順序付けフィールドが「読み取り専用」モードの場合、または階層にフィルターが設定されている場合を除いて、任意のフィールドを再配置できます。

注意:

データは階層ビューによって上書きされるため、データを含むことを目的としたフィールドを順序付けノードとして指定しないでください。

階層ノードに対するアクション

階層ビューの各ノードには、コンテキストアクションを含むメニュー  があります。

リーフノードは、 を使用して、親レコードから分離できます。その後、レコードはツリー内の孤立したノードになり、コンテナのノードの下に編成されます。

リーフノードは、 を使用して親ノードを変更することもできます。データモデルに従って、ノードが複数の親との関係を持つことができる場合、そのノードは現在の親の下と、他の親ノードの下に両方に追加されます。それ以外の場合、リーフノードは他の親ノードの下に移動されます。

階層ノードでのクイック検索

階層ノードでのクイック検索は表形式ビューと似ていますが、ターゲットテーブルのノードのレコードと、ノードのラベルに定義されたフィールドにのみ適用されます。

クイック検索ツールの下には、デフォルトでオフになっている**すべての列でのクイック検索**チェックボックスがあります。デフォルトでオンにしたい場合は、**詳細プロパティ**に移動して対応するオプションを有効にしてください。このオプションはパフォーマンス上の理由からデフォルトで無効になっていることに注意してください。

タイルビュー

このビューは、グリッドに配置されたタイルとしてレコードを表示します。各タイルには、タイトル (必須)、サブタイトル、画像、説明の情報を含めることができます。タイトルと説明については、フィールドを選択して値を自動的に入力するか、表示するテキストを手動で指定するか、両方の方法を組み合わせて使用できます。さらに、ビューに適用される並べ替えおよびフィルター条件を定義できます。ツールバーも追加できます。次の表は、タイルビューを作成するときに使用できるプロパティの一覧と定義を示しています。

タイトル	最初の行に表示されるタイルタイトルを定義します。
サブタイトル	2 行目に表示されるサブタイトルを定義します。
画像	タイル画像を定義します。
画像 > デフォルト画像 URL	代替画像の URL を定義します。
画像 > URL	画像の URL を定義します。
画像 > 高さ (ピクセル)	画像スロットの高さをピクセル単位で設定します (10 から 250)。スロットの幅は常に 180 ピクセルです。画像のサイズが意図したスロットのサイズを超える場合、ビューで定義されたトリミングルールに従って調整されます。
画像 > トリミング	画像のトリミングを定義します。[カバー] の場合、使用可能なすべてのスペースを占有するように、画像が拡大または縮小されます。

	この場合、画像が切り捨てられることがあります。[縮小] の場合、使用可能なスペースを超えた場合にのみ画像が縮小されます。この値に関係なく、画像は常に中央に配置されます。
タイルの説明	タイルテキストを定義します。
タイルの説明 > 表示行数	行数 (3 から 6) を定義します。
タイルの説明 > 行の折り返し	[はい] の場合、定義された行数の制限まで、各行が完全に (行の折り返し付きで) 表示されます。超過分は切り捨てられます。[いいえ] の場合、各行は切り捨てられます。
タイルの説明 > コンテンツ	タイルテキストコンテンツを定義します。
並べ替え基準	ビュー内のレコードの並べ替え順序を指定します。 データの並べ替え を参照してください。
フィルター	このタイルビューに表示されるレコードのフィルターを定義します。 基準エディター を参照してください。
テーブルの上部のツールバー	このテーブルビューの上で使用するツールバーを定義します。
タイルツールバー	このタイルビューの各項目で使用するツールバーを定義します。

ビューの共有

ビューに対する「ビューの共有」権限を持つユーザーは、「ビュー」メニューからこのビューを表示できるユーザーを定義できます。

これを行うには、ビューの構成画面の「共有」フィールドにプロファイルを追加するだけです。

ビューパブリケーション

「ビューの公開」権限を持つユーザーは、「ビュー」メニューにあるビューを公開できます。

公開されたビューは、Web コンポーネント、ワークフローユーザータスク、データサービス、およびパースペクティブを介してすべてのユーザーが利用できます。ビューを公開するには、「ビュー」メニューに移動し、リストされたビューのマウスオーバーに表示される ボタンをクリックして、「パブリケーション名」をビューに追加します。

ビュー管理

推奨ビューの管理

ユーザーがビューを指定せずにログインすると、推奨されるビュー (存在する場合) が適用されます。それ以外の場合は、デフォルトのビューが適用されます。アクションを使用すると、ユーザーとロールに応じて推奨ビューの割り当てルールを定義できます。

推奨されるビューで使用可能なアクションは、割り当てルールの順序の変更、ルールの追加、既存のルールの編集、既存のルールの削除です。

したがって、特定のユーザーについて、推奨されるビューはユーザーのプロファイルに従って評価されます。適用されたルールは、ユーザーのプロファイルに一致する最初のルールになります。

注意:

機能は、データセットの所有者のみが使用できます。

「ビュー」メニューツールバー

「ビュー」メニューツールバーには、次のアクションがあります。

編集	ターゲットビューのツールバーの [編集] ボタンをクリックして、編集可能なフォームにアクセスします。
複製	ターゲットビューのツールバーの [複製] ボタンをクリックして、ビューを複製します。新しいビュー作成フォームは、複製されるビューからフィールド値を事前入力します。

削除	ターゲットビューのツールバーの [削除] ボタンをクリックして、ビューを削除します。
このビューをお気に入りとして定義	ターゲットビューのツールバーの [このビューをお気に入りとして定義] ボタンをクリックします。テーブルにアクセスすると、お気に入りのビューが自動的に適用されます。ボタンをもう一度クリックして、ユーザーのお気に入りのビューとしてのビューを削除します。

グリッド編集

グリッド編集機能を使用すると、テーブルビューのデータを変更できます。この機能には、 ボタンをクリックしてアクセスできます。

テーブルビューから にアクセスするには、ビュー構成で機能が事前にアクティブ化されている必要があります。

以下も参照してください。:

- [グリッド編集](#)

コピー/貼り付け

同じテーブル内の 1 つまたは複数のセルの別のセルへのコピー/貼り付けは、メニューから実行できます。関連するキーボードショートカット *Ctrl+C* および *Ctrl+V* を使用することもできます。

このシステムは、オペレーティングシステムのクリップボードではなく、内部メカニズムを使用します。結果として、セルを外部ファイルにコピーして貼り付けることはできません。逆に、テーブルセルへの値の貼り付けも機能しません。

ビルトインウィジェットを使用するすべての単純型フィールドがサポートされています。

履歴

履歴機能を使用すると、マスターデータの変更を追跡できます。

履歴機能は、データモデルレベルで事前に有効にしておく必要があります。詳細については、[テーブルの詳細プロパティ](#)を参照してください。

データセットの履歴を表示するには、ナビゲーションペインで > を選択します。

テーブルまたは選択したレコードの履歴を表示するには、ワークスペースで > を選択します。

いくつかの履歴モードが存在し、さまざまなパースペクティブに従って履歴を表示できます。

現在のデータスペースの履歴	テーブル履歴ビューには、現在のブランチの操作が表示されます。これがデフォルトのモードです。
現在のデータスペースと祖先の履歴	テーブル履歴ビューには、現在のブランチとそのすべての祖先に対する操作が表示されます。
現在のデータスペースとマージされた子の履歴	テーブル履歴ビューには、現在のブランチとそのすべてのマージされた子に対する操作が表示されます。
すべてのデータスペースの履歴	テーブル履歴ビューには、ブランチ階層全体の操作が表示されます。

履歴ビューで、メニューを使用して、別の履歴モードに切り替えます。

以下も参照してください。:

- [History](#)

[ユーザーガイド](#) [目次](#)

データの編集

- [ユーザーインターフェイスでのレコードの操作](#)
 - [レコードの作成](#)
 - [既存のレコードの更新](#)

- レコードの複製
 - レコードの削除
 - 2つのレコードの比較
 - AIアシスタントの利用
- データのインポートとエクスポート
 - 履歴から復元

ユーザーインターフェイスでのレコードの操作

レコードの編集は、ユーザーインターフェイスのワークスペース部分で行われます。

注意:

このアクションは、または特別に構成されたパースペクティブの許可されたユーザーのみが使用できます。

レコードの作成

表形式またはタイルビューで、表の上にある  ボタンをクリックします。

階層ビューで、新しいレコードを作成する親ノードのメニューから  を選択します。

次に、新しいレコード作成フォームにフィールド値を入力します。必須フィールドはアスタリスクで示されています。

既存レコードの更新

更新するレコードをダブルクリックしてから、レコードフォームのフィールド値を編集します。

進行中の変更を破棄し、編集する前にフィールドを元の値に戻すには、  ボタンをクリックします。

レコードの複製

選択したレコードを複製するには、  を選択します。

新しいレコード作成フォームは、複製されるレコードからフィールド値を事前入力します。(自動インクリメントされたフィールドの場合のように) 自動的に生成されない限り、主キーには一意の値を指定する必要があります。

レコードの削除

選択した 1 つ以上のレコードを削除するには、> を選択します。

2 つのレコードの比較

選択した 2 つのレコードを比較するには、> を選択します。

注意:

集約リストやユーザー定義属性などの複雑なターミナルノードのコンテンツは、比較プロセスから除外されます。つまり、比較サービスは、レコード内の複雑なターミナルノードの値の違いを無視します。

AIアシスタントの利用

AIアシスタントは、生成AIを活用して、レコードデータとユーザー入力に基づいて提案を生成することで、タスクの完了を支援します。

AIアシスタントを実行するには:

- データモデルで現在のテーブルに対して少なくとも1つの関数が定義されており、それが公開されている必要があります。
- レコードと、少なくとも1つの関数に含まれるすべての入力フィールドおよび出力フィールドを編集する権限が必要です。

AIアシスタントを使用するには:

1. テーブルでレコードを選択します。
2. メニューから、> を選択します。
3. アシスタンス関数を実行します。
 - 必要なアシスタンス関数のタブを選択します。
 - 必要に応じて入力フィールドを編集します。
 - > を選択して提案を生成します。
4. 結果を確認して適用します。
 - 出力フィールドで生成された値を確認します。
 - 必要に応じて、値を変更します。

- 生成された値を保持するには、レコードを保存します。入力フィールドに加えられた変更も保存されることに注意してください。
- 生成された値を破棄するには、保存する前に出力フィールドの変更を破棄します。

注意:

同じフィールドが入力フィールドと出力フィールドの両方として構成されている場合:

- 入力として編集しても出力値は変更されず、出力として編集しても入力値は変更されません。
- レコードを保存すると、出力値のみが保持され、レコードに適用されます。

以下も参照してください。:

- [AIアシスタント](#)

データのインポートとエクスポート

テーブルでは、レコードを CSV または XML 形式でエクスポートまたはインポートできます。

エクスポートするテーブルの特定のレコードを手動で選択するか、テーブル全体をエクスポートすることができます。

以下も参照してください。:

- [CSV サービス](#)
- [XML サービス](#)

履歴から復元

テーブルで履歴を有効にすると、登録された履歴に基づいて、レコードを以前の状態に復元できます。レコード (主キーで識別される) がテーブルにまだ存在する場合は、履歴値で更新されて復元されます。それ以外の場合は作成されます。

レコードを以前の状態に復元するには、履歴テーブルビューでレコードを選択し、ワークスペースで > を選択します。実行する更新または作成の詳細を示す要約画面が表示されます。

復元機能は、一度に 1 つのレコードでのみ使用できます。

テーブルトリガーが通常の作成および更新とは異なり、復元時に特定の動作をする必要がある場合、開発者はメソッド `TableTriggerExecutionContext.isHistoryRestore` を使用できます。

注意:

この機能には、履歴機能の制限に関連する制限があります。

- 「履歴から復元」機能は、履歴でサポートされていないリストを含むテーブルでは使用できません。[データモデルの制限](#)を参照してください。
- 計算値、暗号化された値、および履歴が無効になっているフィールドは、履歴からレコードを復元するときに無視されます。これらのフィールドは履歴がないためです。

以下も参照してください。:

- [履歴](#)

[ユーザーガイド](#) [目次](#)

ファイルのインポートおよびエクスポートサービス

- [XML のインポートとエクスポート](#)
- [CSV のインポートとエクスポート](#)

XML のインポートとエクスポート

- [はじめに](#)
- [インポート](#)
 - [インポートモード](#)
 - [Insert および Update 操作](#)

- 楽観ロック
- エクスポート
- フィールド値の処理
 - 日付、時刻、日時の形式
- 既知の制限事項
 - 関連付けフィールド
 - 選択ノード

はじめに

XML のインポートとエクスポートは、ワークスペースの メニューを使用してユーザーインターフェイスからテーブルに対して実行できます。

インポートとエクスポートはどちらも、データセットのコンテキストで実行されます。

インポートとエクスポートはプログラムで行うこともできます。

デフォルトのインポートおよびエクスポートオプション値は、領域の >>> で設定できます。

インポート

注意:

インポートされた XML ドキュメントは UTF-8 でエンコードされている必要があり、その構造はターゲットデータセットの基になるデータモデルに準拠している必要があります。

インポートモード

XML ファイルをインポートするときは、次のインポートモードのいずれかを指定する必要があります。これにより、インポート手順でソースレコードがどのように処理されるかが決まります。

挿入モード	レコードの作成のみが許可されます。ソースレコードと同じ主キーを持つレコードがターゲットテーブルに存在する場合、エラーが返され、インポート操作全体がキャンセルされます。
-------	---

更新モード	既存のレコードの変更のみが許可されます。ソースレコードと同じ主キーを持つレコードがターゲットテーブルに存在しない場合、エラーが返され、インポート操作全体がキャンセルされます。
更新または挿入モード	ソースレコードと同じ主キーを持つレコードがターゲットテーブルにすでに存在する場合、そのレコードは更新されます。それ以外の場合は、新しいレコードが作成されます。
置換 (同期) モード	ソースレコードと同じ主キーを持つレコードがターゲットテーブルにすでに存在する場合、そのレコードは更新されます。それ以外の場合は、新しいレコードが作成されます。レコードがターゲットテーブルに存在するが、ソース XML ファイルには存在しない場合、そのレコードはテーブルから削除されます。

Insert および Update 操作

「byデルタ」モードを使用すると、ソース XML ドキュメントから欠落しているデータモデルエレメントを無視できます。このモードは、データサービスまたは Java API を介して有効にできます。次の表は、ソースドキュメントにエレメントが存在しない場合の Insert 操作と Update 操作の動作をまとめたものです。

詳細については、Java API のデータサービス操作 [update](#) と [insert](#)、および

`ImportSpec.setByDelta` を参照してください。

ソース XML ドキュメントの状態	動作
ソースドキュメントにエレメントが存在しない	「byデルタ」モードが無効になっている場合 (デフォルト) ターゲットフィールドの値は、次のいずれかに設定されます。

ソース XML ドキュメントの状態	動作
	• エlementがデフォルト値を定義している場合、ターゲットフィールド値はそのデフォルト値に設定されます。 • Elementが文字列またはリスト以外のタイプの場合、ターゲットフィールド値は <code>null</code> に設定されます。 • Elementが集約リストの場合、ターゲットフィールド値は空のリストに設定されます。 • Elementが <code>null</code> と空の文字列を区別する文字列の場合、ターゲットフィールドの値は <code>null</code> に設定されます。2つを区別しない文字列の場合は、空の文字列です。 • Element (単純または複合) がデータサービスで非表示になっている場合、ターゲット値は変更されません。 以下も参照してください。: ◦ データサービスでのフィールドの非表示 注意: インポートを実行するユーザーには、ターゲットフィールド値を作成または変更するために必要な権限が必要です。それ以外の場合、値は変更されません。 「byデルタ」モードがデータサービスまたは Java API を介して有効になっている場合 • <code>Update</code> 操作の場合、フィールド値は変更されません。

ソース XML ドキュメントの状態	動作
	• <code>insert</code> 操作の場合、動作は <code>byDelta</code> モードが無効になっている場合と同じです。
エレメントは存在するが、空 (たとえば、<code><fieldA/></code>)	• タイプ <code>xs:string</code> (またはそのサブタイプの 1 つ) で、空の文字列と <code>null</code> を区別する場合、ターゲットフィールドの値は <code>null</code> に設定されます。それ以外の場合、値は空の文字列に設定されます。 • <code>xs:string</code> タイプ以外のノードの場合、XML スキーマに準拠して例外がスローされます。 以下も参照してください。: • データ型の TIBCO EBX® 空白管理
エレメントが存在し、かつ <code>null</code> (たとえば、<code><fieldA xsi:nil="true"/></code>)	サポートされていないリストを除いて、ターゲットフィールドは常に <code>null</code> に設定されます。 <code>xsi:nil="true"</code> 属性を使用するには、名前空間宣言 <code>xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code> をインポートする必要があります。

欠損している値を null として設定

既存のレコードを更新するときに、XML ファイルにノードがないか空の場合：このオプションが の場合、null と見なされます。このオプションが の場合、変更されません。

エクストラ列を無視する

XML ドキュメントに、ターゲットデータモデルに存在しないエレメントが含まれている場合があります。デフォルトでは、この場合、インポート手順は失敗します。ただし、XML ファイルで定義された余分な列を無視するインポート手順をユーザーが起動できるようにすることは可能で

す。これは、XML のインポートウィザードの構成パラメーターで実行できます。このパラメーターのデフォルト値は、領域の 構成で構成できます。

楽観ロック

技術属性 `ebxd:lastTime` がソース XML ファイルに存在する場合、インポートメカニズムは検証を実行して、最後の読み取り以降に変更された可能性のあるレコードの Update 操作を防止します。`ebxd:lastTime` 属性を使用するには、名前空間宣言 `xmlns:ebxd="urn:ebx-schemas:deployment_1.0"` をインポートする必要があります。現在のレコードはこのタイムスタンプと比較されます。それらが異なる場合、更新は拒否されます。

エクスポート

注意:

エクスポートされた XML ドキュメントは常に UTF-8 でエンコードされます。

XML にエクスポートするときに、テーブルにフィルターが適用されている場合、フィルターに対応するレコードのみがエクスポートされたファイルに含まれます。

XML エクスポートオプションは次のとおりです。

ダウンロードファイル名	エクスポートする XML ファイルの名前を指定します。このフィールドには、レコードのエクスポート元のテーブルの名前が事前に入力されています。
ユーザーフレンドリーモード	エクスポートされた値をユーザーフレンドリーな方法で表示するか、標準の XML 生形式で表示するかを指定します。たとえば、ユーザーフレンドリーモードでは、日付と数値はユーザーのロケールに従ってフォーマットされ、外部キーと列挙値は関連するラベルを表示します。 注意：このオプションを選択すると、エクスポートされたファイルを再インポートできなくなります。

テクニカルデータを含める	内部技術データをエクスポートに含めるかどうかを指定します。 注意： このオプションを選択すると、エクスポートされたファイルを再インポートできなくなります。
インデントされています	人間が読みやすくするためにファイルをインデントするかどうかを指定します。
XML コメントを省略	データの場所とエクスポートの日付を説明する生成された XML コメントをファイルから省略するかどうかを指定します。

フィールド値の処理

日付、時刻、日時の形式

次の日付と時刻の形式がサポートされています。

タイプ	形式	例
xs:date	yyyy-MM-dd	2007-12-31
xs:time	HH:mm:ss または HH:mm:ss.SSS	11:55:00
xs:dateTime	yyyy-MM-ddTHH:mm:ss または yyyy-MM-ddTHH:mm:ss.SSS	2007-12-31T11:55:00

既知の制限事項

関連付けフィールド

XML インポートおよびエクスポートサービスは、関連付け値をサポートしていません。

このようなフィールドをエクスポートしてもエラーは発生しませんが、値はエクスポートされません。

このようなフィールドをインポートするとエラーが発生し、インポート手順は中止されます。

選択ノード

XML インポートおよびエクスポートサービスは、選択値をサポートしていません。

このようなフィールドをエクスポートしてもエラーは発生しませんが、値はエクスポートされません。

このようなフィールドをインポートするとエラーが発生し、インポート手順は中止されます。

CSV のインポートとエクスポート

- はじめに
- エクスポート
- インポート
- フィールド値の処理
 - 集約リスト
 - 非表示フィールド
 - 文字列の「Null」値
 - 日付、時刻、日時の形式
- 既知の制限事項
 - グループの集約リスト
 - ターミナルグループ
 - 列ラベルヘッダー
 - 関連付けフィールド
 - 選択ノード

はじめに

CSV のインポートとエクスポートは、ワークスペースのアクションメニューを使用してユーザーインターフェイスからテーブルで実行できます。

インポートとエクスポートはどちらも、データセットのコンテキストで実行されます。

インポートとエクスポートはプログラムで行うこともできます。

デフォルトのインポートおよびエクスポートオプション値は、領域の >>> で設定できます。

以下も参照してください。:

- [デフォルトのオプション値](#)

エクスポート

CSV にエクスポートするときに、テーブルにフィルターが適用されている場合、フィルターに対応するレコードのみがエクスポートされたファイルに含まれます。

CSV エクスポートオプションは次のとおりです。

ダウンロードファイル名	エクスポートする CSV ファイルの名前を指定します。このフィールドには、レコードのエクスポート元のテーブルの名前が事前に入力されています。
ファイルエンコーディング	エクスポートされたファイルに使用する文字エンコードを指定します。デフォルトは UTF-8 です。
継承を有効化	CSV エクスポート中に 継承 を考慮するには、モデルでオプションを定義する必要があります。 継承の詳細については、継承と値の解決 を参照してください。 CSV エクスポート中に継承を考慮するかどうかを指定します。 継承が有効になっている場合、フィールドの解決された値は、レコードまたはフィールドの可能な継承モードを定義する技術データとともにエクスポートされます。 継承が無効になっている場合、フィールドの解決された値がエクスポートされ、非表示にされたレコードは無視されます。

	デフォルトでは、このオプションは無効になっています。 注意：テーブルデータセットに親がない場合、またはテーブルに継承フィールドがない場合、継承は常は無視されます。
ユーザーフレンドリーモード	エクスポートされた値をユーザーフレンドリーな方法で表示するか、生の形式で表示するかを指定します。たとえば、ユーザーフレンドリーモードでは、日付と数値はユーザーのロケールに従ってフォーマット設定され、外部キーと列挙値は関連するラベルを表示します。 注意：このオプションを選択すると、エクスポートされたファイルを再インポートできなくなります。
テクニカルデータを含める	内部技術データをエクスポートに含めるかどうかを指定します。 注意：このオプションを選択すると、エクスポートされたファイルを再インポートできなくなります。
列ヘッダー	CSV ファイルに列ヘッダーを含めるかどうかを指定します。 • ヘッダーなし • ラベル：スプレッドシートの各列について、CSV にラベルが表示されます。各ラベルは、現在のセッションのロケール設定に従ってローカライズされます。ノードに分かりやすいラベルが定義されていない場合は、ノードの技術名称が使用されます。

	• XPath：スプレッドシートの各列について、CSV はテーブル内のノードへのパスを表示します。
フィールド区切り文字	エクスポートに使用するフィールド区切り文字を指定します。デフォルトの区切り文字はカンマです。 [管理] > [ユーザーインターフェイス] で変更できます。
リスト区切り文字	値リストに使用する区切り文字を指定します。デフォルトの区切り文字は改行です。 [管理] > [ユーザーインターフェイス] で変更できます。

プログラムによる CSV エクスポートは、Java API の `ExportSpec` および `ExportImportCSVSpec` クラスを使って行います。

インポート

ダウンロードファイル名	インポートする CSV ファイルの名前を指定します。
インポートモード	CSV ファイルをインポートするときは、次のインポートモードのいずれかを指定する必要があります。これにより、ソーステーブルとターゲットテーブルの間の操作の整合性が制御されます。 • 挿入モード：レコードの作成のみが許可されます。ソースレコードと同じ主キーを持つレコードがターゲットテーブルに存在する場合、エラーが返され、インポート操作全体がキャンセルされます。 • 更新モード：既存のレコードの変更のみが許可されます。ソースレコードと同じ主キーを持つレコードがターゲットテーブルに存在しない場合、エラーが返さ

	れ、インポート操作全体がキャンセルされます。 • 更新または挿入モード：ソースレコードと同じ主キーを持つレコードがターゲットテーブルにすでに存在する場合、そのレコードは更新されます。それ以外の場合は、新しいレコードが作成されます。 • 置換 (同期) モード：ソースレコードと同じ主キーを持つレコードがターゲットテーブルにすでに存在する場合、そのレコードは更新されます。それ以外の場合は、新しいレコードが作成されます。レコードがターゲットテーブルに存在するが、ソースXMLファイルには存在しない場合、そのレコードはテーブルから削除されます。
ファイルエンコーディング	エクスポートされたファイルに使用する文字エンコードを指定します。デフォルトは UTF-8 です。
継承を有効化	CSV インポート中に 継承 を考慮するには、モデルでオプションを定義する必要があります。 継承の詳細については、継承と値の解決 および <code>ExportImportCSVSpec.setInheritanceEnabled</code> を参照してください。 CSV インポート中に継承を考慮するかどうかを指定します。CSV ファイルの技術データで継承モードが定義されている場合、対応するフィールドまたはレコードは強制的に継承されます。技術データが非表示モードを定義している場合、対応するレコードは強制的に非表示にされます。それ以外の場合、フィールドは CSV ファイルから読み取

	られた値で上書きされます。デフォルトでは、このオプションは無効になっています。 注意：テーブルのデータセットに親がない場合、またはテーブルに継承されたフィールドがない場合、継承は常に無視されます。
列ヘッダー	CSV ファイルに列ヘッダーを含めるかどうかを指定します。 • ヘッダーなし • ラベル：スプレッドシートの各列について、CSV にラベルが表示されます。各ラベルは、現在のセッションのロケール設定に従ってローカライズされます。ノードに分かりやすいラベルが定義されていない場合は、ノードの技術名称が使用されます。 • XPath：スプレッドシートの各列について、CSV はテーブル内のノードへのパスを表示します。
フィールド区切り文字	エクスポートに使用するフィールド区切り文字を指定します。デフォルトの区切り文字はカンマです。[管理] > [ユーザーインターフェイス] で変更できます。
リスト区切り文字	値リストに使用する区切り文字を指定します。デフォルトの区切り文字は改行です。[管理] > [ユーザーインターフェイス] で変更できます。

プログラムによる CSV インポートは、Java API の `ImportSpec` および `ExportImportCSVSpec` クラスを使って行います。

フィールド値の処理

集約リスト

CSV インポートおよびエクスポートサービスは、複数值フィールド、つまり集約リストをサポートします。これは、`string`、`date`、または `int` のリストなどの単純な型付きリスト、および外部キーでのみサポートされます。テーブル参照が複合主キーにリンクされている場合、リスト内の各項目は、「true|99」のようにフォーマット設定された文字列です。グループの集約リストはエクスポートされません。

エクスポート時には、リスト内の項目は行区切り記号を使用して区切られます。`osd:html` や `osd:text` など、エクスポートされたフィールドにすでに行区切り文字が含まれている場合、コード `_crnl_` は次のようになります。フィールド値の行区切り文字の代わりに挿入されます。インポート時に同じフォーマットが期待され、フィールド値全体が引用符で囲まれます。

非表示フィールド

非表示フィールドは、`ebx-csv:hidden` 文字列としてエクスポートされます。インポートされた非表示の文字列は、フィールドのコンテンツを変更しません。

文字列の「Null」値

CSV インポートおよびエクスポートサービスを使用して、値が `null` に設定された文字列が空の文字列としてエクスポートされます。したがって、ラウンドトリップのエクスポート/インポート手順では、`null` 文字列値が空の文字列に置き換えられます。

プログラムサービスを使用すると、特定の値 `ebx-csv:nil` を、値が `null` に設定された文字列に割り当てることができます。これを行うと、ラウンドトリップのエクスポート/インポートプロセス中に、`null` 文字列値が空の文字列に置き換えられることはありません。詳細については、Java API の `ExportImportCSVSpec.setNullStringEncoded` を参照してください。

日付、時刻、日時の形式

次の日付と時刻の形式がサポートされています。

タイプ	形式	例
xs:date	yyyy-MM-dd	2007-12-31
xs:time	HH:mm:ss または HH:mm:ss.SSS	11:55:00

タイプ	形式	例
xs:dateTime	yyyy-MM-ddTHH:mm:ss また は yyyy-MM- ddTHH:mm:ss.SSS	2007-12-31T11:55:00

既知の制限事項

グループの集約リスト

CSV インポートおよびエクスポートサービスは、複数値グループ、つまり複合型エレメントの集約リストをサポートしていません。このようなノードをエクスポートしてもエラーは発生しませんが、値はエクスポートされません。

ターミナルグループ

CSV ファイルでは、空のフィールドのみを含む作成済みのターミナルグループと、作成されていないフィールドを区別することはできません。

結果として、エクスポートに続いてインポートを実行した後の比較中に、いくつかの違いが表示される場合があります。インポートとエクスポートの対称性を確保するには、代わりに XML のインポートとエクスポートを使用してください。[XML のインポートとエクスポート](#)を参照してください。

列ラベルヘッダー

2 つの列が同じラベルヘッダーを共有している場合、テーブルのエクスポートは正常に実行できませんが、エクスポートされたデータを後で再インポートすることはできません。

関連付けフィールド

CSV インポートおよびエクスポートサービスは、関連付け値、つまり関連付けられたレコードをサポートしていません。

このようなフィールドをエクスポートしてもエラーは発生しませんが、値はエクスポートされません。

このようなフィールドをインポートするとエラーが発生し、インポート手順が中止されます。

選択ノード

CSV インポートおよびエクスポートサービスは、選択値、つまり選択されたレコードをサポートしていません。

このようなフィールドをエクスポートしてもエラーは発生しませんが、値はエクスポートされません。

このようなフィールドをインポートするとエラーが発生し、インポート手順が中止されます。

既存のデータセットの操作

- データセットの検証
- データセットの複製
- データセットの非アクティブ化
- データセットの権限の管理

データセットの検証

どの時点であっても、データセットを検証するには、ナビゲーションペインから **>** を選択します。生成されたレポートは、検証の結果を提供します。検証レポートから、ボタンをクリックして報告された検証ステータスを更新するか、ボタンをクリックしてデータセットに関連付けられている現在の検証レポートをクリアし、最初から完全な検証を再実行できるようにするオプションがあります。

データの検証は、ナビゲーションペインから目的のテーブルに移動し、ワークスペースから **>** を選択することにより、テーブルレベルで実行することもできます。

増分データ検証の詳細については、[検証](#) を参照してください。

データセットの複製

既存のデータセットを複製するには、ナビゲーションペインの [userdata_intro.html#ui](#)   ニューからデータセットを選択し、**>** を選択します。

データセットの非アクティブ化

データセットがアクティブ化されると、検証の対象になります。つまり、データセットを有効にするには、すべての必須エレメントを定義する必要があります。データセットがアクティブで検証されている場合は、外部システムに安全にエクスポートしたり、他の Java アプリケーションで使用したりできます。

データセットに必須エレメントがない場合は、**>** からプロパティ を に設定することで非アクティブ化できます。

データセットのアクセス許可の管理

データセットの権限にアクセスするには、ナビゲーションペインで > を選択します。

権限は、プロファイルレコードを使用して定義されます。新しい権限プロファイルを定義するには、テーブルに新しいレコードを作成します。

以下も参照してください。:

- [プロファイル](#)

プロファイル	これらの権限が適用されるプロファイルを定義します。
制限ポリシー	[はい] の場合、ここで定義された権限が他の方法で定義されたものよりも厳密である場合、これらの権限が尊重されることを示します。これは、定義された最も許容度の高い権限が優先されるデフォルトとは対照的です。 ユーザー定義ルールの解決 を参照してください。
データセットアクション	データセットに対するアクションの権限を指定します。
子データセットの作成	プロファイルが子データセットを作成できるかどうかを示します。継承もデータモデルでアクティブ化する必要があります。
データセットの複製	プロファイルがデータセットを複製できるかどうかを示します。
データセットの削除	プロファイルがデータセットを削除できるかどうかを示します。

データセットのアクティブ化/非アクティブ化	プロファイルがデータセット情報の <i>[Activated]</i> プロパティを変更できるかどうかを示します。データセットの非アクティブ化を参照してください。
ビューの作成	プロファイルがデータセットにビューと階層を作成できるかどうかを示します。
テーブルポリシー	すべてのテーブルのデフォルトの権限を指定します。[+] ボタンをクリックして、テーブルに特定の権限を定義することもできます。
新しいレコードの作成	プロファイルがテーブルにレコードを作成できるかどうかを示します。
継承されたレコードの上書き	プロファイルがテーブル内の継承されたレコードをオーバーライドできるかどうかを示します。この権限は、データセットの継承を使用する場合に役立ちます。
継承レコードの非表示	プロファイルがテーブル内の継承されたレコードを非表示にできるかどうかを示します。この権限は、データセットの継承を使用する場合に役立ちます。
レコードの削除	プロファイルがテーブル内のレコードを削除できるかどうかを示します。
値のアクセスポリシー	データセットのすべてのノードのデフォルトのアクセス許可を指定し、特定のノードの許可を定義できるようにします。ノードにカスタム権限が定義されていない場合は、デフォルトのアクセス権限が使用されます。

	特定のポリシーセクターを使用すると、ノードに特定のアクセス許可を付与できます。リンク「ReadOnly」、 「ReadWrite」、および「Hidden」は、選択したノードに対応するアクセスレベルを設定します。 「(デフォルト)」リンクを使用して、カスタムアクセス許可を削除することができます。
サービスの権限	このセクションでは、サービスへのアクセス許可を指定します。取り消し線が引かれている場合、プロファイルはサービスにアクセスできません。

ユーザーガイド [目次](#)

データセットの継承

データセットの継承の概念を使用して、親データセットから分岐する子データセットを作成することができます。子データセットは、デフォルトで親データセットから値とプロパティを継承し、必要に応じてオーバーライドできます。複数のレベルの継承が存在する可能性があります。

データセットの継承を使用する例は、親データセットでグローバルなデフォルト値を定義し、それらのデフォルト値をオーバーライドできる特定の地理的領域の子データセットを作成することです。

注意:

デフォルトでは、データセットの継承は無効になっています。基になるデータモデルで明示的にアクティブ化する必要があります。

以下も参照してください。:

- [データモデルの構成](#)

- データセットの継承構造
- 値の継承
 - レコードの継承

データセットの継承構造

ルートデータセットが作成されたら、ナビゲーションペインのデータセットセレクターの  ボタンを使用して、ルートデータセットから子データセットを作成します。

注意:

- 子データセットがある場合、データセットを削除することはできません。子データセットを最初に削除する必要があります。
- 子データセットが複製された場合、新しく作成されたデータセットは、複製されたデータセットの兄弟として既存のデータセットツリーに挿入されます。

値の継承

子データセットが作成されると、そのすべてのフィールド値が親データセットから継承されます。レコードは、デフォルトの継承値を保持するか、オーバーライドすることができます。

表形式のビューでは、継承された値はセルの左上隅にマークされます。

 ボタンを使用して、値を上書きできます。

レコードの継承

子データセットのテーブルは、その祖先データセットのテーブルからレコードを継承します。子データセットのテーブルは、レコードを追加、変更、または削除できます。レコードのタイプを区別するために、いくつかの状態が定義されています。

ルート	ルートレコードは、現在のデータセットで作成されたレコードであり、親データセットには存在しません。ルートレコードは、現在のデータセットの子データセットに継承されます。
-----	--

継承済み	継承されたレコードは、現在のデータセットの祖先データセットで定義されているレコードです。
上書き	上書きされたレコードは、現在のデータセットで値が変更された継承されたレコードです。上書きされた値は、現在のデータセットの子データセットに継承されます。
非表示済み	非表示化されたレコードは、現在のデータセットで削除された継承されたレコードです。取り消し線が引かれたレコードとして現在のデータセットに引き続き表示されますが、現在のデータセットの子データセットには継承されません。

ボタン  がオンになっている場合、レコードまたは値が親データセットから継承されていることを示します。このボタンをオフに切り替えて、レコードまたは値を上書きできます。非表示化されたレコードの場合は、ボタンをオンに切り替えて、[非表示化](#)に戻します。

次の表は、レコードを作成、変更、または削除するときの、初期状態に応じたレコードの動作をまとめたものです。

状態	作成	値の変更	削除
ルート	標準の新しいレコードの作成。新しく作成されたレコードは、現在のデータセットの子データセットに継承されます。	既存のレコードの標準的な変更。変更された値は、現在のデータセットの子データセットに継承されます。	標準レコードの削除。レコードは、現在のデータセットと現在のデータセットの子データセットに表示されなくなります。
継承	既存の継承されたレコードと同じ主キーを使用してレコードが作成された場合、そのレコードは上書きされ、その値は作	継承されたレコードは、その値を変更するために、最初に上書きとしてマークする必要があります。	継承されたレコードを削除すると、その状態が非表示済みに変わります。

状態	作成	値の変更	削除
	成時に送信されたものになります。		
上書き	適用できない。現在のデータセットで主キーがすでに使用されている場合、新しいレコードを作成できません。	オーバーライドされたレコードは継承された状態に戻すことができますが、変更された値は失われます。上書きされたレコードの個々の値は、継承に設定することも、変更することもできます。	上書きされたレコードを削除すると、その状態が非表示になります。
非表示済み	既存の非表示済みレコードの主キーを使用してレコードが作成された場合、レコードの状態は上書きに変更され、作成時に送信されたものに従ってその値が変更されます。	適用不可。非表示にしたレコードは変更できません。	適用不可。非表示にされたレコードはすでに削除されたと見なされます。

以下も参照してください。:

- [レコードルックアップメカニズム](#)

[ユーザーガイド 目次](#)

コラボレーション機能 (チームアップ)

- [コラボレーション機能 \(チームアップ\) の概要](#)
- [コメント](#)
- [評価](#)
- [タグ](#)

コラボレーション機能 (チームアップ) の概要

- 概要
- チームアップ機能へのアクセス

概要

チームアップを使用して、他のユーザーとのコラボレーションと連携を行います。レコードへのコメントの追加、レコードの評価、レコードへのタグの割り当てが行えます。これらのインタラクションにより、データの全体的な理解を深めることができます。

チームアップ機能の概要は、以下のとおりです。

- コメント：レコードへのコメントの追加と返信、他のコメントへのリアクションの追加を行います。テーブルにモデレーターを割り当てて、他のユーザーからのコメントを管理できます。
- タグ：レコードにタグを適用します。タグは、タグカタログに分類されます。管理者を割り当てて、タグカタログを管理します。
- 評価：各レコードに星 1 つから星 5 つまでの評価を残します。評価を残すときは、オプションでそれを説明するメモを追加します。

チームアップ機能へのアクセス

レコードで利用可能なチームアップ機能にアクセスするには、次の手順を実行します。

1. レコードの詳細ページを開きます。
2. 右側のサイドバーで、使用するチームアップ機能に対応するアイコンをクリックします。

既知の制限事項: カスタムユーザーサービスがメインワークスペースで開かれている場合、チームアップ機能にはアクセスできません。

Master Data - Reference ▾ Author [lastNameAuthor6 firstNameAuthor6 [00-0-006]]

Tutorial ▾

Actions ▾

Titles

Publishers

▾ Authors

lastNameAuthor6 firstNameAuthor6

Royalties

Contracts

BookRental

Authors EditorWithContracts

Id 00-0-006

Lastname * lastNameAuthor6

Firstname * firstNameAuthor6

Address addressAuthor6

Birth date / /

Death date / /

City cityAuthor6

Country France

Books + Actions ▾ No records ▾

Id	Title	Type	Creation date	Publisher	Price	Sales
No records found.						

Save Save and close Revert Close

以下も参照してください。:

- [コメント](#)
- [評価](#)
- [タグ](#)

[ユーザーガイド 目次](#)

コメント

- [概要](#)
- [コメントの追加](#)
- [コメントの編集](#)
- [コメントの削除](#)
- [コメントへの返信](#)
- [コメントへのリアクション](#)
- [コメントのモデレート](#)

概要

チームアップのコメント機能を使用して、レコードの詳細について議論できます。自分のコメントを追加したり、他のユーザーからのコメントに返信したり、リアクションを追加したりできます。

The screenshot displays the TIBCO EBX 6.2.3 user interface. On the left, a sidebar contains a navigation menu with options like 'Tutorial', 'Actions', 'Titles', 'Publishers', 'Authors' (selected), 'Royalties', 'Contracts', and 'BookRental'. The main area shows the 'Author' record for 'lastNameAuthor1 firstNameAuthor1 [00-0-001]'. The 'Authors' tab is active, showing fields for Id, Lastname, Firstname, Address, Birth date, Death date, City, and Country. Below these fields is a table for 'Books' with columns for Id, Title, Type, Creation date, and Publisher. The 'Comments' sidebar on the right shows a list of comments from 'User1 User1', 'User2 User2', and 'User3 User3'. Each comment includes a timestamp, the text of the comment, and reaction icons (thumbs up, thumbs down, and a speech bubble). At the bottom of the sidebar, there is a text input field labeled 'Write your comment' and a send button.

コメントの追加

サイドバーの下部にあるテキスト入力を使用してコメントを追加します。

Master Data - Reference ▾ Author [lastNameAuthor1 firstNameAuthor1 [00-0-001]]

Tutorial ▾

Actions ▾

Titles

Publishers

▼ Authors

lastNameAuthor1 firstNameAuthor1 [00-0-001]

Royalties

Contracts

BookRental

Authors EditorWithContracts

Id 00-0-001

LastName lastNameAuthor1

FirstName firstNameAuthor1

Address addressAuthor1

Birth date / /

Death date / /

City cityAuthor1

Country France

Books + Actions No records

Id Title Type Creation date Publisher

No records found.

Save Save and close Revert Close

Comments

User1 User1 10/19, 17:54

Lorem ipsum dolor sit amet.

1 0 0

1 reply to display ▴

User2 User2 10/19, 18:07

Consectetur adipiscing elit. (moderated)

Moderated by User3 User3 10/19, 18:14

Dapibus eu interdum vitae.

0 0 1

User3 User3 10/19, 18:15

Nam et mi eu leo mollis aliquet. Duis at q uam.

0 1 0

Write your comment

コメントの編集

ツールバーの編集ボタンをクリックして、コメントを編集します (下図で強調表示)。モデレーターが編集したコメントは編集できないことに注意してください。

コメントの管理の詳細については、[コメントの管理](#)を参照してください。

Master Data - Reference ▾ Author [lastNameAuthor1 firstNameAuthor1 [00-0-001]]

Tutorial ▾

Actions ▾

Titles

Publishers

▼ Authors

lastNameAuthor1 firstNameAuthor1 [00-0-001]

Royalties

Contracts

BookRental

Authors EditorWithContracts

Id 00-0-001

LastName lastNameAuthor1

FirstName firstNameAuthor1

Address addressAuthor1

Birth date / /

Death date / /

City cityAuthor1

Country France

Books + Actions No records

Id Title Type Creation date Publisher

No records found.

Save Save and close Revert Close

Comments

User1 User1 10/19, 17:54

Lorem ipsum dolor sit amet.

1 0 0

1 reply to display ▾

User3 User3 10/19, 18:15

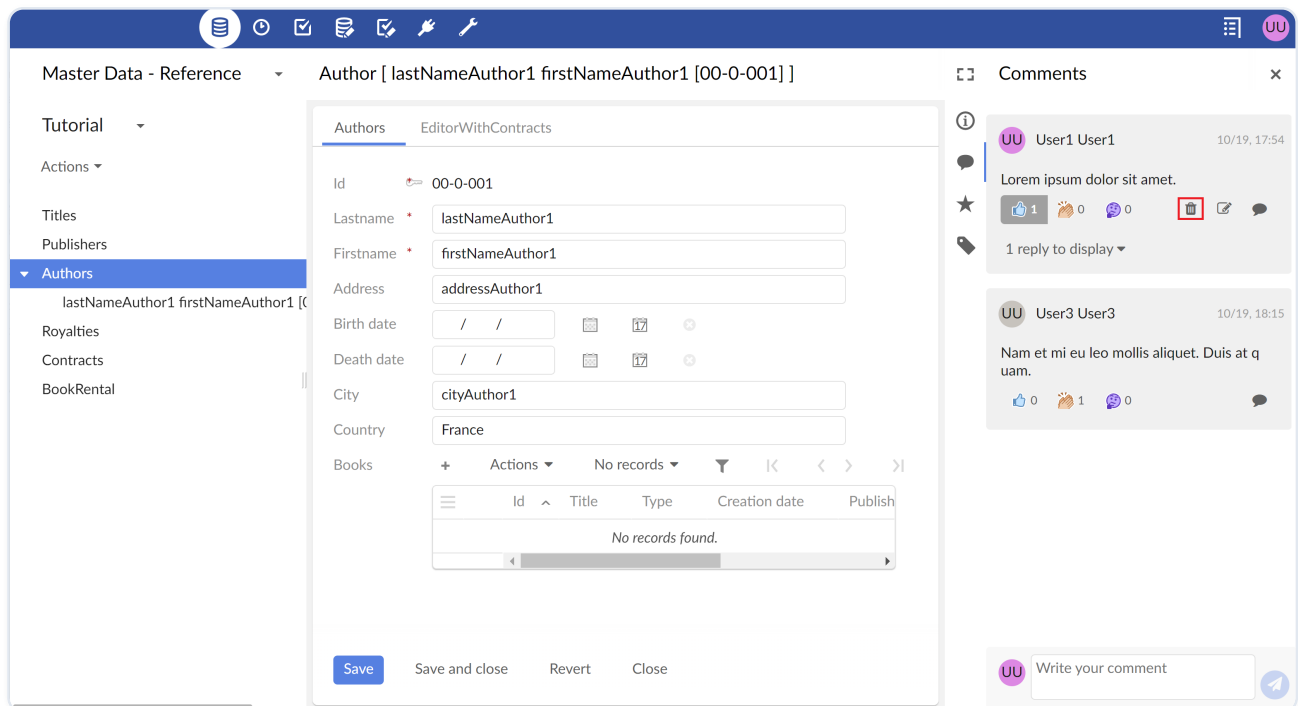
Nam et mi eu leo mollis aliquet. Duis at q uam.

0 1 0

Write your comment

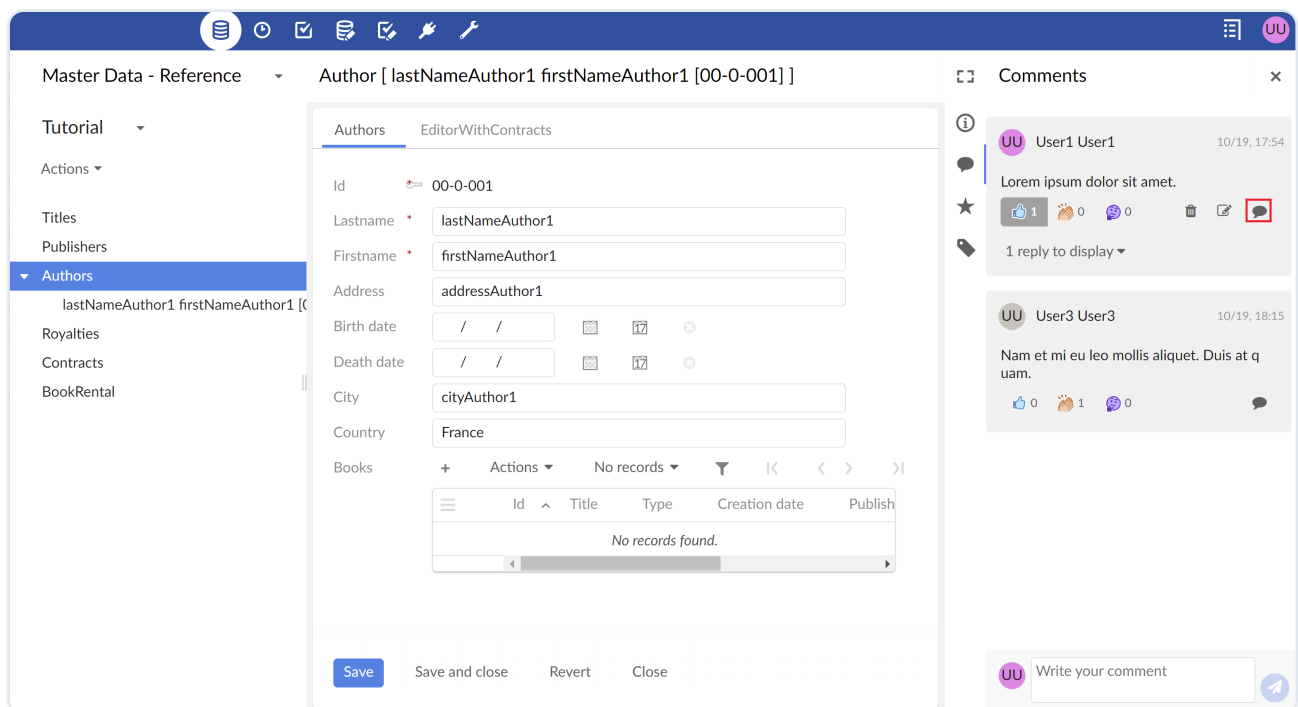
コメントの削除

コメント下部のツールバーの ボタンをクリックして、コメントの 1 つを削除します。



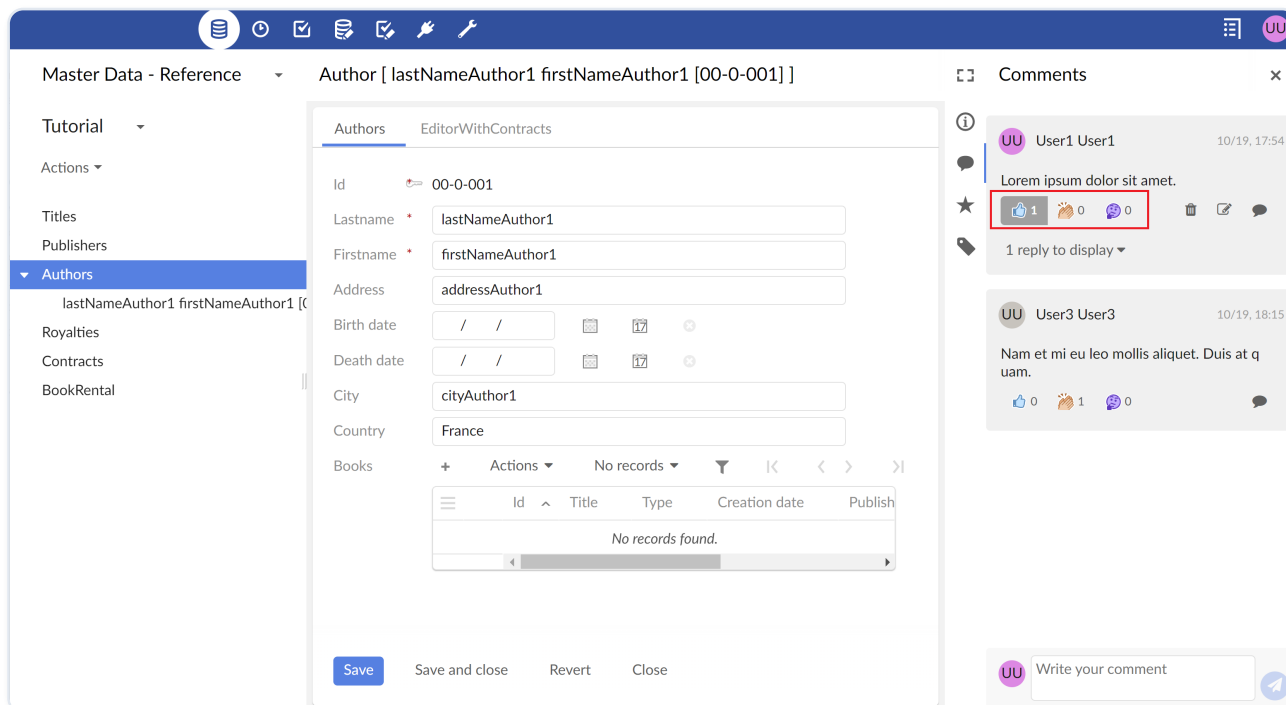
コメントへの返信

コメント下部のツールバーの ボタンを使用して、コメントに返信します。



コメントへのリアクション

コメント下部のツールバーで利用可能な ボタンを使用して、コメントへのリアクションを追加します。別のリアクションをクリックして、リアクションを変更します。同じリアクションをもう一度クリックして、リアクションを削除します。



コメントのモデレート

下図で強調表示されている アイコンをクリックして、他のユーザーのコメントをモデレートおよび編集します。他のユーザーのコメントを編集するときに、元のコンテンツがモデレートされた理由を示すメモを残すことができます。

Master Data - Reference ▾ Author [lastNameAuthor1 firstNameAuthor1 [00-0-001]]

Tutorial ▾

Actions ▾

Titles

Publishers

▾ Authors

lastNameAuthor1 firstNameAuthor1

Royalties

Contracts

BookRental

Authors EditorWithContracts

Id 00-0-001

Lastname * lastNameAuthor1

Firstname * firstNameAuthor1

Address addressAuthor1

Birth date / /

Death date / /

City cityAuthor1

Country France

Books + Actions ▾ No records ▾

Id	Title	Type	Creation date	Publisher
No records found.				

Save Save and close Revert Close

Comments

UU User1 User1 10/19, 17:54

Lorem ipsum dolor sit amet.

1 0 0

1 reply to display ▾

UU User3 User3 10/19, 18:15

Nam et mi eu leo mollis aliquet. Duis at q uam.

0 1 0

Write your comment

モデレーターの管理方法の詳細については、[ユーザーアクセスとチームアップ機能の管理](#)を参照してください。

以下も参照してください。:

- [評価](#)
- [タグ](#)

[ユーザーガイド 目次](#)

評価

- [概要](#)
- [レコードの評価と評価の編集](#)
- [評価の削除](#)

概要

最大 5 つの星を残すことでレコードに評価を付け、オプションで評価の説明を追加します。サイドバーには以下が表示されます。

- [レコードの総合評価](#)
- [他のユーザーによって提供された個々の評価](#)

• それらの評価に関連するコメント

The screenshot shows the 'Authors' form in the TIBCO EBX interface. The form is titled 'Author [lastNameAuthor1 firstNameAuthor1 [00-0-001]]'. It contains fields for 'Id' (00-0-001), 'Lastname' (lastNameAuthor1), 'Firstname' (firstNameAuthor1), 'Address' (addressAuthor1), 'Birth date', 'Death date', 'City' (cityAuthor1), and 'Country' (France). Below these fields is a table for 'Books' with columns 'Id', 'Title', 'Type', 'Creation date', and 'Publisher'. The table is currently empty, showing 'No records found.' At the bottom of the form are buttons for 'Save', 'Save and close', 'Revert', and 'Close'.

On the right side, there is a 'Ratings' panel. It shows a 4.0 rating with 2 ratings. The panel includes a star icon, a list of users (User2 User2 and User1 User1) with their respective star ratings and timestamps, and a section for 'Your rating' with a star icon and a text input field labeled 'Explain your rating'.

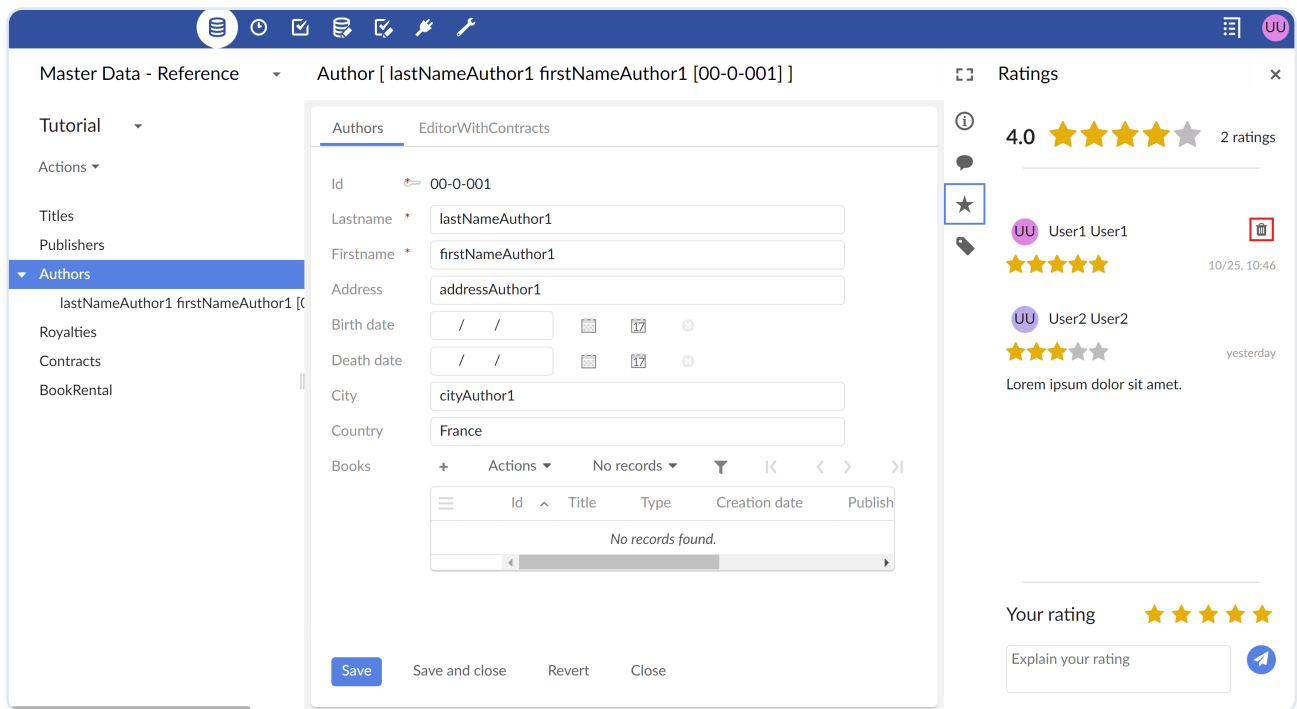
レコードの評価と評価の編集

サイドバー下部の入力機能を使用して評価を追加し、必要に応じてコメントを残します。同じアクションを実行して、評価を編集します。

This screenshot is similar to the one above, showing the 'Authors' form and the 'Ratings' panel. However, a red box highlights the 'Your rating' section at the bottom of the ratings panel, which includes a star icon and a text input field labeled 'Explain your rating'. The 'Ratings' panel also shows a 4.0 rating with 2 ratings, and a list of users (User2 User2 and User1 User1) with their respective star ratings and timestamps.

評価の削除

評価の右側にある アイコンを使用して、評価を削除します。各自の評価は常に個々の評価リストの最上部に表示されることに注意してください。



以下も参照してください。:

- [コメント](#)
- [タグ](#)

[ユーザーガイド 目次](#)

タグ

- [概要](#)
- [タグの適用](#)
 - [既存のタグの適用](#)
 - [新しいタグの適用](#)
- [タグの削除](#)

概要

サイドバーを使用して、1 つまたは複数のタグをレコードに割り当てます。

The screenshot displays the TIBCO EBX 6.2.3 user interface. The main window is titled 'Master Data - Reference' and shows the 'Author [lastNameAuthor1 firstNameAuthor1 [00-0-001]]' record. The left sidebar contains a navigation menu with 'Authors' selected. The central area shows the 'Authors' form with fields for Id (00-0-001), Lastname (lastNameAuthor1), Firstname (firstNameAuthor1), Address (addressAuthor1), Birth date, Death date, City (cityAuthor1), and Country (France). Below these fields is a 'Books' section with a table showing 'No records found.' The right sidebar, titled 'Tags', shows 'Applied tags' (Tag A, Tag H), 'Apply tag' (Search or create tag), 'Last used' (Tag B), 'Most used' (Tag B, Tag C), and 'My tags' (Tag B, Tag C). At the bottom of the main window are buttons for 'Save', 'Save and close', 'Revert', and 'Close'.

タグの適用

既存のタグの適用

、、 セクションからタグをクリックして適用します。

Master Data - Reference ▾ Author [lastNameAuthor1 firstNameAuthor1 [00-0-001]]

Tutorial ▾

Actions ▾

Titles

Publishers

▾ Authors

lastNameAuthor1 firstNameAuthor1

Royalties

Contracts

BookRental

Authors EditorWithContracts

Id 00-0-001

LastName lastNameAuthor1

Firstname firstNameAuthor1

Address addressAuthor1

Birth date / /

Death date / /

City cityAuthor1

Country France

Books + Actions No records

Id Title Type Creation date Publisher

No records found.

Save Save and close Revert Close

Tags

Applied tags

Tag A Tag H

Apply tag

Search or create tag

Last used click to apply

Tag B Tag H

Most used click to apply

Tag B Tag C

My tags click to apply

Tag B Tag C

必要なタグが、、セクションの下に表示されない場合は、入力フィールドを使用して検索します。

Master Data - Reference ▾ Author [lastNameAuthor1 firstNameAuthor1 [00-0-001]]

Tutorial ▾

Actions ▾

Titles

Publishers

▾ Authors

lastNameAuthor1 firstNameAuthor1

Royalties

Contracts

BookRental

Authors EditorWithContracts

Id 00-0-001

LastName lastNameAuthor1

Firstname firstNameAuthor1

Address addressAuthor1

Birth date / /

Death date / /

City cityAuthor1

Country France

Books + Actions No records

Id Title Type Creation date Publisher

No records found.

Save Save and close Revert Close

Tags

Applied tags

Tag A

Apply tag

Tag

Tag B Tag C

Most used click to apply

Tag B Tag C

My tags click to apply

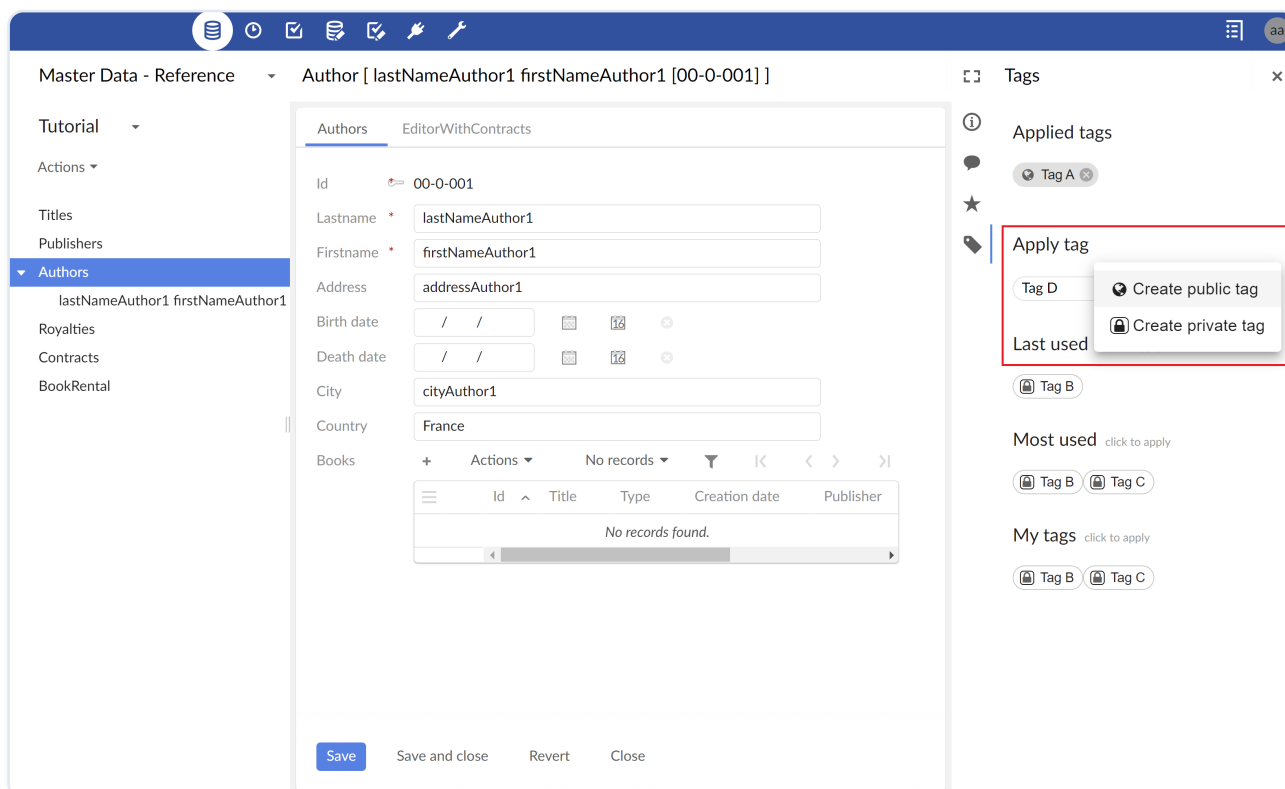
Tag B Tag C

タグカタログの管理の詳細については、[タグとタグカタログの管理](#)を参照してください。

新しいタグの適用

必要なタグがテーブルのタグカタログに存在しない場合は、その名前を入力フィールドに入力して Enter キーを押すか、 ボタンをクリックしてタグを作成します。

テーブルのタグカタログの管理者は、新しいタグを作成するときに、タグを公開するか非公開にするかを選択できます。すべてのプロファイルが公開タグを表示でき、作成者のみが非公開タグを表示できます。



タグカタログ管理者の管理の詳細については、[タグとタグカタログの管理](#)を参照してください。

タグの削除

セクションの下にタグに表示される ボタンを使用して、レコードに割り当てられたタグを削除します。

The screenshot displays the TIBCO EBX 6.2.3 user interface. On the left, a sidebar contains navigation links: Master Data - Reference, Tutorial, Actions, Titles, Publishers, **Authors** (selected), lastNameAuthor1 firstNameAuthor1 [00-0-001], Royalties, Contracts, and BookRental. The main area shows the 'Authors' form for the author 'lastNameAuthor1 firstNameAuthor1 [00-0-001]'. The form includes fields for Id (00-0-001), Lastname, Firstname, Address, Birth date, Death date, City, and Country (France). Below these fields is a 'Books' section with a table header (Id, Title, Type, Creation date, Publisher) and a message 'No records found.' At the bottom of the form are buttons: Save, Save and close, Revert, and Close. On the right, a 'Tags' sidebar is open, showing 'Applied tags' (Tag A, Tag H), 'Apply tag' (Search or create tag), 'Last used' (Tag B), 'Most used' (Tag B, Tag C), and 'My tags' (Tag B, Tag C).

以下も参照してください。:

- コメント
- 評価

ユーザーガイド [目次](#)

ワークフローモデル

- ワークフローモデルの概要
- ワークフローモデルの作成と実装
- ワークフローモデルの構成
- ワークフローモデルの公開

ワークフローモデルの概要

- 概要
 - ワークフローモデルとは
 - ワークフローモデルに関連する基本概念

- ワークフローモデル領域のユーザーインターフェイスの使用
- 汎用メッセージテンプレート
 - 例
- ワークフローの制限

概要

ワークフローモデルとは

TIBCO EBX® のワークフローは、リポジトリ内のデータの共同管理を容易にします。ワークフローには、特定のイベントの通知をサポートしながら、データに対する人間のアクションと自動化されたタスクを同様に含めることができます。

ワークフローを実現するための最初のステップは、ステップの進行、ユーザーの責任、およびワークフローに関連するその他の動作を定義する **ワークフローモデル** を作成することです。

ワークフローモデルが定義されると、それを検証して **ワークフローパブリケーション** として公開できます。次に、ワークフローパブリケーションからデータワークフローを起動して、ワークフローモデルで定義されたステップを実行できます。

以下も参照してください。:

- [ワークフローモデル \(用語集\)](#)
- [データワークフロー \(用語集\)](#)

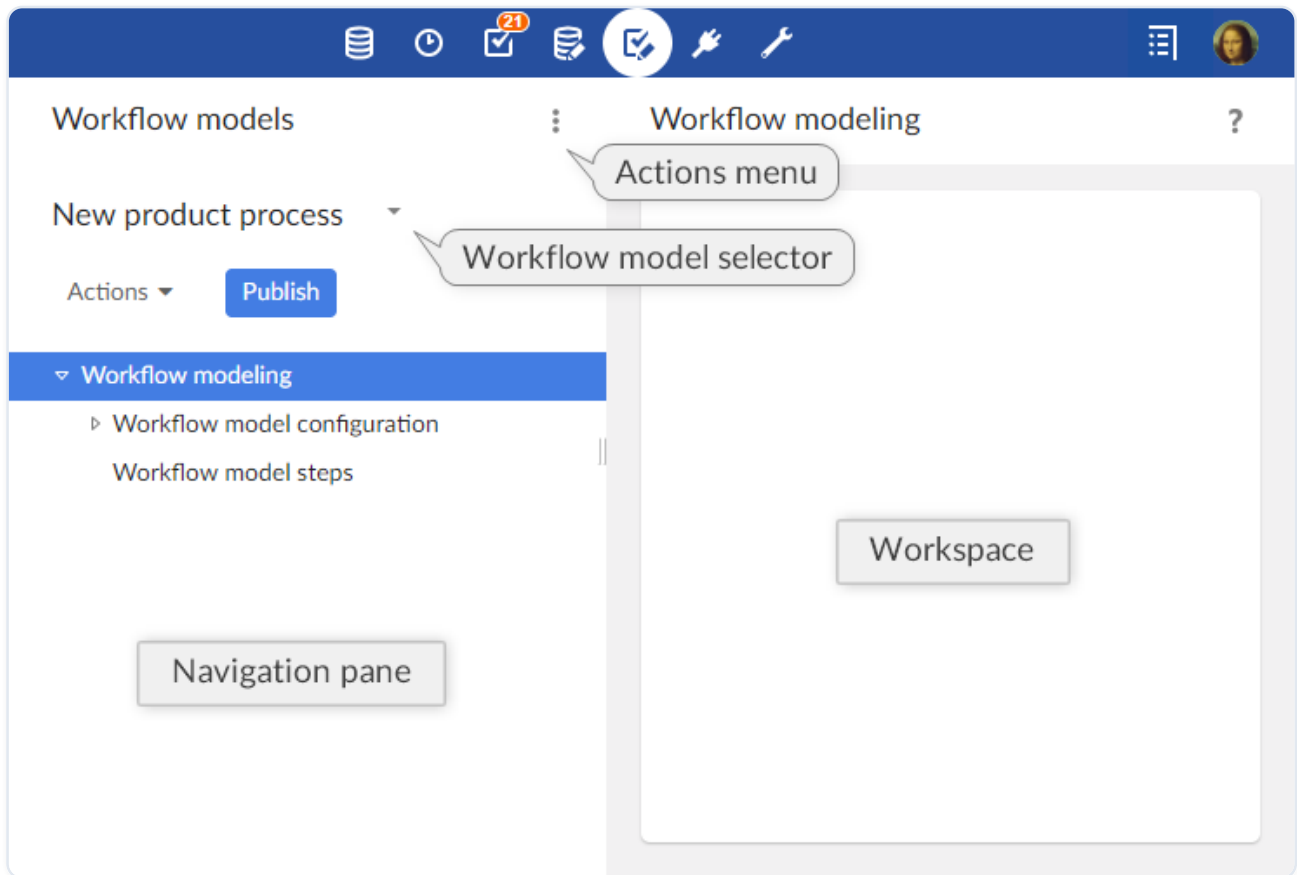
ワークフローモデルに関連する基本概念

ワークフローモデルの作成を進めるには、次の用語の基本的な理解が必要です。

- [スクリプトタスク](#)
- [ユーザータスク](#)
- [作業項目](#)
- [ワークフローの状態](#)
- [サブワークフローの呼び出し](#)
- [待機タスク](#)

- データコンテキスト

ワークフローモデル領域のユーザーインターフェイスの使用



注意:

この領域は、「高度なパースペクティブ」で許可されたユーザーのみが使用できます。許可されたユーザーのみがこれらのインターフェイスにアクセスできます。

汎用メッセージテンプレート

データワークフローの実行中に特定のイベントをユーザーに通知するために、通知メールを送信できます。

汎用テンプレートは、リポジトリ内の任意のワークフローモデルで定義および再利用できます。汎用テンプレートを使用するには、ワークフローモデル領域のメニューから選択します。

これらのテンプレートは、すべてのワークフローモデルで共有されており、ワークフローモデルの公開時に静的に含まれています。したがって、テンプレートの変更を考慮に入れるには、影響を受けるモデルを再公開して、既存の公開を更新する必要があります。

これらのテンプレートをアーカイブにエクスポートする場合は、メッセージテンプレートを含むデータセット「構成」を選択する必要があることに注意してください。

新しいテンプレートを作成するときは、次の 2 つのフィールドが必要です。

- **ラベルと説明**：テンプレートに関連付けられたローカライズされたラベルと説明を指定します。
- **メッセージ**：メッセージのローカライズされた件名と本文を指定します。

メッセージテンプレートには、`${variable.name}` などのデータコンテキスト変数を含めることができます。これらの変数は、通知が送信されるときに置き換えられます。使用できるシステム変数は次のとおりです。

system.time	リポジトリのシステム時刻。
system.date	リポジトリのシステム日付。
workflow.lastComment	前のユーザータスクに関する最後のコメント。(注意：この変数は、現在のタスクではなく、最後のユーザータスクを参照します。また、現在のタスクはワークフローが配置されているタスクであり、ユーザータスクの完了通知も含まれます)。
workflow.lastDecision	前のユーザータスクで行われた最後の決定。(注意：この変数は、現在のタスクではなく、最後のユーザータスクを参照します。また、現在のタスクはワークフローが配置されているタスクであり、ユーザータスクの完了通知も含まれます)。
user.fullName	通知されたユーザーのフルネーム。
user.login	通知されたユーザーのログイン。
workflow.process.label	現在のワークフローのラベル。
workflow.process.description	現在のワークフローの説明。
workflow.workItem.key	現在の作業項目のキー。

workflow.workItem.label	現在の作業項目のラベル。
workflow.workItem.description	現在の作業項目の説明。
workflow.workItem.offeredTo	現在の作業項目が提供されているロール。
workflow.workItem.allocatedTo	現在の作業項目が割り当てられているユーザー。
workflow.workItem.link	Web コンポーネント API を使用して、作業項目の受信トレイ内の現在の作業項目にアクセスするためのリンク。 このリンクは、現在の作業項目が定義されており、URL がワークフロー実行のメール構成で構成されている場合にのみ計算できます。
workflow.workItem.link.allocateAndStart	Web コンポーネント API を使用して、作業項目の受信トレイ内の現在の作業項目にアクセスするためのリンク。ターゲット作業項目がまだ開始されていない場合、ユーザーがリンクをクリックすると、ターゲット作業項目が自動的に割り当てられて開始されます。 このリンクは、現在の作業項目が定義されており、URL がワークフロー実行のメール構成で構成されている場合にのみ計算できます。
workflow.currentStep.label	現在のステップのラベル。
workflow.currentStep.description	現在のステップの説明。

例

一般的なテンプレートメッセージ

Today at \${system.time}, a new work item was offered to you

結果のメール

Today at 15:19, a new work item was offered to you

ワークフローの制限

次の機能は現在、EBX® ではサポートされていません。

- **スケジュールされたタスク。**順番が来るとすぐに実行されるタスクで、実行を遅らせることはできません。
- **イベントタスク。**Web サービス呼び出しなどのイベントを受信すると、ワークフローを進めることができます。
- タスク期間の **時間制限**。
- **ユーザープロフィール名は最大 64 文字です。**そうでない場合、互換性は保証されません。

関連する概念:

- [データワークフロー](#)

[ユーザーガイド](#) [目次](#)

ワークフローモデルの作成と実装

- [ワークフローモデルの作成](#)
- [ステップの実装](#)
 - [次のステップの進捗戦略](#)
 - [非表示の進行中のビュー](#)
- [ユーザータスク](#)
 - [モード](#)
 - [プロフィールのリスト](#)
 - [ユーザー割り当ての優先設定](#)
 - [サービス](#)

- 構成
- スクリプトタスク
 - ライブラリ
 - 特定
 - スクリプト
- 条件
 - ライブラリ条件
 - 特定の条件
- サブワークフローの呼び出し
- 待機タスク
- ワークフローダイアグラムの編集
 - 概要
 - アクション
 - ビュー
 - ボタン
 - 編集
 - 機能

ワークフローモデルの作成

領域に新しいワークフローモデルを作成できます。作成時に必要な情報は、リポジトリ内で一意の名前のみです。

ワークフローモデルのステップは、単一の遷移で初期化されます。ワークフローモデルを完全に実装するには、この最初の移行以降の一連のステップを定義する必要があります。

ステップの実装

ワークフローモデルは、データに対して実行する必要があるさまざまな操作と関連する条件に対応するステップを定義します。次のタイプのステップがあります。

- ユーザータスク
- スクリプトタスク

- 条件
- サブワークフローの呼び出し
- 待機タスク

データコンテキストは、各データワークフローにリンクされています。このデータコンテキストを使用して、ワークフローのステップで入力変数および出力変数として使用できる変数を定義できます。

次のステップの進捗戦略

各ステップタイプ (複数の作業項目を持つユーザータスクを除く) について、プロパティを使用して、次のステップに適用する必要がある進捗戦略を定義できます。ステップが完了すると、ワークフローの実行時のナビゲーションを定義するために、この戦略が評価されます。デフォルトでは、進捗戦略は「作業項目テーブルの表示」に設定されています。その場合、ステップ実行後、次の作業項目を選択するために、作業項目テーブル (作業項目の受信トレイまたは監視 > 作業項目) が自動的に表示されます。

別の戦略として、を選択することができます。この戦略により、ユーザーはこのワークフローで作業を続け、次のステップを直接実行できます。この実行に続いて、作業項目に到達し、接続されたユーザーがそれを開始できる場合、作業項目は自動的に開かれます (複数の作業項目に到達した場合、最初に作成されたものが開かれます)。それ以外の場合は、次のステップの進捗戦略が評価されます。作業項目に到達していない場合は、作業項目テーブルが表示されます。

この戦略は、作業項目の受信トレイに戻らずに、連続していくつかのステップを実行するために使用されます。

この戦略を無視することにつながるいくつかの制限があります。その場合、作業項目テーブルが自動的に表示されます。このプロパティは、現在のステップが複数の作業項目を持つユーザータスクの場合に無視されます。

条件の場合、「true の場合、次のステップを自動的に開く」および「false の場合、次のステップを自動的に開く」という他の 2 つの戦略を使用できます。これらの戦略により、条件の結果に応じて適用する戦略を選択できます。

サブワークフローの呼び出しの場合、進行戦略を正確にするために専用のプロパティ «フォアグラウンドサブワークフロー» を使用できます。前の進捗戦略 «次のステップを直接開く» の場合、フォアグラウンドサブワークフローを選択する必要があります。このフォアグラウンドサブワークフローに含まれるステップと関連する進捗戦略のみが評価されます。フォアグラウンドサブワークフロープロパティに関する次の特定のルールに注意してください。

- サブワークフローが 1 つだけ起動された場合、それは自動的にフォアグラウンドで考慮されます。
- 前のステップで進捗戦略「次のステップを自動的に開く」が選択されていない場合、フォアグラウンドサブワークフロープロパティは無視されます。
- すべてのサブワークフローが完了し、最後に完了したサブワークフローがフォアグラウンドの場合、サブワークフロー呼び出しステップで定義された進捗戦略が評価されます。進捗戦略が「次のステップを自動的に開く」の場合、受信トレイを表示せずに次のステップを開くことができます。すべての場合において、サブワークフローの最後のステップの進捗戦略は常に無視されます。

非表示の進行中のビュー

ステップタイプごとに、ワークフローの進行状況ビューでデフォルトで非表示にする必要があるステップを定義するためのプロパティを使用できます。

このプロパティを有効にすると、管理者以外のユーザー (ビルトインの管理者でもワークフロー管理者でもない) のワークフローの進行状況ビューでステップが自動的に非表示になります。非表示のステップは、オンデマンドで表示できます。

ユーザータスク

ユーザータスクは、人間のユーザーが実行するアクションを含むステップです。それらのラベルと説明はローカライズできます。

モード

下位互換性の理由から、デフォルトモードとレガシーモードの 2 つのユーザータスクモードを使用できます。

デフォルトでは、ユーザータスクは単一の作業項目を生成します。このモードでは、プロファイルのリストに作業項目を提供したり、ワークフローの進行状況ビューにアバターを直接表示したりするなど、より多くの機能が提供されます。

レガシーモードでは、ユーザータスクはいくつかの作業項目を生成できます。

デフォルトでは、ユーザータスク作成サービスはレガシーモードで非表示になっています。これを表示するには、`ebx.properties` ファイルでプロパティを有効にする必要があります。詳細については、[ユーザータスクのレガシーモードの無効化](#)を参照してください。

プロファイルのリスト

ユーザータスクのプロファイルの定義は、ユーザータスクモードによって異なる場合があります。

次のプロファイルに提供

定義されたプロファイルは、ユーザータスクが提供されているロールまたはユーザーです。ユーザータスクを実行すると、単一の作業項目が生成されます。単一のユーザーが定義されている場合、作業項目はこのユーザーに自動的に割り当てられます。ロールが定義されている場合、作業項目はロールのメンバーに提供されます。複数のユーザーとロールが定義されている場合、作業項目はこれらのユーザーとこれらのロールのメンバーに同時に提供されます。

参加者

参加者は、ユーザータスクの対象となるロールまたはユーザーです。デフォルトでは、ユーザータスクを実行すると、プロファイルによって作業項目が生成されます。プロファイルがロールではなくユーザーを参照している場合、作業項目はそのユーザーに直接割り当てられます。プロファイルがロールの場合、作業項目はロールのメンバーに提供されます。

詳細情報

以下も参照してください。:

- [拡張機能](#)

ユーザー割り当ての優先設定

現在のタスクを、ワークフロー内の以前のタスクを最も最近完了したユーザーに割り当てるための優先設定を行うことができます。**次のタスクを完了するユーザーへの割り当てを優先** の下のメニューを使用して、以前のタスクを選択します。そのタスクを完了したユーザーが現在のタスクが提供されるプロファイルのいずれかに属している場合、システムはそのユーザーにタスクを割り当てます。そのユーザーがそれらのプロファイルのいずれにも属していない場合、システムは通常どおり、対象となるプロファイルのユーザーにタスクを提供します。

注意:

この機能はレガシーユーザータスクでは利用できません。

サービス

TIBCO EBX® には、次のビルトインサービスが含まれています。

- データスペースへのアクセス
- データへのアクセス (デフォルトサービス)
- データスペースマージビューへのアクセス
- コンテンツの比較
- 新しいレコードの作成
- レコードの複製
- データを CSV ファイルにエクスポート
- データを XML ファイルにエクスポート
- データを CSV ファイルからインポート
- データを XML ファイルからインポート
- データスペースのマージ
- データスペース、スナップショット、またはデータセットの検証

以下も参照してください。:

- [EBX® ビルトインサービス](#)

構成

メインオプション > 拒否を有効化

デフォルトでは、決定を保存するときに アクションのみがユーザーに提供されます。

このフィールドを に設定することにより、ユーザーが作業項目を拒否できるようにすることもできます。

メインオプション > 確認リクエストを有効化

デフォルトでは、ユーザーが または ボタンをクリックして決定を保存すると、ユーザータスクの実行後に確認要求が表示されます。

この確認プロンプトを無効にするには、このフィールドを に設定します。

メインオプション > コメントを有効化

デフォルトでは、コメントは有効になっています。作業項目が開いているときは、ボタンが表示され、ユーザーはコメントを入力できます。

このプロパティを に設定すると、この ボタンを非表示にできます。

メインオプション > コメントが必要

デフォルトでは、作業項目に関連付けられたコメントを送信することはオプションです。

このフィールドを目的の検証基準に設定することにより、決定を保存する前にユーザーにコメントの入力を要求することができます。コメントは、常に必須、作業項目が承認された場合にのみ必須、または作業項目が拒否された場合にのみ必須に設定できます。

メインオプション > カスタマイズされたラベル

ユーザータスクが実行されると、ユーザーは対応するボタンをクリックして作業項目を承認または拒否できます。ワークフローモデルでは、ユーザータスクがこれら 2 つのボタンのカスタマイズされたラベルと確認メッセージを定義することができます。これを使用して、ボタンをより具体的なコンテキストに適合させることができます。

終了 > タスク終了基準

単一のユーザータスクを複数の 参加者 に割り当てることができるため、ワークフローの実行中に複数の作業項目が生成されます。ワークフローモデルでユーザータスクを定義する場合、コンポーネントの作業項目のステータスに基づいて、ユーザータスクが終了したかどうかを判断するための事前定義された方法の 1 つを選択できます。ユーザータスクの終了要件が満たされると、データワークフローはそのモデルで定義された次のステップに進みます。

たとえば、製品レコードを承認する必要があるユーザータスクの場合、3 人の潜在的な参加者を指定できます。タスク終了基準では、製品レコードを 3 人のユーザー全員が承認する必要があるのか、最初に応答するユーザーのみが承認する必要があるのかを指定できます。

デフォルトのユーザータスク終了基準は、 です。

注意:

ユーザータスクの完了の判定を処理するためにメソッド

`UserTask.handleWorkItemCompletion` を上書きするサービス拡張を指定する場合、ユーザーインターフェイスを通じて定義されたタスク終了基準が満たされていることをコード内

から検証するのは、拡張の開発者の責任です。詳細については、JavaDoc の `UserTask.handleWorkItemCompletion` を参照してください。

終了 > 拒否許容度

デフォルトでは、ユーザーがワークフローの実行中に作業項目を拒否すると、ユーザータスクはエラー状態になり、ワークフローの進行は停止します。ユーザータスクがエラー状態の場合、ワークフロー管理者は、ワークフローの実行を続行するために、エラーが発生したステップを再生することによって介入する必要があります。

このデフォルトの動作を変更するために、許容する特定の数の作業項目の拒否を定義することができます。許容される拒否の制限内では、エラーは発生せず、ユーザータスクをいつ終了するかを決定するのはタスク終了基準です。

次のタスク終了基準は、すべての拒否を自動的に許容します。

- 「すべての作業項目が承認または拒否されたとき」
- 「すべての作業項目が受け入れられたとき、または、1つの作業項目が拒否されたとき直ちに」

拡張機能

特定のデータワークフローのコンテキストでタスクが動的に動作するように、カスタムクラスを指定できます。たとえば、これを使用して、デフォルトの動作とは異なる方法で作業項目を作成したり、ユーザータスクを完了したりできます。

指定されたルールは、`UserTask` クラスを拡張する必要がある JavaBean です。

注意:

ルールが指定され、`handleWorkItemCompletion` メソッドが上書きされると、完了戦略は自動的にチェックされなくなります。開発者はこのメソッド内で完了をチェックする必要があります。

通知

特定のイベントが発生したときに、通知メールをユーザーに送信できます。イベントごとに、コンテンツテンプレートを指定できます。

ユーザータスクに関連して送信されるすべての電子メールを受信するモニタープロファイルを定義することができます。

以下も参照してください。:

- 汎用メッセージテンプレート

リマインダー

未処理の提供または割り当てられた作業項目のリマインダーメールは、関係するユーザーに定期的に送信できます。リマインダーの受信者は、作業項目が提供または割り当てられるユーザーと、コピーの受信者です。

リマインダーメールの内容は、作業項目の現在の状態によって決まります。つまり、作業項目が提供されている場合、通知には「提供された作業項目」テンプレートが使用されます。作業項目が割り当てられている場合、通知は「割り当てられた作業項目」テンプレートを使用します。

期限

各ユーザータスクには、完了期限を設定できます。この日付が過ぎ、関連する作業項目が完了していない場合、関係するユーザーに通知メールが送信されます。この同じ通知メールは、タスクが完了するまで毎日送信されます。

期限には2つのタイプがあります。

- **絶対期限**: カレンダーの日付。
- **相対期限**: 時間、日、または月単位の期間。期間は、ユーザータスクの開始またはワークフローの開始である参照日に基づいて評価されます。

スクリプトタスク

スクリプトタスクは、人間のユーザーが関与することなく実行される自動タスクです。

スクリプトタスクには3つの種類があり、定義するとワークフローモデルのステップで使用できます。

ライブラリ	EBX® には、そのまま使用できる多数のビルトインライブラリスクリプトタスクが含まれています。 追加のライブラリスクリプトタスクは、<code>module.xml</code> ファイルで宣言する必要があります。ライブラリスクリプトタスクでは、ラベル、説明、およびパラメーターを定義する必要があります。ユーザーがワークフローモデルのステップでライブラリスクリプトタスクを選択すると、関連するパラメーターが動的に表示されます。
特定	カスタムアクションを実行する Java クラスを指定します。関連するクラスは、ワークフローモデルと同じモジュールに属している必要があります。そのラベルと説明は、ワークフローモデルのユーザーに動的に表示されません。
スクリプト	DSL (ドメイン固有言語) に基づくスクリプトを指定して、カスタム操作を定義します。EBX® IDE は、スクリプトの作成を容易にするために使用されます。 スクリプトは、名前、ラベル、説明、およびパラメーターによって定義されます。ユーザーがワークフローモデルのステップのスクリプトの名前を選択すると、関連付けられたパラメーターが動的に表示されます。

TIBCO EBX® モジュールのパッケージ化

ライブラリ

EBX® には、次のビルトインライブラリスクリプトタスクが含まれています。

- データスペースの作成
- スナップショットの作成

- データスペースのマージ
- アーカイブのインポート
- データスペースを閉じる
- データコンテキスト変数の設定
- メールの送信
- レコードの削除 (注意：このスクリプトは複数のレコードを削除できます)

ライブラリスクリプトタスクは、クラス `ScriptTaskBean` を拡張するクラスです。ビルトインのライブラリスクリプトタスクに加えて、ワークフローモデルで使用するために追加のライブラリスクリプトタスクを定義できます。それらのラベルと説明はローカライズできます。

メソッド `ScriptTaskBean.executeScript` は、データワークフローが対応するステップに到達したときに呼び出されます。

注意:

メソッド `ScriptTaskBean.executeScript` は、メソッドが実行されるとすぐにデータワークフローが進行するため、スレッドを作成してはなりません。したがって、このメソッドの各操作は同期している必要があります。

例 を参照してください。

実装が Java Bean 仕様に従っている場合、ライブラリスクリプトタスクの変数を動的に設定することができます。変数は、`module.xml` でライブラリスクリプトタスクの Bean のパラメーターとして宣言する必要があります。ワークフローデータコンテキストには、Java Bean からアクセスできません。

注意:

一部のビルトインライブラリスクリプトタスクは、国際化と互換性がないため、「非推奨」としてマークされています。国際化と互換性のある新しいスクリプトタスクを使用することをお勧めします。

特定

特定のスクリプトタスクは、クラス `ScriptTask` のサンプル を拡張するクラスです。

メソッド `ScriptTask.executeScript` は、データワークフローが対応するステップに到達したときに呼び出されます。

注意:

メソッド `ScriptTask.executeScript` は、メソッドが実行されるとすぐにデータワークフローが進行するため、スレッドを作成してはなりません。したがって、このメソッドの各操作は同期している必要があります。

例 を参照してください。

特定のスクリプト タスクの Bean の変数を動的に設定することはできません。ただし、ワークフロー データ コンテキストには Java Bean からアクセスできます。

スクリプト

EBX® スクリプトは、DSL (ドメイン固有言語) に基づく手続き型言語です。カスタムタスクの定義に使用できる大規模なライブラリを提供します。

スクリプトのメソッド `executeScriptTask` は、データワークフローが対応するステップに到達したときに呼び出されます。例 を参照してください。

条件

条件は、ワークフローの決定ステップです。

2 種類の条件が存在し、一度定義すると、ワークフローモデルのステップで使用できます。

ライブラリの条件

EBX® には、そのまま使用できる多数のビルトインライブラリ条件が含まれています。

追加のライブラリスクリプトタスクは、`module.xml` ファイルで宣言する必要があります。ライブラリ条件は、そのラベル、説明、およびパラメーターを定義する必要があります。ユーザーがワークフローモデルのステップのライブラリ条件を選択すると、関連するパラメーターが動的に表示されます。

特定の条件

カスタム条件を実装する Java クラスを指定します。関連するクラスは、ワークフローモデルと同じモジュールに属している必要があります。そのラベルと説明は、ワークフローモデルのユーザーに動的に表示されません。

TIBCO EBX® モジュールのパッケージ化

ライブラリの条件

EBX® には、次のビルトインライブラリ条件が含まれています。

- データスペースを変更したかどうか
- データが有効かどうか
- 最後のユーザータスクが受け入れられたかどうか
- 値が null または空かどうか
- 値が等しいかどうか

ライブラリ条件は、クラス `ConditionBean` を拡張するクラスです。ビルトインのライブラリ条件に加えて、ワークフローモデルで使用するために追加のライブラリ条件を定義できます。それらのラベルと説明はローカライズできます。

[例](#) を参照してください。

ライブラリ条件の変数は、その実装が Java Bean 仕様に従っている場合は動的に設定できます。変数は、`module.xml` でライブラリ条件の Bean のパラメーターとして宣言する必要があります。ワークフローデータコンテキストは、Java Bean からアクセスできません。

特定の条件

特定の条件は、クラス `Condition` を拡張するクラスです。

[例](#) を参照してください。

特定の条件に対して Bean の変数を動的に設定することはできません。ただし、ワークフローデータ コンテキストには Java Bean からアクセスできます。

サブワークフローの呼び出し

ワークフローモデルのサブワークフロー呼び出しステップは、現在のワークフローを待機状態にし、1 つ以上のワークフローを呼び出します。

サブワークフロー呼び出しステップで単独で呼び出すことにより、現在のワークフローに別のワークフローモデル定義を含めることができます。

複数のサブワークフローが単一のステップで呼び出される場合、それらは同時に並行して実行されます。元のワークフローが次のステップに進む前に、すべてのサブワークフローを終了する必要があります。各サブワークフローのラベルと説明はローカライズできます。

プロパティ « Foreground sub-workflow » は、複数のサブワークフローが起動されたときに進捗戦略を正確にするのに役立ちます。前のステップの進行戦略が「次のステップを直接開く」の場合、このプロパティは、フォアグラウンドで考慮する必要があるサブワークフローを定義します (この動作を保持できるサブワークフローは 1 つだけです)。このフォアグラウンドサブワークフローで生成された作業項目のみが、受信トレイを通過せずに自動的に開くことができます。サブワークフローが 1 つだけ定義されている場合、このプロパティは無視されます (単一のサブワークフローがフォアグラウンドで自動的に考慮されます)。詳細については、進捗戦略の [次のステップの進捗戦略](#) を参照してください。

サブワークフロー呼び出しには 2 つの種類があります。

静的	データワークフローでステップに到達するたびに呼び出される 1 つ以上のサブワークフローを定義します。サブワークフローごとに、ローカライズされたラベルと説明、およびデータコンテキストでの入力変数と出力変数のマッピングを設定できます。 このモードは、起動するサブワークフローと出力マッピングが事前に決定されている場合に役立ちます。
動的	カスタムサブワークフロー呼び出しを実装する Java クラスを指定します。コードによってサブワークフローとして呼び出される可能性のあるすべてのワークフローは、依存関係として宣言する必要があります。 ワークフローデータコンテキストには、Java Bean から直接アクセスできます。

動的なサブワークフローの呼び出しは、`module.xml` ファイルで宣言する必要があります。

このモードは、サブワークフローの起動が条件付きである場合 (たとえば、データコンテキスト変数に依存している場合)、または出力マッピングがサブワークフローの実行に依存している場合に役立ちます。

待機タスク

ワークフローモデルの待機タスクステップは、特定のイベントが受信されるまで、現在のワークフローを待機状態にします。

待機タスクに到達すると、ワークフローエンジンは、待機タスクに関連付けられた一意の再開識別子を生成します。この識別子は、待機タスクを再開するために必要であり、その結果、関連するワークフローが必要になります。

待機タスクは、待機タスクを再開する権限を持つプロファイルと、待機タスク Bean を実装する Java クラス `WaitTaskBean` を指定します。

ワークフローデータコンテキストには、Java Bean から直接アクセスできます。

待機タスク Bean は `module.xml` ファイルで宣言する必要があります。

まず、ワークフローが待機を開始すると、待機タスク Bean が呼び出されます。このとき、生成された再開識別子は、たとえば Web サービスを呼び出すために使用できます。次に、待機タスクが再開されると、待機タスク Bean が呼び出されます。このようにして、受信したパラメーターに応じてデータコンテキストを更新できます。

注意:

ビルトインの管理者には、ワークフローを再開する権利が常にあります。

ワークフローダイアグラムの編集

概要

ワークフローモデルは、BPMN のような編集可能なダイアグラムで表示されます。

このビューは完全な編集機能を提供し、モデラーが設計しているワークフローモデルを明確に表示するのに役立ちます。

また、ダイアグラムは BPMN 標準から派生していますが、EBX® ワークフローの概念がわずかに異なるため、BPMN を厳密に表したものではありませんことにも注意してください。

レイアウトの保存

変更したレイアウトを保存することができます。これはユーザーベースの保存ではないことに注意してください。すべてのユーザーによって共有されます。

アクション

PNG としてエクスポート	PNG 画像を作成します。
SVG としてエクスポート	SVG 画像を作成します。
PDF としてエクスポート	ワークフローモデルから PDF ドキュメントを作成します。エクスポートをカスタマイズするために、いくつかのプロパティを使用できます。PDF ページの向き (横向きまたは縦向き)、サイズ (A3、文字など) を構成できます。モデルに詳細を提供する可能性のある 2 つのプロパティもあります。 - 識別子の表示 (デフォルト値は true) : ワークフローダイアグラムツールバーの [ステップ識別子の表示] と同じ動作ですが、PDF のエクスポートに制限されています。各ステップのステップ識別子が表示されます。 - インデックスの追加 (デフォルト値は true) : pdf の最後に、ダイアグラムのすべてのステップを記録するインデックスが追加されます。複数のステップのタイトルが長すぎて完全に表示できない場合に役立つことがあります。

このプロパティでは、どのインデックス行がどのステップに対応しているかを認識するのに役立つため、ステップ識別子を表示することをお勧めします。

ビュー

レイアウト > デフォルトレイアウト	ダイアグラムにデフォルトのレイアウトを適用します。
表示 > グリッドの表示/非表示	グリッドが表示されていない場合はグリッドを表示し、表示されていない場合は非表示にします。
表示 > ズームイン	ダイアグラム上でズームインが実行されます。より速く進むには、マウスポインタがダイアグラムの境界を指している状態で「Ctrl+上マウスホイール」を使用できます。
表示 > ズームアウト	ダイアグラム上でズームアウトが実行されます。より速く進むには、マウスポインタがダイアグラムの境界を指している状態で「Ctrl+下マウスホイール」を使用できます。
表示 > ステップ識別子の表示/非表示	ワークフローステップの識別子を表示するバッジを表示/非表示にします。
プランビュー > 階層	有効になっている場合、特定のワークフローモデルの階層ビューを表示します。

ボタン

レイアウトを保存	現在のレイアウトを保存します。
レイアウトを保存して閉じる	現在のレイアウトを保存し、サービスを閉じます。

元に戻す	変更を元に戻し、以前に保存したレイアウトを再読み込みします。
終了	サービスを閉じます。

編集

作成	リンクにカーソルを合わせるか選択すると、「+」が表示されます。これは、新しいステップを作成するために使用されます。
削除	DEL [DEL] キーを押すと、選択したノードが削除されます。 このアクションは元に戻せないことに注意してください。
ノードツールバー	ノードにカーソルを合わせるか、ノードを直接選択すると、ツールバーが表示されます。次のいずれかに使用されます。 • ステップを編集します。 • ステップを削除します。 • ステップの複製を作成します。 • 既存のステップに再リンクします。 • 進行中のビューを表示または非表示にします。
ノードの編集	ステップをすばやく編集するには、ノードをダブルクリックします。

機能

ダイアグラムビューは、便利な追加機能を提供します。

最後のアクションを元に戻す	CTRL + Z 以下の操作を行った後は実行できないことに注意してください。 • ステップの作成/削除/編集/複製 • 再リンク • 進行中のビューを表示/非表示 • 元に戻す
ズームイン/ズームアウト	マウスの中央ボタン、次にマウスホイール/CTRL、次にマウスホイール。[表示] も参照してください
複数選択	Ctrl を押しながら選択したノードまたはリンクをクリック/選択長方形を描画 (領域を描画する前に、左クリックを 1 秒間押し続ける必要があります)。
リンク描画のカスタマイズ	リンクをクリックすると、角に表示される四角をドラッグしてセグメントを移動するか、中央の円を移動して特定のセグメントを分離することができます。
ステップの編集	ステップをダブルクリックすると、エディションフォームが表示されます。
概要	パネル内を移動するために使用できるミニチュアワークフローダイアグラムビューを備えたパネルが利用可能になりました。このパネルは、ワークフローダイアグラムビューに割り当てられた領域内で折りたたんだり、展開したり、ドラッグしたりできます。

ワークフローモデルの構成

- ワークフローモデルに関連する情報
- ワークフローモデルのプロパティ
- データコンテキスト
- カスタムワークフロー実行ビュー
- 関連するデータワークフローの権限
- ワークフローモデルのスナップショット
- ワークフローモデルの削除

ワークフローモデルに関連する情報

ワークフローモデルの所有者とドキュメントを表示および編集するには、ナビゲーションペインでワークフローモデルの [ワークフローモデルの \[アクション\] メニュー](#) から [ワークフローモデルの \[アクション\] メニュー](#) を選択します。

所有者	ワークフローモデルの所有者を指定します。所有者は、ワークフローモデルの情報を編集し、そのアクセス許可を定義する権限を持ちます。
ローカライズドキュメント	ワークフローモデルのローカライズされたラベルと説明。
有効化済み	<i>このプロパティは非推奨です。</i> ワークフローモデルがアクティブ化されているかどうか。公開できるようにするには、ワークフローモデルをアクティブ化する必要があります。

ワークフローモデルのプロパティ

ワークフローモデルの構成には、 [ワークフローモデルの構成](#) の下のナビゲーションペインからアクセスできます。

| モジュール名 | 特定の Java リソース (ユーザータスク拡張、特定のスクリプトおよび条件) を含むモジュール。 || 開始の通知 | データワークフローが起動されたときに、テンプレートに基づいて通知を送信するプロファイルのリスト。 [汎用メッセージテンプレート](#) を参照してください。 || 完了の通知 | データワークフローが完了したときに、テンプレートに基づいて通知を送信するプロファイルのリスト。通知は、ワークフローが通常の状況で完了した場合、つまり管理アクションが原因ではない場合にのみ送信されます。 [汎用メッセージテンプレート](#) を参照してください。 ||

エラーの通知 | データワークフローがエラー状態のときに、テンプレートに基づいて通知を受信するプロファイルのリスト。汎用メッセージテンプレートを参照してください。|| 優先度 | デフォルトでは、このモデルに関連付けられている各ワークフローは、この優先度で起動されます。優先度の設定はオプションです。ここで優先度が定義されておらず、リポジトリにデフォルトの優先度が設定されている場合、リポジトリのデフォルトの優先度は、優先度が割り当てられていない関連ワークフローに使用されます。詳細については、作業項目の優先度を参照してください。 **注意: **ワークフロー管理者として定義されているユーザーのみが、関連するデータワークフローの優先度レベルを手動で変更できます。|| クイック起動を有効化 | デフォルトでは、ワークフローが起動されると、ユーザーは新しいワークフローのドキュメントを中間形式で入力するように求められます。このドキュメントはオプションです。プロパティを に設定すると、このドキュメントの手順をスキップして、ワークフローの起動に直接進むことができます。|| 最初のステップを自動的に開く | ワークフローの起動後にナビゲーションを決定できます。デフォルトでは、ワークフローが起動されると、現在のテーブル (ワークフローランチャーまたはモニタリング > パブリケーション) が自動的に表示されます。このプロパティを有効にすると、ワークフローユーザーは起動されたワークフローで作業を続けることができます。最初のワークフローステップが実行された後、作業項目に到達し、この作業項目をワークフロー作成者が開始できる場合、作業項目は自動的に開かれます (複数の作業項目に到達した場合、最初に作成されたものが開かれます)。これにより、ユーザーは作業項目の受信トレイから対応する作業項目を選択する必要がなくなります。作業項目に達していない場合は、次のステップの進捗戦略が評価されます。作業項目が開かれていない場合は、ワークフローが開始されたテーブルが表示されます。 **制限事項: **最初のステップがサブワークフローの呼び出しである場合、このプロパティは無視されます。|| ワークフロートリガー | ワークフローのメインイベントをインターセプトするコンポーネント。この Bean は `module.xml` ファイルで宣言する必要があります。例を参照してください。|| 権限 | ワークフローモデルに関連付けられたデータワークフローに関連するアクションの権限。この Bean は `module.xml` ファイルで宣言する必要があります。例を参照してください。|| プログラムによるアクションの許可 | ワークフローの権限を処理するカスタムコンポーネントを定義します。設定されている場合、これはプロパティ 'Permissions' で定義されているすべての権限を上書きします。 |

データコンテキスト

データコンテキスト構成には、ナビゲーションペインからアクセスできます。

各ワークフローには独自のデータコンテキストがあるため、実行中に独自のローカルデータスペースを使用できます。これにより、ワークフローの実行を指示する値を保存および変更することができます。

データコンテキストは変数のリストによって定義されます。各変数には次のプロパティがあります。

名称	変数の識別子。
デフォルト値	定義されている場合、変数はこのデフォルト値で初期化されます。
入力パラメーター	この変数を入力パラメーターとして定義するには、[はい] をチェックする必要があります。
出力パラメーター	この変数を出力パラメーターとして定義するには、「はい」をチェックする必要があります。そうでない場合、この変数はタスク定義インターフェイスの出力パラメーターのリストに表示されません。

カスタムワークフロー実行ビュー

ワークフロー実行ビューのカスタマイズには、ナビゲーションペインからアクセスできます。

カスタマイズにより、作業項目とワークフロービュー (受信トレイ、作業項目の監視、アクティブなワークフローの監視、および完了したワークフロー) の特定の列を構成できます。ユーザーが特定のニーズに合わせてカスタム値を持つことができるようにします。これらの値は、データコンテキストの変数が評価されることで動的になります。

関連するデータワークフローの権限

ワークフロー管理	ワークフローで管理アクションを実行できるプロファイルを定義します。管理アクションには、ステップの再生、ワークフローの再開、ワークフローの終了、公開の無効化、および公開解除が含まれます。これらのアクションを実行するために、このプロファイルには「ワークフローの視覚化」権限が自動的に付与されます。ビルトインの管理者には、常にワークフロー管理権限があります。
ワークフロー管理 > ステップの再生	ワークフローステップの再生を許可するプロファイルを定義します。このアクションを実行するために、このプロファイルには

	「ワークフローの視覚化」権限が自動的に付与されます。[監視] > [アクティブなワークフロー] セクションのボタンを使用して、ステップを再生できます。「ワークフロー管理」権限を持つプロファイルは、この特定のアクションの実行を自動的に許可されます。ビルトインの管理者には、常にステップを再生する権限があります。
ワークフロー管理 > ワークフローの終了	ワークフローの終了とクリーンアップを許可するプロファイルを定義します。このアクションを実行するために、このプロファイルには「ワークフローの視覚化」権限が自動的に付与されます。[監視] > [アクティブなワークフロー] セクションのボタンを使用して、アクティブなワークフローを終了およびクリーンアップできます。[完了したワークフロー] セクションのボタンを使用して、完了したワークフローを削除できます。「ワークフロー管理」権限を持つプロファイルは、この特定のアクションの実行を自動的に許可されます。ビルトインの管理者には、ワークフローを終了する権限が常にあります。
ワークフロー管理 > ワークフローの強制再開	待機中のワークフローを強制的に再開できるプロファイルを定義します。このアクションを実行するために、このプロファイルには「ワークフローの視覚化」権限が自動的に付与されます。[監視] > [アクティブなワークフロー] セクションのボタンを使用して、ワークフローを再開できます。「ワークフロー管理」権限を持つプロファイルは、この特定のアクションの実行を自動的に許可されます。ビルトインの管理者には、ワークフローを再開する権利が常にあります。

ワークフロー管理 > パブリケーションの無効化	ワークフローパブリケーションを無効にできるプロファイルを定義します。このアクションを実行するために、このプロファイルには「ワークフローの視覚化」権限が自動的に付与されます。[監視] > [パブリケーション] セクションのボタンを使用して、パブリケーションを無効にすることができます。アクティブなパブリケーションにのみ表示されます。「ワークフロー管理」権限を持つプロファイルは、この特定のアクションの実行を自動的に許可されます。ビルトインの管理者には、パブリケーションを無効にする権限が常にあります。
ワークフロー管理 > 非公開	ワークフローパブリケーションの非公開を許可するプロファイルを定義します。このアクションを実行するために、このプロファイルには「ワークフローの視覚化」権限が自動的に付与されます。[監視] > [パブリケーション] セクションのボタンは、無効になっているパブリケーションを非公開にする場合にのみ使用できます。「ワークフロー管理」権限を持つプロファイルは、この特定のアクションの実行を自動的に許可されます。ビルトインの管理者には、常に非公開権限があります。
割り当て管理	作業項目の割り当てを管理できるプロファイルを定義します。割り当てアクションには、作業項目の割り当て、作業項目の再割り当て、および作業項目の割り当て解除が含まれます。これらのアクションを実行するために、このプロファイルには「ワークフローの視覚化」権限が自動的に付与されます。ビルトインの管理者には、常に割り当て管理権限があります。

割り当て管理 > 作業項目の割り当て	作業項目の割り当てを許可するプロファイルを定義します。これらのアクションを実行するために、このプロファイルには「ワークフローの視覚化」権限が自動的に付与されます。[監視] > [作業項目] セクションのボタンを使用して、作業項目を割り当てることができます。提供された作業項目にのみ表示されます。「割り当て管理」権限を持つプロファイルは、この特定のアクションを自動的に実行できます。ビルトインの管理者には、常に作業項目の割り当て権限があります。
割り当て管理 > 作業項目の再割り当て	作業項目の再割り当てを許可するプロファイルを定義します。このアクションを実行するために、このプロファイルには「ワークフローの視覚化」権限が自動的に付与されます。[監視] > [作業項目] セクションのボタンを使用して、作業項目を再割り当てできます。割り当てられた作業項目にのみ表示されます。「割り当て管理」権限を持つプロファイルは、この特定のアクションを自動的に実行できます。ビルトインの管理者には、常に作業項目の再割り当て権限があります。
割り当て管理 > 作業項目の割り当て解除	作業項目の割り当てを解除できるプロファイルを定義します。このアクションを実行するために、このプロファイルには「ワークフローの視覚化」権限が自動的に付与されます。[監視] > [作業項目] セクションのボタンを使用して、作業項目の割り当てを解除できます。割り当てられた作業項目にのみ表示されます。「割り当て管理」権限を持つプロファイルは、この特定のアクションを自動的に実行できます。ビルトインの管理者には、常に作業項目の割り当て解除権限があります。

ワークフローの起動	新しいワークフローを手動で起動できるプロファイルを定義します。この権限により、[ワークフローランチャー] セクションのアクティブなパブリケーションからワークフローを起動できます。ビルトインの管理者には、常に起動ワークフローの権限があります。 ワークフローを起動する権限は、API による場合でも常に自動的に検証されます。これらの自動権限チェックをバイパスして、特定の認証規則を手動で管理することもできます。 このためには、<code>ebx.properties</code> <code>permissionCheckDelegated</code> を <code>true</code> に設定し、専用 API <code>WorkflowEngine.isUserAuthorizedToLaunch</code> を使用して権限評価を開発者に委任します。
ワークフローの視覚化	ワークフローの視覚化を許可するプロファイルを定義します。デフォルトでは、エンドユーザーは [受信トレイ] セクションで提供または割り当てられた作業項目のみを表示できます。この権限により、このワークフローモデルに関連付けられているパブリケーション、ワークフロー、および作業項目を [監視] セクションと [完了したワークフロー] セクションで視覚化することもできます。このプロファイルには、「完了したワークフローの視覚化」権限が自動的に付与されます。ビルトインの管理者には、常にワークフローの視覚化権限があります。
ワークフローの視覚化 > ワークフローの作成者はワークフローを視覚化できます	有効にすると、ワークフローの作成者は、起動したワークフローを表示する権限を持ちます。この制限された権限により、彼が起動したワークフローと、[モニタリング] > [アクティブなワークフロー]、[モニタリング] > [作業項目]、および [完了済みワークフロー]

	セクションの関連する作業項目へのアクセスが許可されます。デフォルト値は「いいえ」です。
ワークフローの視覚化 > 完了したワークフローの視覚化	完了したワークフローを視覚化できるプロファイルを定義します。この権限により、「完了したワークフロー」セクションで完了したワークフローを視覚化し、それらの履歴にアクセスできます。「ワークフローの視覚化」権限を持つプロファイルは、このアクションの実行を自動的に許可されます。ビルトインの管理者は、完了したワークフローの視覚化権限を常に持っています。

注意:

特定の権限が割り当てられていないユーザーは、そのユーザーに提供または割り当てられている、このワークフローに関連付けられた作業項目のみを表示できます。

以下も参照してください。:

- [ワークフロー管理](#)

ワークフローモデルのスナップショット

ワークフローモデルのスナップショットの履歴は、> から管理できます。

履歴テーブルには、選択したワークフローモデルを含むすべてのスナップショットが表示され、ワークフローモデルが公開されているかどうかを示されます。スナップショットごとに、ボタンを使用して、対応するワークフローモデルをエクスポートまたは表示できます。

ワークフローモデルの削除

ワークフローモデルは削除できますが、関連するパブリケーションはデータワークフロー領域で引き続きアクセスできます。削除されたワークフローモデルと同じ名前で新しいワークフローモデルが作成された場合、パブリッシングは古いパブリケーションを置き換えるように求めます。

以下も参照してください。:

- [ワークフローモデルの公開](#)

[ユーザーガイド 目次](#)

ワークフローモデルの公開

- [ワークフローパブリケーションについて](#)
- [公開とワークフローモデルのスナップショット](#)
- [パブリケーションのサブワークフロー](#)

ワークフローパブリケーションについて

ワークフローモデルの定義後、ワークフローモデルを公開することで、承認されたユーザーが関連付けられたデータワークフローを起動できるようにします。

公開するには、次の 2 つのサービスを利用できます。

- ナビゲーションペインの ボタン：このサービスでは、現在編集されているモデルのみを公開できます。
- セクションタイトル と同じレベルのツールバーにある ボタン：このサービスにより、一度に多数のモデルを公開できます。最初のステップでは、公開する多くのモデルを選択できます。このステップでは、サブワークフローの呼び出しを含まないモデルのみが表示されます。実際、サブワークフローを持つワークフローは、大量公開の資格がありません。

ワークフローモデルは複数回公開できます。パブリケーションは、そのパブリケーション名で識別されます

公開およびワークフローモデルのスナップショット

ワークフローモデルを公開すると、現在の状態のスナップショットが取得されます。作成するスナップショットのラベルと説明を指定できます。デフォルトのスナップショットラベルは、発行の日付と時刻です。デフォルトの説明は、ワークフローモデルを公開したユーザーを示します。

公開されるワークフローモデルごとに、指定された公開名は一意にする必要があります。ワークフローモデルがすでに公開されている場合は、同じ公開名を再利用することで既存の公開を更新できます。特定のワークフローモデルに関連付けられている既存のワークフローパブリケーションの名前は、ドロップダウンメニューで利用できます。パブリケーションの更新の場合、古い

バージョンはデータワークフローの起動に使用できなくなりますが、既存のワークフローを終了するために使用されます。さまざまなバージョンのコンテンツは、ワークフローモデルのスナップショット履歴で表示できます。

以下も参照してください。:

- [ワークフローモデルのスナップショット](#)

パブリケーションのサブワークフロー

サブワークフロー呼び出しステップを含むワークフローモデルを公開する場合、サブワークフローのモデルを個別に公開する必要はありません。管理の観点から、メインワークフローのモデル (現在ユーザーによって公開されているもの) とサブワークフローのモデルは、単一のエンティティとして公開されます。

システムは、サブワークフローとして使用されるワークフローモデルへの依存関係を計算し、依存モデルごとに 1 つのパブリケーションを自動的に作成します。これらの技術パブリケーションは、サブワークフローを起動するワークフロー エンジン専用であり、ワークフローデータ領域では使用できません。

複数のパブリケーションは、サブワークフロー呼び出しステップを含むワークフローモデルでは使用できません。この場合、公開の最初のステップ (公開するワークフローモデルの選択) が提供されないのはこのためです。

メインワークフローモデルを再公開すると、呼び出されたサブワークフローモデルが自動的に更新されます。

サブワークフローモデルはメインワークフローモデルとして個別に公開できますが、このサブワークフローを使用してすでに公開されているメインワークフローモデルで使用されているバージョンは更新されません。

[ユーザーガイド](#) [目次](#)

データワークフロー

- [データワークフローの概要](#)
- [データワークフロー領域のユーザーインターフェイスの使用](#)
- [作業項目](#)

- データワークフローの管理
 - データワークフローの起動と監視
 - データワークフローの管理

データワークフローの概要

概要

データワークフローは、ワークフローモデルのパブリケーションを使用して定義された、実行される段階的なデータ管理プロセスです。これにより、ユーザーは自動化された手順だけでなく、一連のデータに対して共同でアクションを実行できます。ワークフローモデルが開発および公開されると、結果の公開を使用してデータワークフローを起動し、定義されたステップを実行できます。

ワークフローモデルによって定義されたワークフローユーザー権限に応じて、ユーザーは関連するデータワークフローに対して次のアクションの1つ以上を実行できます。

- デフォルトの権限を持つユーザーとして、割り当てられた作業項目に取り組み、完了します
- ワークフローの起動権限を持つユーザーとして、ワークフローモデルのパブリケーションから新しいデータワークフローを作成します
- ワークフローモニターとして、進行中のデータワークフローの進行状況を追跡し、完了したデータワークフローの履歴を参照します
- 作業項目の割り当ての管理者として、他のユーザーおよび役割の作業項目の割り当てを手動で変更します
- ワークフロー管理者として、ステップの再生、進行中のワークフローの終了、データワークフローの起動に使用できないパブリケーションのレンダリングなど、さまざまな管理アクションを実行します

以下も参照してください:

- 作業項目
- データワークフローの起動と監視
- データワークフローの管理
- 関連するデータワークフローの権限

関連する概念:

- ワークフローモデル

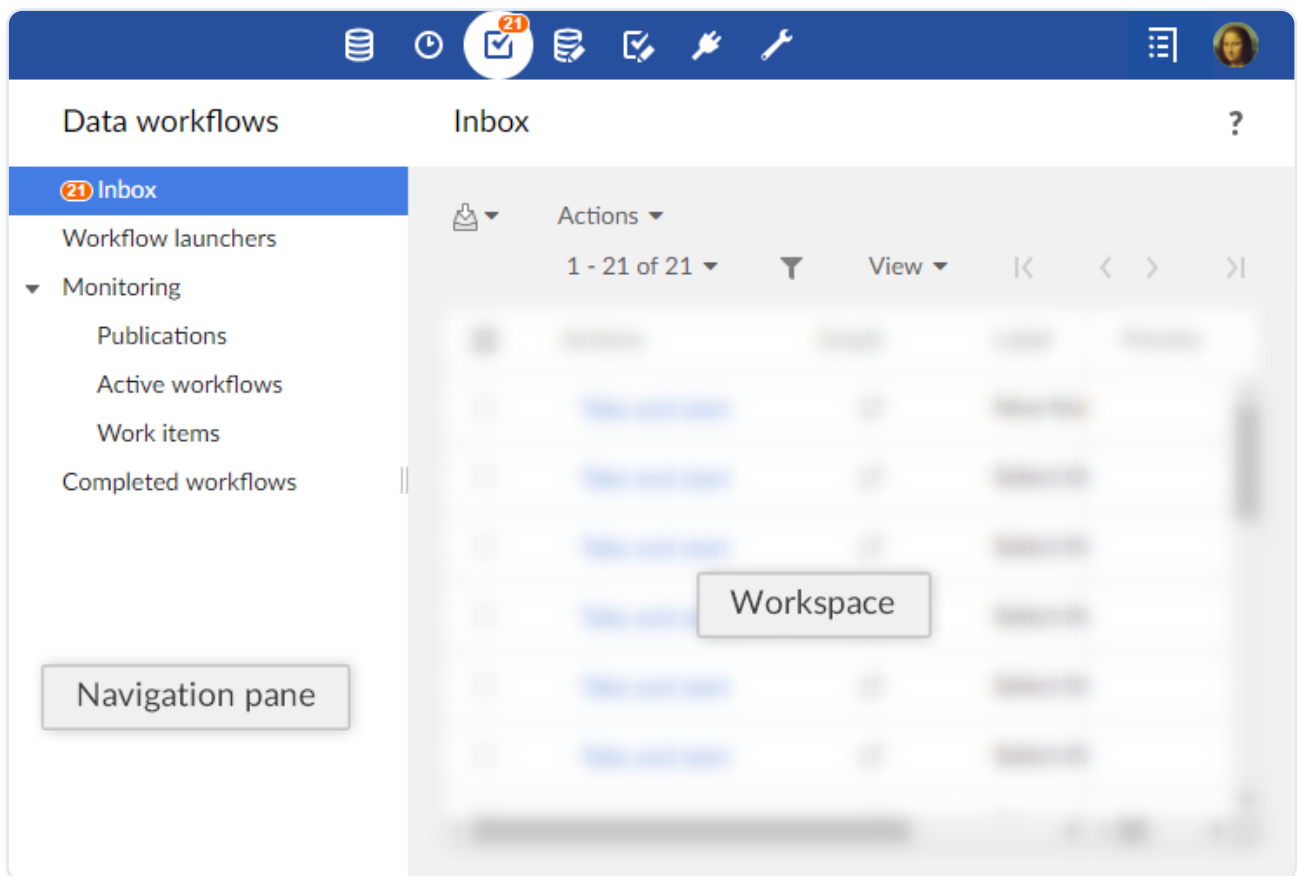
ユーザーガイド [目次](#)

データワークフロー領域のユーザーインターフェイスの使用

- インターフェイス内のナビゲート
- ナビゲーションルール
 - 作業項目の受信トレイ
 - ワークフローランチャー
- カスタムビュー
- 特定の列
- ビュー内のアイテムのフィルタリング
- ワークフローの進行状況ビュー

インターフェイス内のナビゲート

データワークフロー機能は、TIBCO EBX®ユーザーインターフェイスのデータワークフロー領域にあります。



注意:

この領域は、「詳細パースペクティブ」または特別に構成された観点から許可されたユーザーのみが使用できます。許可されたユーザーのみがこれらのインターフェイスにアクセスできます。

ナビゲーションペインは、いくつかのエントリーで構成されています。これらのエントリーは、関連するグローバル権限に従って表示されます。さまざまなエントリーは次のとおりです。

作業項目の受信トレイ	定義されたタスクを実行する必要がある、割り当てられた、または提供されたすべての作業項目。
ワークフローランチャー	ユーザー権限に応じて、データワークフローの起動を許可されているワークフローモデル公開のリスト。

監視	必要な表示権限があるデータワークフローのビューを監視します。
公開	必要な閲覧権限を持っている公開。追加の管理権限がある場合は、このビューから特定の公開からデータワークフローを起動する機能を無効にすることもできます。
アクティブなワークフロー	必要な表示権限を持っている実行中のデータワークフロー。追加の管理者権限がある場合は、データワークフローのステップの再生や、このビューからのデータワークフローの実行の終了などのアクションを実行することもできます。
作業項目	必要な表示権限がある作業項目。追加の管理者権限がある場合は、このビューから特定のユーザーまたはロールに作業項目を割り当てるなど、作業項目の管理に関連するアクションを実行することもできます。
完了済みワークフロー	実行が完了したデータワークフロー。必要な表示権限があります。データワークフローの実行履歴を表示できます。追加の管理権限がある場合は、このビューからリポジトリから完了したワークフローをクリーンアップすることもできます。

注意:

各セクションには、ポータル統合などのWebコンポーネントを介して、またはJava APIの `ServiceKey` クラスを使用してプログラムでアクセスできます。

ナビゲーションルール

作業項目の受信トレイ

デフォルトでは、作業項目が実行されると、作業項目の受信ボックスが表示されます。

この動作は、次のステップの進捗戦略に従って変更できます。これにより、作業項目の受信ボックスに戻らずに、複数のステップを連続して実行できます。

ワークフローモデリングの[ワークフローステップの進捗戦略](#)を参照してください。

ワークフローランチャー

デフォルトでは、ワークフローが起動されると、ワークフローランチャーテーブルが表示されます。

この動作は、モデル構成に従って変更できます。これにより、ワークフローランチャーテーブルを表示せずに最初のステップを直接開くことができます。

ワークフローモデリングの[最初のワークフローステップの自動開始](#)を参照してください。

カスタムビュー

ワークフローテーブルのビューを定義し、関連するすべてのメカニズム（公開を含む）の恩恵を受けることができます。

ワークフローテーブルビューを作成および管理するためのアクセス許可は、データテーブルビューのアクセス許可と同じです。したがって、この機能を利用するには、[ワークフロー管理](#) > [ワークフロー](#)を選択して、セクションの権限を変更する必要がある場合があります。

詳細については、[ビュー](#)を参照してください。

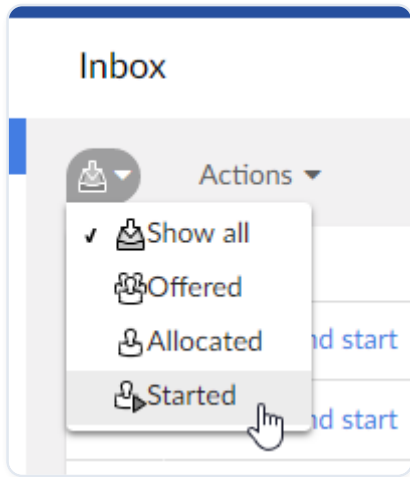
特定の列

デフォルトでは、特定の列は、その恩恵を受けることができるビューで非表示になっています（受信ボックス、作業項目の監視、アクティブなワークフローの監視、および完了済みワークフロー）。

特定の列を表示するには、カスタムビューを作成して適用する必要があります。次に、各作業項目またはワークフローについて、関連するワークフローモデルで定義されたマッチングが適用されます。列に対して式が定義されており、データコンテキスト変数が含まれている場合、これらの変数はワークフローデータコンテキストが更新されるたびに評価されます。

ビュー内のアイテムのフィルタリング

「作業項目の受信ボックス」などの特定のテーブルでは、特定の状態のエントリーのみを表示することで、テーブルに表示されるエントリーを絞り込むことができます。このようなビューには、状態を選択して対応する項目を表示できるメニューがあります。



ワークフローの進行状況ビュー

実行する作業項目を持つユーザーとして、またはデータワークフローモニターまたは管理者として、データワークフローのユーザーインターフェイス全体のテーブルの列に表示される  ボタンをクリックすると、データワークフロー実行の進行状況または履歴を表示できます。これにより、データワークフローの実行のインタラクティブな進行状況ビューを表示するポップアップが開きます。このビューでは、実行の全体的な進行状況を確認できるだけでなく、個々のステップをクリックしてその情報の詳細を表示できます。

ステップがワークフローモデリングで非表示として定義されている場合、それらは非管理者ユーザー（非ビルトイン管理者および非ワークフロー管理者）のワークフロー進行状況ビューで自動的に非表示になります。非表示のステップを表示するためのボタンを使用できます。ユーザーの選択（ステップの表示または非表示）は、ユーザーセッション中の公開によって、ユーザーによって保存されます。

新しいモード（単一の作業項目）を使用して実行されるユーザータスクの場合、該当する場合、単一の作業項目に関する主な情報（作業項目に関連付けられたユーザーのアバター、および作業項目に対して下された決定（承認または拒否））がワークフロー進行状況ビューに直接表示されます。

[ユーザーガイド目次](#)

作業項目

- [作業項目について](#)

- 作業項目の状態
- 参加者としての作業項目への取り組み
- 作業項目の優先度

作業項目について

作業項目は、ユーザータスクの一部として人間のユーザーが実行する必要のある作業の単位です。デフォルトでは、ワークフローモデルがユーザータスクを定義する場合、そのモデルのパブリケーションから起動されるデータワークフローは、ユーザータスクにリストされている参加者ごとに個別の作業項目を生成します。

以下も参照してください。:

- 概要

作業項目の状態

データワークフローが実行中にモデル化されたユーザータスクの作業項目を生成すると、作業項目はいくつかの可能な状態 (提供、割り当て、開始、完了) を通過します。

作業項目の作成

デフォルトモード

デフォルトでは、定義されたプロファイルのリストに関係なく、単一の作業項目が生成されます。

デフォルトでは、プロファイルのリストで1人のユーザーが定義されている場合、作成された作業項目は **割り当て済み** 状態になります。

デフォルトでは、その他の場合、作成された作業項目は **提供** 状態になります。

注意:

上記のデフォルトの動作は、ユーザータスクで定義されたプログラムによる拡張機能によって上書きできます。この場合、作業項目はプログラムで生成される可能性があり、必ずしもユーザータスクの参加者リストに基づくとは限りません。

レガシーモード

デフォルトでは、ユーザータスクの参加者として定義されたユーザーごとに、データワークフローは *割り当て済み* 状態の作業項目を作成します。

デフォルトでは、ユーザータスクの参加者として定義された役割ごとに、データワークフローは *提供* 状態の作業項目を作成します。

注意:

上記のデフォルトの動作は、ユーザータスクで定義されたプログラムによる拡張機能によって上書きできます。この場合、作業項目はプログラムで生成される可能性があり、必ずしもユーザータスクの参加者リストに基づくとは限りません。

作業項目の状態のバリエーション

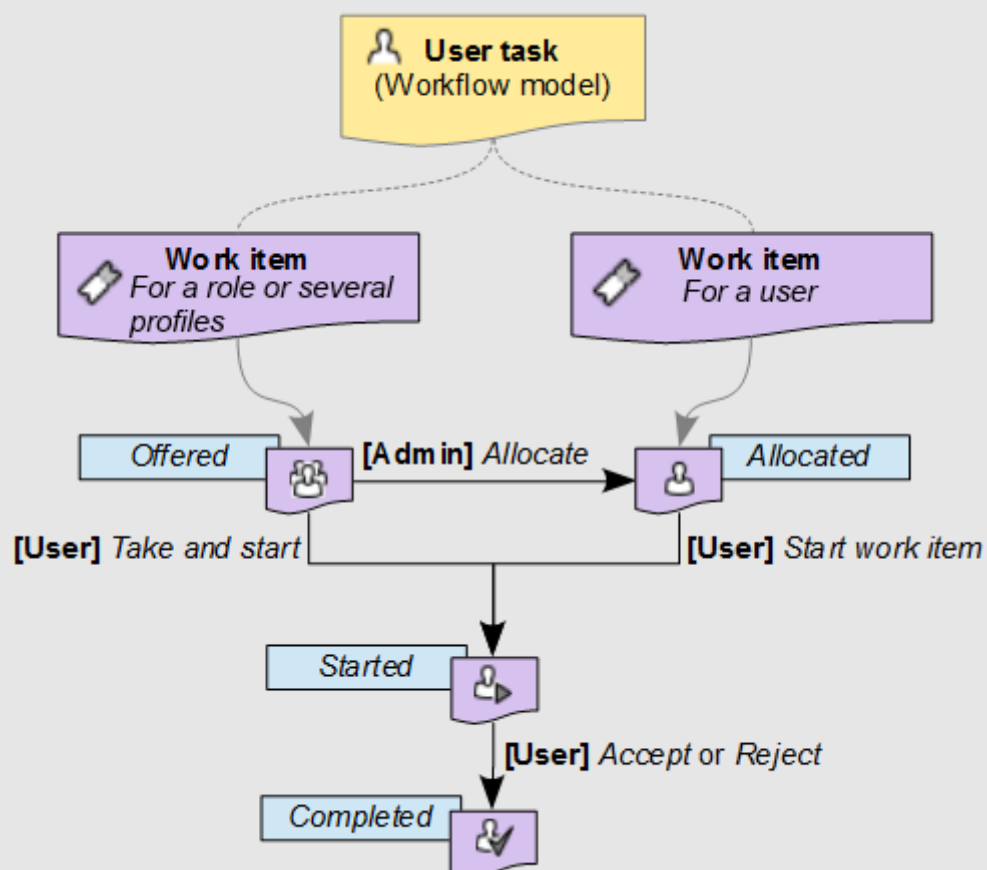
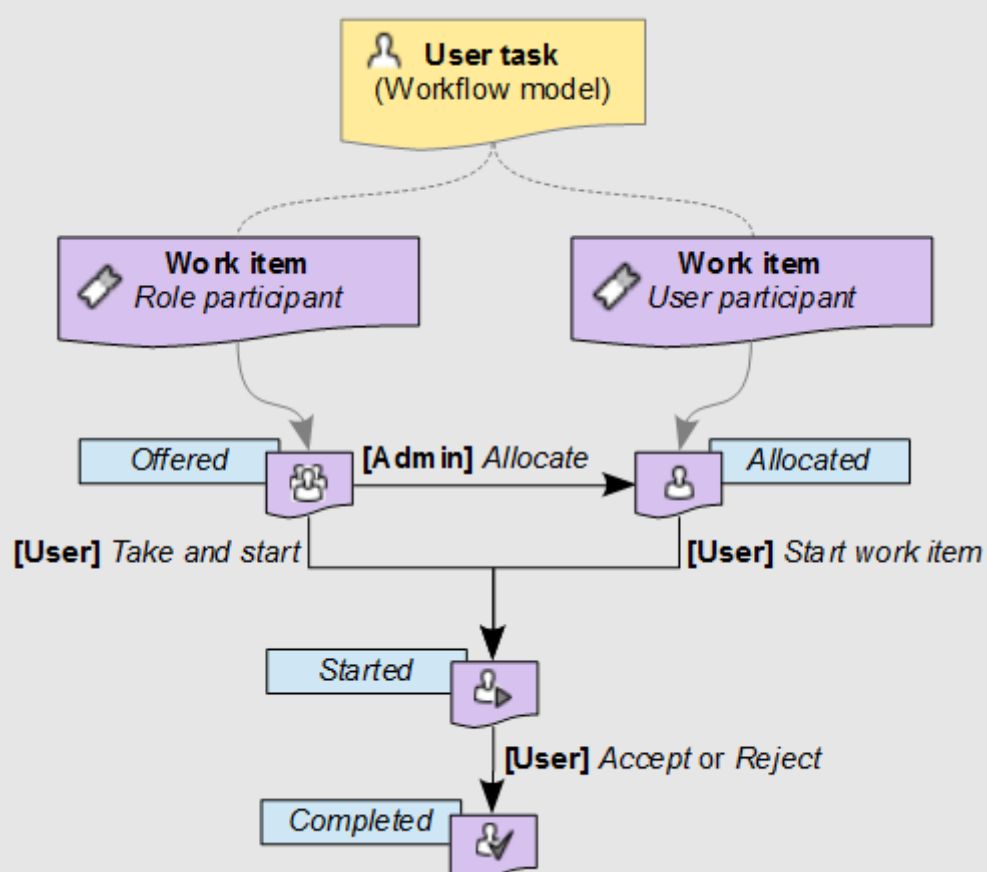
作業項目が *割り当て済み* 状態の場合、定義されたユーザーは、アクションを使用して、割り当てられた作業項目の作業を直接開始できます。作業項目の状態は *開始* になります。

作業項目が *提供* 状態の場合、作業項目が提供されるすべてのユーザーまたは役割のメンバーは、アクションを使用して作業項目を実行できます。作業項目の状態は *開始* になります。

ユーザーが提供された作業項目を要求する前に、ワークフロー割り当てマネージャーが介入して、作業項目を 1 人のユーザーに手動で割り当てることができます。これにより、作業項目が *割り当て済み* 状態に移行します。次に、そのユーザーがアクションを実行して作業項目の作業を開始すると、作業項目は *開始* 状態に進みます。

最後に、作業項目を開始したユーザーが割り当てられたアクションを終了した後、最後の承認または拒否アクションにより、作業項目が *完了* 状態に移行します。ユーザーが作業項目を完了すると、データワークフローは、ワークフローモデルで定義された次のステップに自動的に進みます。

作業項目の状態のダイアグラム



参加者としての作業項目への取り組み

個々のユーザーに関連するすべての作業項目 (提供または割り当てられている) は、作業項目の受信トレイに表示されます。作業項目の作業を開始するときに、データワークフローの他の参加者、およびワークフローの管理者とモニターに表示される関連コメントを追加できます。作業項目に取り組んでいる限り、戻ってこのコメントを編集できます。

作業項目に割り当てられたすべての必要なアクションを実行した後、**受入** または **拒否** ボタンのいずれかをクリックして、完了を通知する必要があります。これらの 2 つのボタンのラベルは、作業項目のコンテキストによって異なる場合があります。

作業項目の受信トレイに待機中の作業項目があるデータワークフローの現在の進行状況を確認するには、テーブルの列にある  ボタンをクリックします。ポップアップには、この時点までのデータワークフローと今後のステップのインタラクティブな進行状況ビューが表示されます。ステップをクリックすると、そのステップの詳細を表示できます。

注意:

ブラウザを閉じるかログアウトするなどして、開始された作業項目の途中で現在のセッションを中断した場合、現在の作業項目の状態が保持されます。作業項目に戻ると、中断したところから続行します。

作業項目の優先度

作業項目には優先度の値が含まれている場合があります。これは、未処理の作業項目の並べ替えとフィルタリングに役立ちます。作業項目の優先度は、個々の作業項目自体に固有ではなく、データワークフローのレベルで設定されます。したがって、データワークフローが緊急であると見なされる場合、関連するすべての未解決の作業項目も緊急であると見なされます。デフォルトでは、「非常に低い」から「緊急」までの 6 つの優先度レベルがありますが、使用可能な優先度レベルの視覚的表現と命名は、TIBCO EBX® リポジトリの構成によって異なります。

以下も参照してください。:

- [ユーザータスク \(用語集\)](#)

関連する概念:

- ユーザータスク

ユーザーガイド 目次

データワークフローの管理

- データワークフローの起動と監視
- データワークフローの管理

データワークフローの起動と監視

- データワークフローの起動
- 監視アクティビティ
- 作業項目の割り当ての管理

データワークフローの起動

ワークフローモデルに、公開からデータワークフローを起動する権限が付与されている場合は、ナビゲーションペインの エントリから新しいデータワークフローを作成できます。ワークフローモデル公開から新しいデータワークフローを作成するには、公開のエントリにある ボタンをクリックします。

オプションで、起動する新しいデータワークフローのローカライズされたラベルと説明を定義できます。

監視アクティビティ

ワークフローモデルの権限によってワークフロー監視用のユーザーまたはロールが構成されている場合は、現在実行中のデータワークフローの進行状況を追跡できます。ナビゲーションパネルのセクションから監視ビューにアクセスできます。追加のワークフロー管理権限がある場合は、これらのビューから関連するアクションを実行することもできます。

監視するデータワークフローの実行が完了すると、の下に表示され、実行履歴を確認できます。

作業項目の割り当ての管理

ワークフローモデルで、ユーザーまたはユーザーが属するロールの特別な割り当て管理権限が定義されている場合、関連するデータワークフローの実行中に、作業項目の割り当てに手動で介入することができます。この場合、作業項目に対して以下の 1 つ以上のアクションを実行できます。

ナビゲーションペインの セクションで を選択します。実行できるアクションは、作業項目の現在の状態に応じて、テーブル内の作業項目のエントリの メニューに表示されます。

割り当て	作業項目を特定のユーザーに割り当てます。 このアクションは、 <i>提供</i> 状態の作業項目で使用できます。
割り当て解除	<i>割り当て済み</i> 状態の作業項目を <i>提供済み</i> 状態に再設定します。
再割り当て	作業項目が割り当てられているユーザーを変更します。このアクションは、 <i>割り当て済み</i> 状態の作業項目で使用できます。

以下も参照してください。:

- [作業項目](#)
- [関連するデータワークフローの権限](#)

関連する概念:

- [ワークフローモデル](#)

[ユーザーガイド](#) [目次](#)

データワークフローの管理

データワークフローに関連する管理アクティビティの権限が付与されている場合、関連するパブリケーション、アクティブなデータワークフロー、および作業項目は、ナビゲーションパネルの セクションのエントリの下に表示されます。これらの監視ビューから、テーブルエントリの メニューから管理タスクを直接実行できます。

注意:

ワークフローモデルで管理者権限が付与されると、パブリケーション、アクティブなデータワークフロー、作業項目など、データワークフローの実行に関連するすべての側面に対する監視権限が自動的に付与されます。

- データワークフロー実行の概要
- データワークフロー管理アクション
 - パブリケーションのアクション
 - データワークフローに対するアクション

データワークフロー実行の概要

データワークフローが起動すると、現在実行中のステップを示す トークン が作成され、ワークフローの開始時に配置されます。各ステップが完了すると、このトークンは、データワークフローのベースとなる公開のワークフローモデルで定義されている次のステップに進みます。

データワークフローの実行中の任意の時点で、トークンは次のいずれかに配置されます。

- スクリプトタスク。これは自動的に実行され、ユーザーの操作を必要としません。定義されたアクションの実行が終了すると、スクリプトタスクが完了します。
- ユーザータスク。ユーザーが手動で実行する 1 つ以上の作業項目を生成します。各作業項目は、ユーザーからの明示的な「受入」または「拒否」アクションによって完了し、包括的ユーザータスクの完了は、ワークフローモデルでユーザータスクに定義されたタスク終了基準に従って決定されます。
- データワークフローの実行における次のステップを決定するために自動的に評価される条件。
- サブワークフローの呼び出し。関連するサブワークフローを起動し、起動されたサブワークフローの終了を待ちます。
- 特定のイベントが受信されるまでワークフローを一時停止する待機タスク。

トークンは次の状態になります。

- ****実行：**トークンは、ワークフローモデルに基づいて、次のステップに進むプロセスです。
- ****実行中：**トークンは、スクリプトタスクまたは処理中の条件に配置されます。
- ****ユーザー：**トークンはユーザータスクに配置され、ユーザーアクションを待機しています。
- ****サブワークフローの待機中：**トークンはサブワークフローの呼び出しに配置され、起動されたすべてのサブワークフローの終了を待機しています。

- ****イベントの待機：****トークンは待機タスクに配置され、特定のイベントが受信されるのを待機しています。
- ****終了：****トークンがデータワークフローの最後に到達しました。
- ****エラー：****エラーが発生しました。

以下も参照してください。:

- [ワークフロー管理](#)

データワークフロー管理アクション

パブリケーションのアクション

ワークフローパブリケーションの無効化

特定のワークフローパブリケーションから新しいデータワークフローが起動されないようにするには、それを無効にすることができます。ナビゲーションペインから エントリを選択し、無効にするパブリケーションのエントリで > を選択します。

無効にすると、パブリケーションはユーザーの「ワークフローランチャー」ビューに表示されなくなりますが、進行中のすでに起動されているデータワークフローは引き続き実行されます。

注意:

パブリケーションを無効にすると、領域から再度有効にすることはできません。ビルトインリポジトリの「管理者」ロールを持つユーザーのみが、管理領域から無効なパブリケーションを再度有効にできますが、ワークフロー操作の整合性を確保することが重要であるため、テクニカルテーブルを手動で編集することは一般的に推奨されません。

ワークフローパブリケーションの非公開

ワークフローパブリケーションが不要になった場合は、公開を解除することで、領域のビューから完全に削除できます。そうするために、

1. [ワークフローパブリケーションの無効化](#) で説明されているように、ワークフローパブリケーションを無効にして、ユーザーがワークフローパブリケーションから新しいデータワークフローを起動し続けることができないようにします。

2. 同じ公開テーブルのワークフローパブリケーションのエントリから > を選択して、ワークフローパブリケーションを非公開にします。

注意:

ワークフローパブリケーションの非公開を選択すると、このワークフローパブリケーションから起動された進行中のデータワークフローと関連する作業項目の終了とクリーンアップを確認するように求められます。データワークフローを強制的に終了した結果として失われたデータは、回復できません。

データワークフローに対するアクション

データワークフローのテーブルから、特定のデータワークフローのレコードのメニューからアクションを実行することができます。

ステップの再生

アクセス権の問題や利用できないリソースなど、ステップ中に予期しない障害が発生した場合は、データワークフロー管理者としてステップを「再生」できます。ステップを再生すると、関連する作業項目やサブワークフローを含む関連する実行環境がクリーンアップされ、トークンが現在のステップの先頭にリセットされます。

データワークフローの現在のステップを再生するには、テーブルのワークフローのエントリから > を選択します。

アクティブなデータワークフローの終了とクリーンアップ

現在進行中のデータワークフローを停止してクリーンアップするには、テーブルのワークフローのエントリから > を選択します。これにより、データワークフローの実行が停止し、データワークフローと、関連するすべての作業項目およびサブワークフローがクリーンアップされます。

注意:

このアクションは、「実行中」状態のワークフロー、および別のワークフローから起動されたサブワークフローでは使用できません。

注意:

ワークフロー履歴データは削除されません。

アクティブなデータワークフローの強制終了

現在進行中のデータワークフローを停止するには、テーブルのワークフローのエントリから > を選択します。これにより、データワークフローの実行が停止し、関連するすべての作業項目とサブワークフローがクリーンアップされます。

注意:

このアクションは、サブワークフロー、および最後のステップでエラーがブロックされたワークフローで使用できます。

注意:

ワークフロー履歴データは削除されません。

待機中のデータワークフローの強制再開

現在イベントを待機しているデータワークフローを再開するには、テーブルのワークフローのエントリから > を選択します。これにより、データワークフローが再開されます。このアクションを実行する前に、データワークフローが次のステップを実行できることを確認するために、データコンテキストを更新するのは管理者の責任です。

注意:

このアクションは、「イベントの待機中」状態のワークフローでのみ使用できます。

完成したデータワークフローのクリーニング

データワークフローの実行が完了すると、その履歴は、そのワークフローのモニターと管理者がビューで表示できます。完了したワークフローを削除するには、ワークフローに対してクリーンな操作を実行できます。これを行うには、テーブルのワークフローのエントリから > を選択します。

クリーンアップすると、ワークフローはビューに表示されなくなりますが、その履歴は領域から引き続き利用できます。

注意:

このアクションは、別のワークフローから起動されたサブワークフローでは使用できません。

以下も参照してください。:

- [ワークフロー管理](#)

データワークフローの優先度の変更

データワークフローが起動された後、ワークフローの管理者はその優先度レベルを変更できます。そうすることで、データワークフローによって作成されるすべての既存および将来の作業項目の優先度が変更されます。データワークフローの優先度レベルを変更するには、テーブルのワークフローのエントリから > を選択します。

以下も参照してください。:

- [関連するデータワークフローの権限](#)

[ユーザーガイド](#) [目次](#)

データサービス

- [データサービスの概要](#)
- [データサービス WSDL の生成](#)

データサービスの概要

- [概要](#)
 - [データサービスとは](#)

- リネージ
- 用語集
- データサービスエリアのユーザーインターフェイスの使用

概要

データサービスとは

データサービス は次のとおりです。

- TIBCO EBX® と相互作用する標準の Web サービス。

SOAP データサービスは、領域のデータモデルに基づいて動的に生成できます。

- EBX® との対話を可能にする RESTful サービス。

ビルトイン REST データサービスには OpenAPI インターフェイスがあり、返されたメタデータを通じて自己記述的でもあります。

これらは、ユーザーインターフェイスから利用できる機能の一部にアクセスするために使用できません。

以下も参照してください。:

- WSDL/SOAP
- REST

リネージ

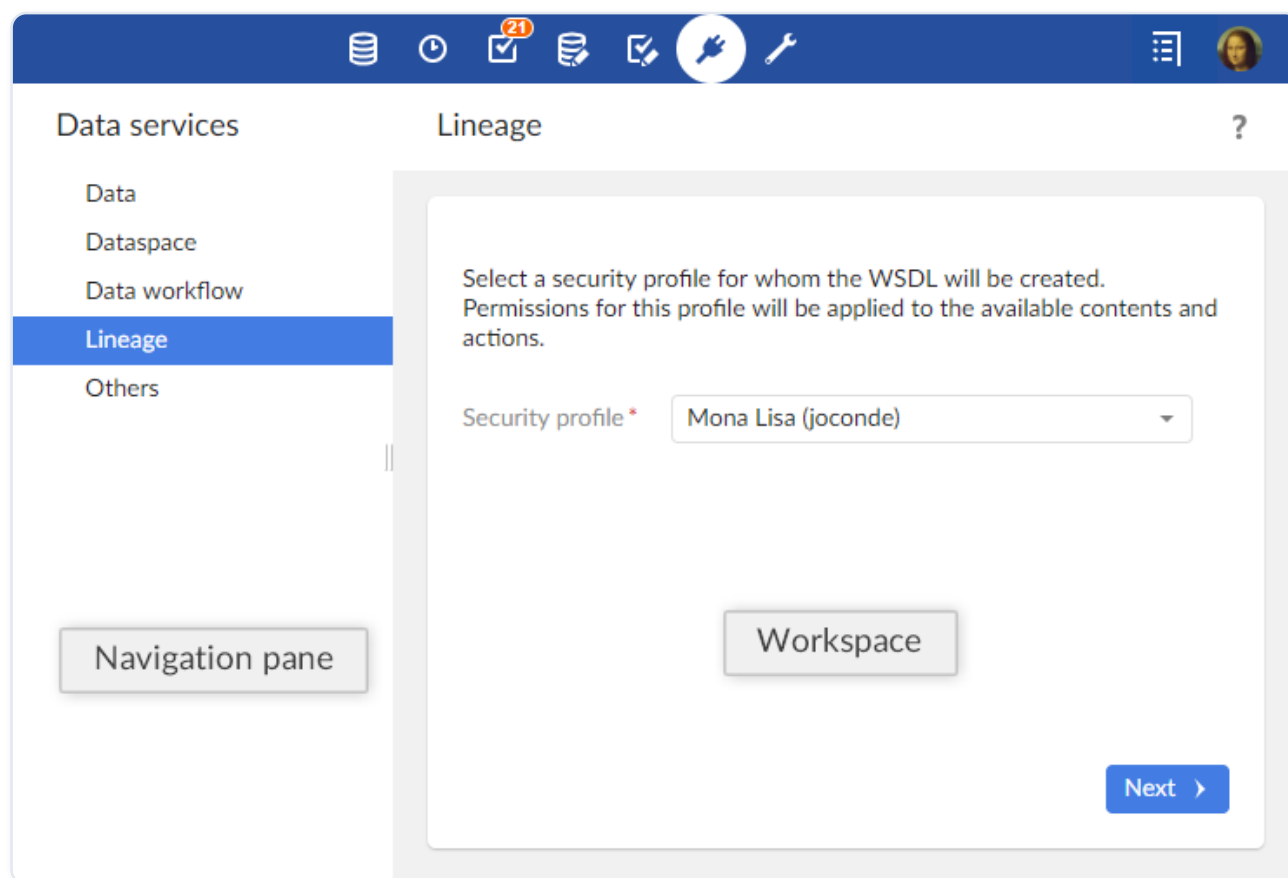
リネージ は、人間以外のユーザー、つまりデータサービスのユーザー権限プロファイルを確立するために使用されます。WSDL インターフェイスを使用してデータにアクセスする場合、データサービスはリネージを通じて確立されたアクセス許可プロファイルを使用します。

用語集

以下も参照してください。:

- データサービス

データサービスエリアのユーザーインターフェイスの使用



注意:

この領域は、で許可されたユーザーのみが使用できます。

関連する概念:

- データスペース
- データセット
- データワークフロー
- はじめに

データサービス WSDL の生成

- データ操作作用の WSDL の生成
 - データセットの操作
 - テーブルの操作
- データスペース操作作用の WSDL の生成
 - データスペースの操作
- データワークフロー操作作用の WSDL の生成
 - データワークフローの操作
- リネージの WSDL の生成
- 管理用の WSDL の生成
 - 管理のための操作
- デフォルトディレクトリを変更するための WSDL の生成
 - デフォルトディレクトリでの操作

データ操作作用の WSDL の生成

データにアクセスするための WSDL を生成するには、領域のナビゲーションパネルで **を選択し、ウィザードの手順に従います。**

1. WSDL をデータセットレベルでの操作作用にするか、テーブルレベルでの操作作用にするかを選択します
2. 操作が実行されるデータスペースとデータセットを特定します
3. 操作が許可されているテーブルと、許可されている操作を選択します
4. **WSDL のダウンロードボタン**をクリックして、生成された WSDL ファイルをダウンロードします

データセットの操作

次の操作は、データセットレベルでの操作作用に生成された WSDL を使用して実行できます。

- データスペースまたはスナップショットのデータセットコンテンツの選択

- データスペースまたはスナップショットの間のデータセット変更の取得
- レプリケーションユニットの更新

テーブルの操作

次の操作は、選択されている場合、テーブルレベルでの操作用に生成された WSDL を使用して実行できます。

- レコードの挿入
- レコードの選択
- レコードの更新
- レコードの削除
- レコードのカウント
- データスペースまたはスナップショット間の変更の取得
- 認証情報の取得
- データセット内のテーブルに対する複数操作の実行

以下も参照してください。:

- HTTP プロトコルからの WSDL のダウンロード
- データモデルから生成された操作

データスペース操作用の WSDL の生成

データスペースレベルの操作用の WSDL を生成するには、領域のナビゲーションパネルで を選択します。生成された WSDL はすべてのデータスペースに共通であるため、追加情報は必要ありません。

WSDL のダウンロードボタンをクリックして、生成された WSDL ファイルをダウンロードします。

データスペースの操作

次の操作は、データスペースレベルでの操作用に生成された WSDL を使用して実行できます。

- データスペースの作成
- データスペースを閉じる

- スナップショットの作成
- スナップショットを閉じる
- データスペースのマージ
- データスペースのロック
- データスペースのロック解除
- データスペースまたはスナップショットを検証
- データセットの検証

以下も参照してください。:

- HTTP プロトコルからの WSDL のダウンロード
- データセットとデータスペースの操作

データワークフロー操作の WSDL の生成

データワークフローを制御する WSDL を生成するには、領域から を選択します。生成された WSDL は特定のワークフロー公開に固有のものではないため、追加情報は必要ありません。

WSDL のダウンロード ボタンをクリックして、生成された WSDL ファイルをダウンロードします。

データワークフローの操作

- データワークフローの開始
- データワークフローの再開
- データワークフローの終了

以下も参照してください。:

- HTTP プロトコルからの WSDL のダウンロード
- データワークフローの操作

リネージの WSDL の生成

リネージの WSDL を生成するには、領域から を選択します。これは、領域の セクションで管理者によって定義された承認済みプロファイルに基づきます。

テーブルへのアクセスに使用できる操作は、[データ操作の WSDL](#) の場合と同じです。

リネージの WSDL を生成する手順は次のとおりです。

1. 権限が使用されるプロファイルを選択します。選択したユーザーまたはロールは、管理者によるリネージでの使用が許可されている必要があります
2. 操作が実行されるデータスペースとデータセットを特定します
3. 操作が許可されているテーブルと、許可されている操作を選択します
4. ボタンをクリックして、生成された WSDL ファイルをダウンロードします

以下も参照してください。:

- [リネージ](#)

管理用の WSDL の生成

このアクションは管理者のみが使用できます。

WSDL を生成するには、次の手順を実行します。

- ユーザーインターフェイスの管理
- システム情報の取得

領域から を選択します。

管理のための操作

- ユーザーインターフェイスを閉じる
- ユーザーインターフェイスを開く
- システム情報の取得

以下も参照してください。:

- HTTP プロトコルからの WSDL のダウンロード
- ユーザーインターフェイスの操作
- システム情報の操作

デフォルトディレクトリを変更するための WSDL の生成

このアクションは、管理者のみが使用でき、デフォルトのディレクトリを使用している場合にのみ使用できます。

デフォルトのディレクトリを更新する WSDL を生成するには、領域から [領域から](#) を選択します。

デフォルトディレクトリでの操作

テーブルへのアクセスに使用できる操作は、[データ操作の WSDL](#) の場合と同じです。

以下も参照してください。:

- HTTP プロトコルからの WSDL のダウンロード
- ディレクトリサービス

[ユーザーガイド 目次](#)

EBX® メタデータ管理

- [メタデータ管理の概要](#)
- [キーワード](#)
- [メタデータパースペクティブへのアクセス](#)
- [ユーザーインターフェイス](#)
 - [ユーザーインターフェイスの概要](#)
 - [ナビゲーションペイン](#)
 - [ナビゲーションペインの概要](#)
 - [メタデータカタログ](#)
 - [カタログの概要](#)

- アプリケーションの作成
- データエレメントの作成
- プロビジョニング
- 正規表現
- ガバナンス
 - ガバナンスの概要
 - ビジネス用語の定義
- メタデータ
 - メタデータセクションの概要
 - 資産の管理
- テクノロジー
 - テクノロジーの概要
 - インスタンスの作成
 - システムの作成
 - システムタイプの作成
 - インフラストラクチャーの作成
 - プロバイダーの作成
- プライバシー
 - プライバシーの概要
 - 処理アクティビティ
 - 同意
 - データ侵害
 - リクエスト
- ワークスペース
- ホームページ
- カスタムビュー
- 高度なデータサービス

- 高度なデータサービスの概要
- メタデータの収集
- 資産のサンプリング
- 資産分類
- リネージ
- コラボレーション
 - コラボレーションの概要
 - メタデータコメント
 - メタデータ評価
 - メタデータのタグ付け
 - コラボレーションワークフローによるメタデータの管理

メタデータ管理の概要

- メタデータとその重要性の理解
- 課題と解決策
- 対象ユーザー
- メタデータ管理へのアプローチ
- 次のステップ

メタデータとその重要性の理解

メタデータは、データ自体に関する情報 (データの内容、目的、編成方法など) を提供します。また、データに関連するビジネスおよび技術プロセス、およびさまざまなデータ間の接続に関する情報も含まれます。

メタデータを適切に管理することで、組織は業務、システム、作業の進め方を理解し、制御できるようになります。これは、保有するデータを最大限に活用するために重要であり、場合によっては法律で義務付けられています。

メタデータを適切に管理しないと、データが乱雑でわかりにくくなる可能性があります。それは、カタログのない図書館で特定の本を見つけようとするようなものです。どこから探し始めればよいのか、あるいはその本がそこにあるのかどうかさえわかりません。

要約すると、メタデータは、どのようなデータを持っているかを示し、それを効果的に検索して使用するのに役立ちます。そうでなければ、持っているすべての情報を理解するのは困難です。

課題と解決策

最も大きな課題の 1 つは、データ資産の説明、管理、マッピング、検出の方法を統一するメタデータ管理戦略を実装することです。戦略実装の成功には、ビジネス関係者間の戦略の相互理解、制御されたデータフロー、および高品質データの確実な定義が不可欠です。

EBX® のメタデータ管理機能により、エンドツーエンドの管理が保証され、データの管理、マスタリング、マッピング、交換が可能になります。EBX® メタデータ管理アプリケーションに含まれる機能を使用すると、次のことを実装できます。

- **データカタログ**：データソースを分類して情報システムを文書化するのに役立ちます。目標は、技術、ビジネス、データスペシャリストのプロファイルなど、さまざまな観点から情報システムを明確に把握することです。データを文書化してマッピングすると、潜在的なギャップを特定して埋めることができます。これにより、ユーザーは必要なときに必要なデータを見つけることができます。
- **ビジネス用語集**：ビジネス用語、その定義、用語間の関係を文書化できます。統一された用語集は、ビジネス概念の共通理解を提供することで、組織内のコミュニケーションを促進します。ビジネス用語集を作成することで解決される問題には、一貫性のないビジネス用語集、データの冗長性と重複、データの鮮度、データの品質などがあります。
- **メタデータ管理**：メタデータを使用してデータ資産のインベントリを作成し、評価および分析することで、データから価値を引き出すことができます。たとえば、追加のメタデータやデータセットの識別やキャプチャなど、潜在的な付加価値についてメタデータとデータを分析できます。つまり、メタデータ管理は、自分が持っているかもしれないデータを取得するのに役立ちます。メタデータ管理は、ビジネスソリューションや製品を操作する際のユーザーのエクスペリエンスを強化します。これにより、共同作業のエクスペリエンスが生まれ、さまざまなタイプのユーザーが作業に適した仮想環境を見つけられるようになります。

対象ユーザー

EBX® メタデータ管理アプリケーションは、最も多様な組織のニーズに対応し、チーム間のコラボレーションを促進します。ビジネスユーザー、データアーキテクト、データエンジニア、データスチュワード、アナリスト、アプリケーションアーキテクト、アプリケーション所有者などに、コラボレーションワークフローが提供されます。EBX® は、プロファイルを使用してロールベースのアクセスをサポートし、主要なデータエレメントの適切な公開を保証し、必要なコンプライアンスを提供します。

メタデータ管理へのアプローチ

EBX® を使用したメタデータ管理には、主に 2 つのアプローチがあります。

- ボトムアップの例：情報システムのマッピング、メタデータカタログの初期化、またはアプリケーションを遡及的に文書化したいアプリケーション所有者から始めます。
- トップダウンの例：アーキテクチャボードは、ビジネス/アプリケーション アーキテクチャから技術アーキテクチャに至るまで、情報システム地図作成プロジェクトを開始します。

次のステップ

ドキュメントで使用されている **主要な用語** を理解することが重要です。ビジネスユーザーの場合は、**ユーザーインターフェイス** コンポーネントの概要と、一般的な機能の使用方法を理解しておく必要があります。管理者の場合は、管理者が使用できる **構成オプション** を理解しておく役立ちます。

キーワード

次の表は、EBX® メタデータ管理アプリケーションの機能について説明する際によく使用される主要な用語を定義しています。

用語	説明
アプリケーション	データを活用、操作、または公開するソフトウェア、サービス、レポート、またはその他のドキュメント。アプリケーションはビジネスプロセスに役立ち、ルールの対象となる場合があります。
資産	参照および文書化できる構造化データのコンテンツまたはプロバイダー。システムインスタンスには資産が含まれたり公開されたりします。また、資産は他の資産で構成できます。たとえば、Excel ファイルは資産です。このファイルには、テーブル、列、行、セルを含むシートが含まれており、これらはすべて資産です。資産は、テクニカルフローを介して他の資産に供給したり、他の資産から供給されたりします。
ビジネスプロセス	特定の順序で完了する関連アクティビティまたはタスクの集合。顧客またはユーザー向けの製品またはサービスの形でビジネス価値を生み出します。

用語	説明
ビジネス用語	部門またはドメインで使用される概念を説明する単語またはフレーズ。ビジネス用語集は、ビジネス用語を管理および伝達します。
データエレメント	正確な意味またはセマンティクスを持つデータの原子単位。顧客や製品などの複雑なオブジェクトを表すことも、生年月日や名などの単純な値を定義することもできます。単純なデータエレメントを組み合わせることで、より複雑なエレメントを作成できます。たとえば、顧客は人物で構成され、人物は名とその他のデータエレメントで構成されます。データエレメントは、他のデータエレメントと関係を持つこともできます。
データセット	同じデータエレメントまたはデータエレメントのセットの母集団を定義する、同じ構造のデータのコレクション。データセットは、国を網羅したリストや、18 歳から 24 歳のフランス人顧客のリストにすることができます。これらの 2 つの例では、データセットと、それぞれ国または顧客である 1 つの高レベルデータエレメントとの間に自然なリンクがあります。ただし、技術的な便宜上、同じ構造を持つ異なるデータエレメントを表すことができる ERP からの製品エントリのリストなど、より複雑で不自然な場合もあります。
環境	アプリケーションは、さまざまな理由から、多くの環境内または環境上に存在することがあります。地理的な組織が原因で、非機能要件のためにアプリケーションを冗長化する必要がある場合があります。また、職務の分離によって多くの独立した環境を作成する必要があります。ビジネスまたは構造組織が原因である場合もあります。ただし、ほとんどの場合、これは、実稼働前にテストおよび認定する必要があります。アプリケーションのビルドと構成に必要です。環境は、そのアーキテクチャが依存する

用語	説明
	すべてのシステムインスタンスで構成されます。
フロー	2つのアプリケーション間でのデータの移動。データの交換には、ソースとターゲット間の多数の個別のフローが含まれます。論理レベルでは、技術的なフローはデータ交換を定義し、それを可能にします。さらに、フローの非機能要件と、それらを理解または構築するのに役立つ情報を指定します。
グループ	人のグループには、アプリケーションのユーザーと非ユーザーが含まれますが、グループのすべての人は、データセット所有者、システム管理者など、データの管理において何らかのロールを持つことが意図されています。グループの例としては、マーケティンググループ、ITグループなどがあります。このグループの人物を識別することで、システム、インスタンスなどの文書化されたオブジェクトの所有権が割り当てられます。さまざまなオブジェクトの管理に関連する責任もグループにリンクされます。
インフラストラクチャー	システムをサポートする基盤またはフレームワーク。データの保存、処理、分析、交換をサポートする物理または論理リソースで構成されます。インフラストラクチャーは、ほとんどの場合、オペレーティングシステムを実行する物理サーバーまたは仮想サーバーです。
インスタンス	特定のシステムの具体的なインストールには、ソフトウェア、ファイルシステム、サービスプロバイダーが提供するサービス、内部サービス、またはデータ資産プロバイダーのその他のコンテナやアプリケーションのコンポーネントのインストールが含まれます。これはインフラストラクチャー上で実行されます。
リンク	これは、データ資産内の2つのデータエレメント間の関係を具体化します。次に、2つの資産間のリンクが包含システム内でどのように

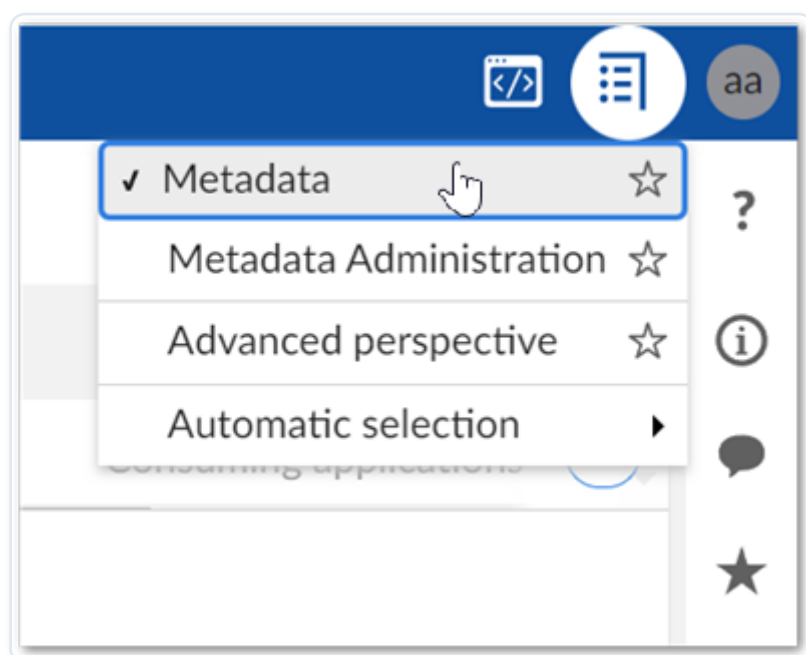
用語	説明
	確保されるかを定義します。これは、データベースの外部キー、ハイパーリンク、または単なるコードやその他の技術的表現にすることができます。
関係	2つのデータエレメントをリンクすると、一方が他方の表現になります。データエレメントが関連している理由と条件を定義します。ルールを適用することもできます。同じ2つのデータエレメントの間には、多くの関係が存在する可能性があります。「会社」は、別の「会社」の「サプライヤー」と「顧客」の両方になることができます。これらの2つのケースには異なるルールを適用できるため、複数の条件を持つ1つの関係ではなく、2つの異なる関係を作成できるのは興味深いことです。
Role	これは、EBX® 内のユーザーロールの定義ではなく、情報システム内のデータに対するロールです。ロールは、必ずしもデータガバナンスやデータ管理に対してアクティブである必要はありませんが、たとえば、受動的な通知ロールを定義できます。RACI マトリックスの各エレメントは、一連の人物グループに与えられるロールとして定義できます。
ルール	特定のデータカテゴリのいくつかの側面を定義または制限して、ビジネスデータの動作を制御または影響します。これは、意思決定やプログラムまたはポリシーの管理に使用されます。
プロバイダー	1つまたは複数のシステムを提供する企業。ソフトウェアベンダー、システムインテグレーター、その他のアウトソーシング企業である場合もありますが、社内組織の場合もあります。
システム	アプリケーションの開発に関与するあらゆる技術コンポーネント、またデータを提供または使用するあらゆる技術コンポーネント。シ

用語	説明
	システムはインスタンスによって具体化されません。
テクニカルフロー	データは使用されることを目的としています。データは、発生源から中間業者を経由して最終目的地に到達し、その後、将来別の目的地に到達します。2つ以上のアプリケーション間のフローであるこの過程の各ステップは、テクニカルフローです。テクニカルフローは、2つの資産間の橋渡しを定義し、あらゆるレベルでターゲットのソースが何であることを伝えます。

メタデータパースペクティブへのアクセス

EBX® メタデータ管理アプリケーションとその関連機能には、パースペクティブからアクセスできます。このパースペクティブは、メタデータモジュールが展開され、管理者によってアクセスが許可されている場合にのみ使用できます。

メタデータ パースペクティブにアクセスするには、メインツールバーの右上にある **アイコン** をクリックし、**を選択**します。パースペクティブは管理者ロールでのみ使用できることに注意してください。



ユーザーインターフェイス

- [ユーザーインターフェイスの概要](#)

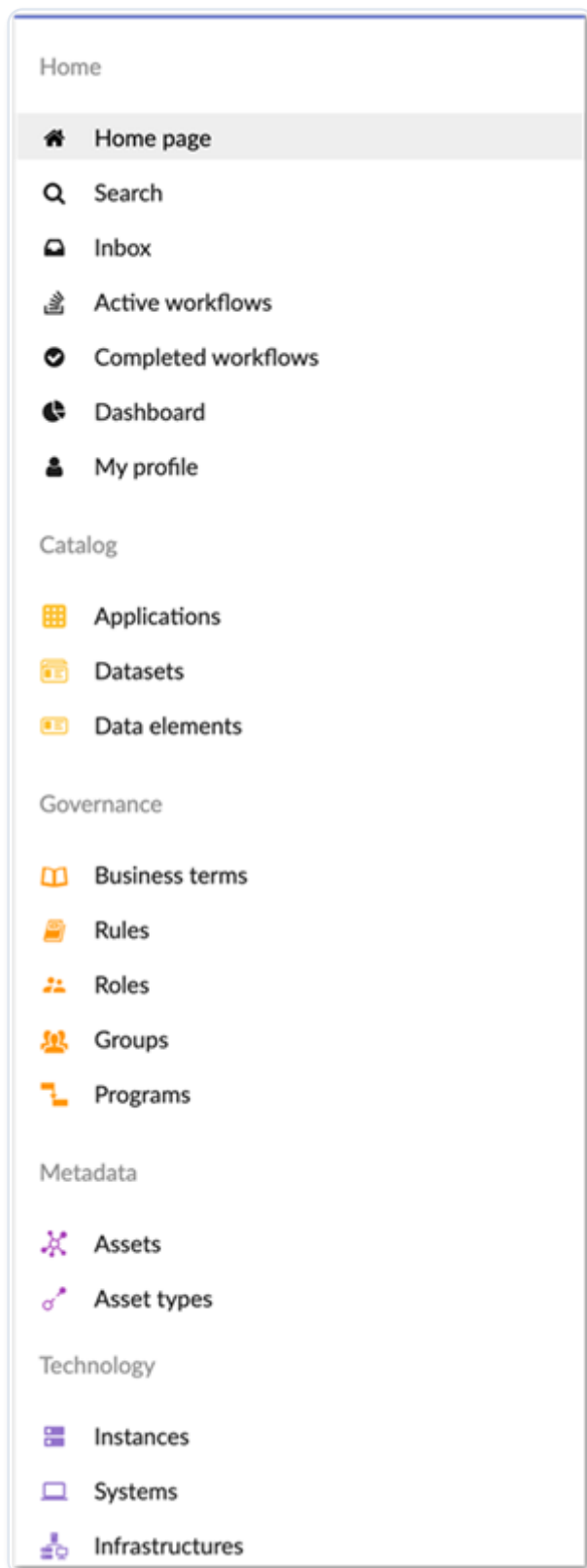
- ナビゲーションペイン
 - ナビゲーションペインの概要
 - メタデータカタログ
 - カタログの概要
 - アプリケーションの作成
 - データエレメントの作成
 - プロビジョニング
 - 正規表現
 - ガバナンス
 - ガバナンスの概要
 - ビジネス用語の定義
 - メタデータ
 - メタデータセクションの概要
 - 資産の管理
 - テクノロジー
 - テクノロジーの概要
 - インスタンスの作成
 - システムの作成
 - システムタイプの作成
 - インフラストラクチャーの作成
 - プロバイダーの作成
 - プライバシー
 - プライバシーの概要
 - 処理アクティビティ
 - 同意
 - データ侵害
 - リクエスト

- ワークスペース
- ホームページ
- カスタムビュー

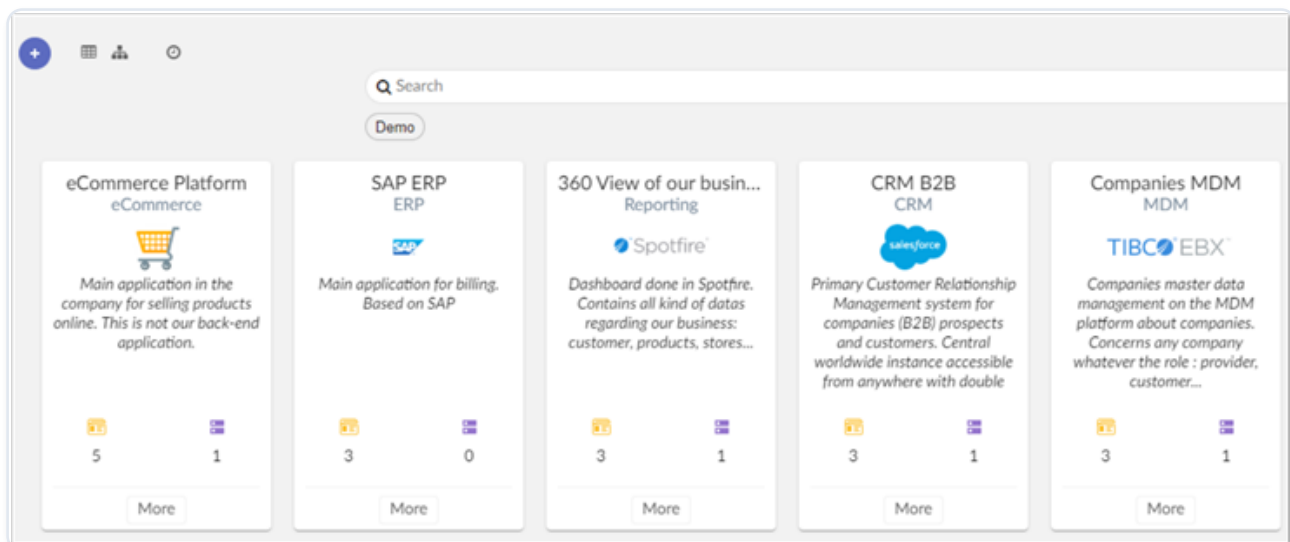
ユーザーインターフェイスの概要

メイン UI

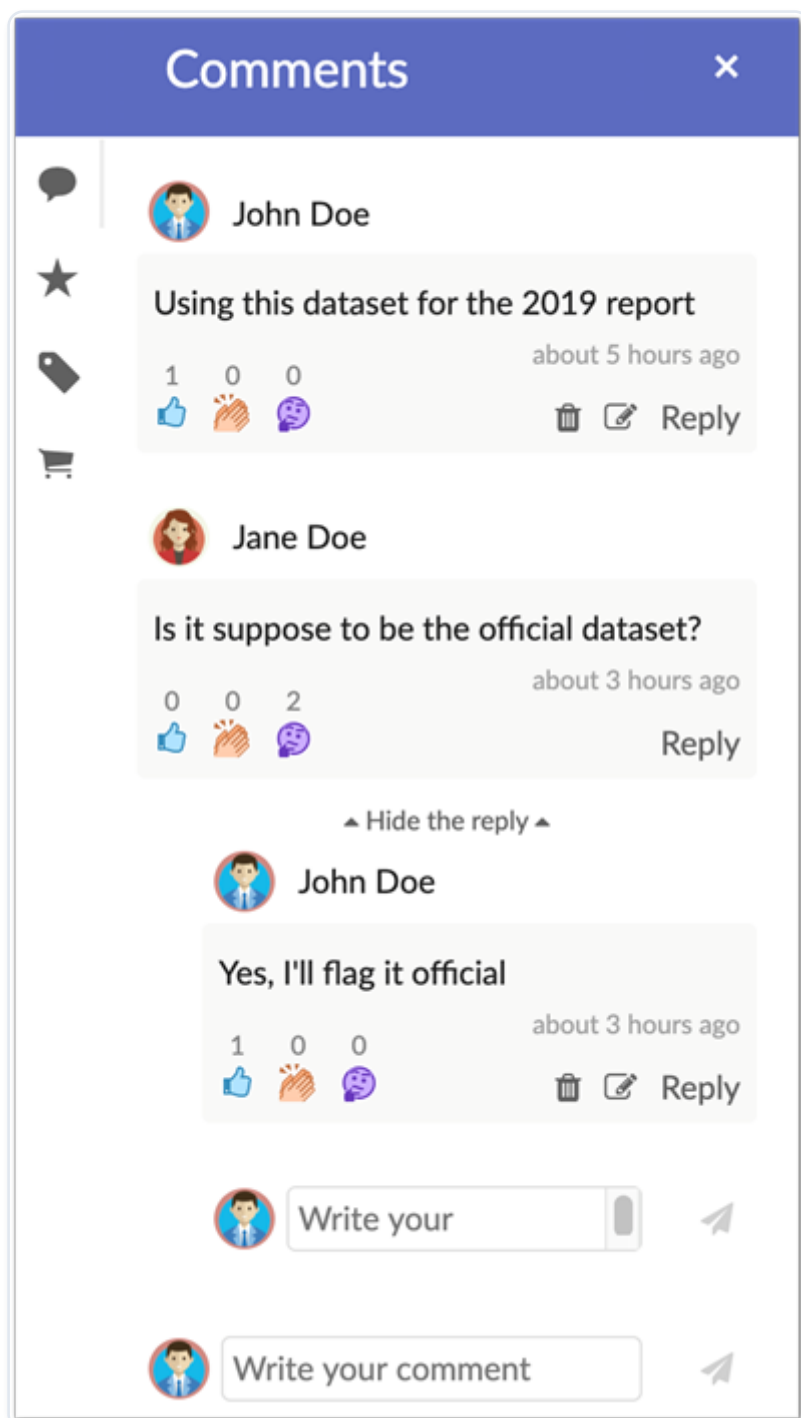
画面の左側はメインのナビゲーションペインです。



画面のメイン部分はワークスペースです。



ワークスペースの右側には、コメントや選択した資産に関する情報など、ワークスペースのコンテンツに応じたコンテキストサービスが含まれるサイドバーがあります。



ナビゲーションペイン

- ナビゲーションペインの概要
- メタデータカタログ
 - カタログの概要
 - アプリケーションの作成
 - データエレメントの作成
 - プロビジョニング

- 正規表現
- ガバナンス
 - ガバナンスの概要
 - ビジネス用語の定義
- メタデータ
 - メタデータセクションの概要
 - 資産の管理
- テクノロジー
 - テクノロジーの概要
 - インスタンスの作成
 - システムの作成
 - システムタイプの作成
 - インフラストラクチャーの作成
 - プロバイダーの作成
- プライバシー
 - プライバシーの概要
 - 処理アクティビティ
 - 同意
 - データ侵害
 - リクエスト

ナビゲーションペインの概要

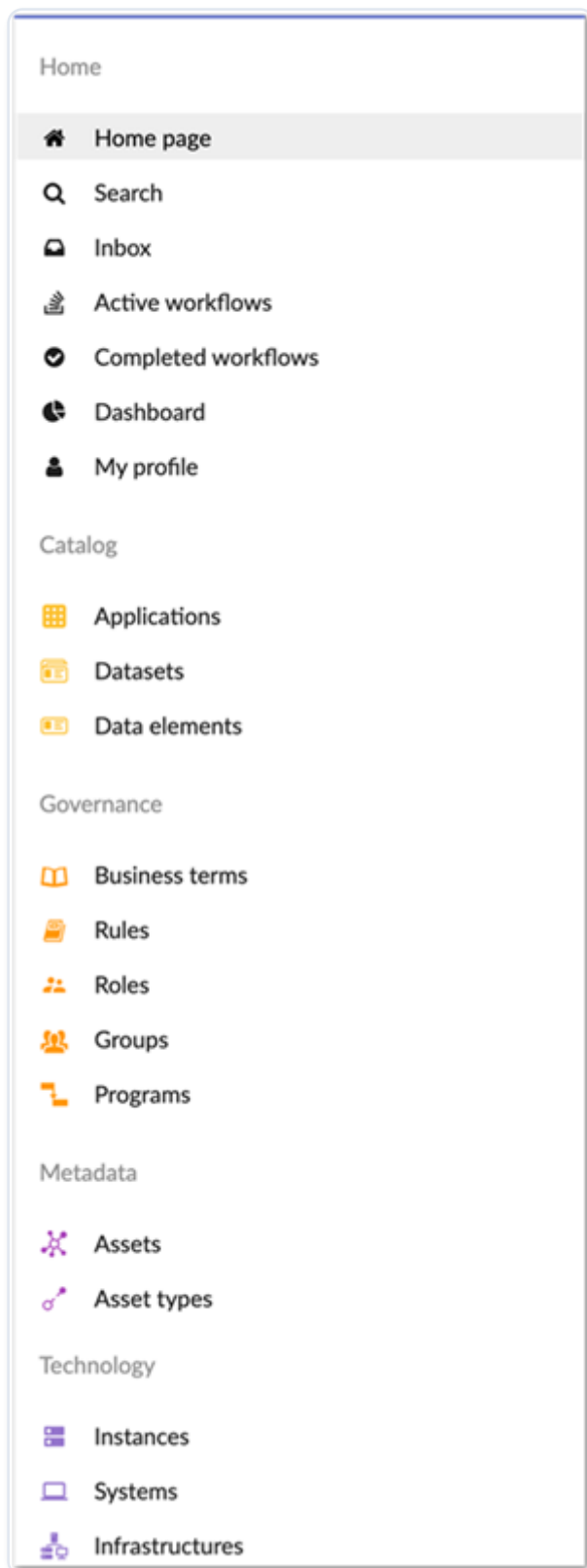
- 概要
 - ホームセクション
 - カタログセクション
 - ガバナンスセクション
 - メタデータセクション
 - テクノロジーセクション

- 。 プライバシーセクション

概要

ナビゲーションペイン は画面の左側にあります。ここから、プロファイルで利用可能なすべての機能にアクセスできます。管理者は権限設定を使用して、アクセス可能な機能を決定します。各セクションには、関連する機能が色分けされたグループで含まれています。

ナビゲーションペイン のサイズを変更するには、右の境界線をドラッグします。また、右上隅の矢印をクリックすると、表示を折りたたんでアイコンのみを表示できます。



ホームセクション

ホーム セクションのオプションは、特定の種類のメタデータには関連付けられていません。このセクションには、ホームページ、グローバル検索、ワークフローへのアクセス、ダッシュボード、詳細を変更できるプロフィールが含まれます。

カタログセクション

カタログ セクションには、アプリケーション、データセット、データエレメントなどのすべての論理説明がグループ化されています。論理説明は、エンティティをよりわかりやすく表現する方法です。論理説明は、まだ存在しないエンティティに適用できます。たとえば、アプリケーションを作成する前に説明したり、物理資産に格納されているデータ値や物理資産によって提供されるデータ値をデータセットに取り込む前にデータセットを文書化したりできます。

ガバナンスセクション

ガバナンス セクションには、さまざまなビジネス ドメインでデータがどのように管理されるかについてのドキュメントが含まれています。各ドメインには、独自のデータ関連の用語と役割があります。このセクションでは、ユーザー グループ、関連する役割、およびデータ権限も定義します。包括的なメタデータ管理プログラムは、データに関するあらゆるプログラム、イニシアチブ、またはプロジェクトを登録することを目的としています。

メタデータセクション

カタログに記述されているデータセットの具体化は、**メタデータ** セクションに記載されています。このセクションには、データを含むか提供する物理リソースである資産が含まれます。資産は、カタログ 資産の技術構造内の論理的な記述の具体化であり、そのタイプに応じて資産の技術構造とプロパティが保存されます。

テクノロジーセクション

テクノロジー セクションでは、メタデータ管理のより技術的な側面にアクセスできます。主要なコンポーネントの 1 つはインスタンスです。インスタンスは、データを公開する 1 つ以上のシステムで構成されます。また、システムとは、データ管理およびガバナンスのプロセスに含まれる任意のシステムのことです。システムは、ベンダー、外部組織、または内部組織であるプロバイダーによって提供されます。

プライバシーセクション

多くの規制当局が要求しているように、データプライバシーの管理では、処理活動と個人の同意の一覧を規定しています。また、個人データ所有者からのリクエストはすべて登録され、データ侵害は所有者に通知されることも求められます。リクエスト条件はすべて、それらを含む物理資産、それらを処理するシステム、およびそのネイチャーを定義する概念と結びついて、統一され、その使用に伴う規則に準拠していることが保証されます。

メタデータカタログ

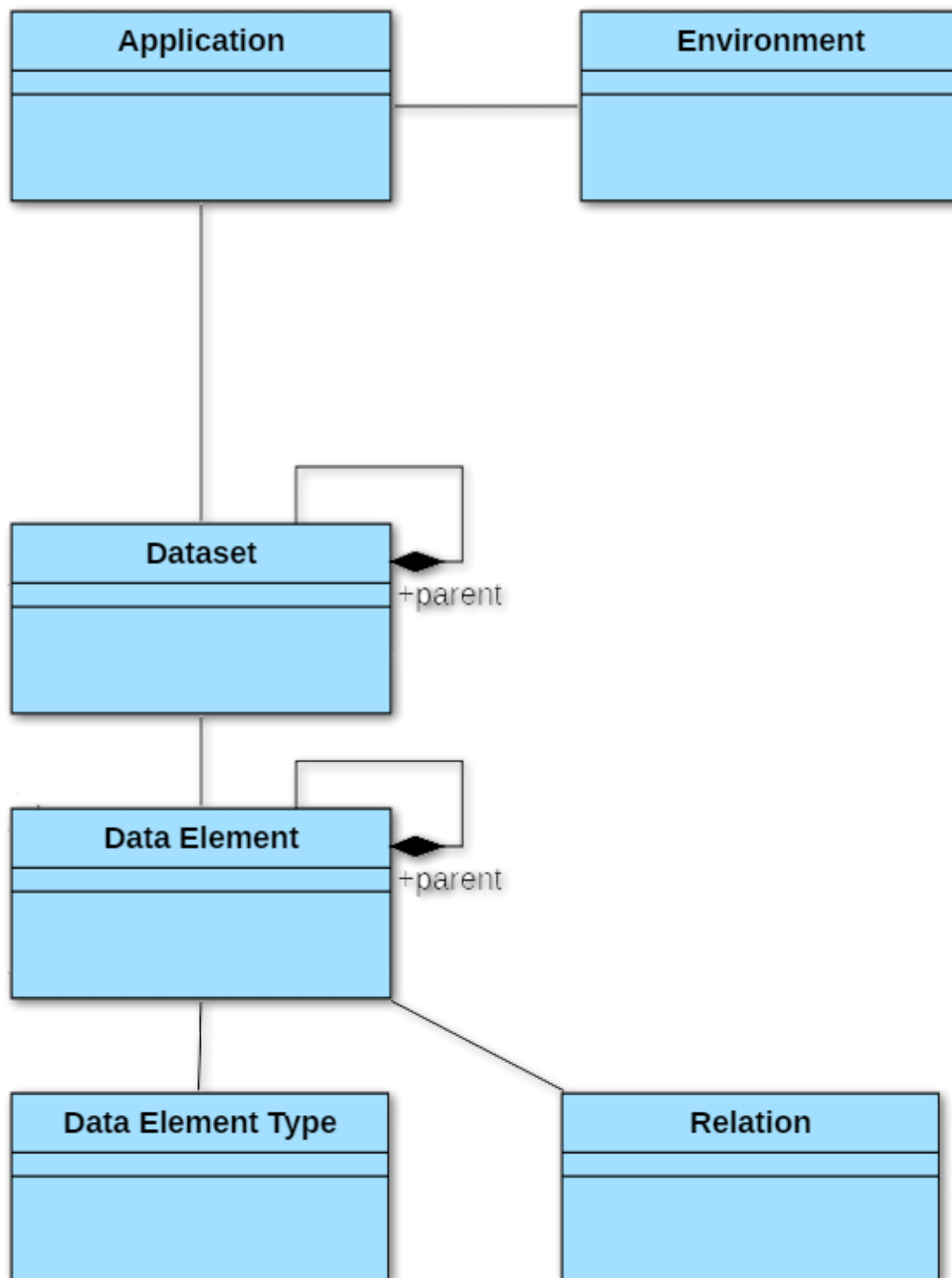
- [カタログの概要](#)
- [アプリケーションの作成](#)
- [データエレメントの作成](#)

- プロビジョニング
- 正規表現

カタログの概要

このセクションでは、アプリケーション、データセット、およびデータエレメントの論理的な説明について説明します。論理的な説明は、エンティティの普遍的で共通の説明です。これらの説明は、エンティティが作成される前に存在することができます。例：

- アプリケーションは、存在する前に *記述* することができます。
- データセットは、対応するデータ値が物理資産に保存される前、または物理資産によって提供される前に、*文書化* することができます。



アプリケーションの作成

- 概要
- 新しいアプリケーションの作成

- 新しいデータ保護レベル (CIA) エントリの作成
- 新しいアプリケーションタイプの作成

概要

メインコントロールパネルのメニューで、 を選択してアプリケーション画面にアクセスします。既存のすべてのアプリケーションをリストしたメイン ページが表示されます。

新しいアプリケーションの作成

左上隅の アイコンを選択すると、必要な情報を入力してアプリケーションを作成できます。疑問があるプロパティの説明を表示するには、ツールヒントを使用します。アプリケーションの作成が完了したら、 を選択します。

新しいデータ保護レベル (CIA) エントリの作成

アプリケーションに適用する定義済みのデータ保護レベル構成を作成できます。アプリケーションを作成または更新するときに、メニューから を選択します。

The screenshot shows a web-based configuration form for an application. The form is organized into a table-like structure with labels on the left and input fields on the right. The fields are as follows:

- Identifier:** [auto-incremented value] (with a key icon)
- Label:** My Application (with a red asterisk icon)
- Description:** (large text area)
- Codename:** (text input)
- Type:** [not defined] (dropdown menu)
- Owner:** [not defined] (dropdown menu)
- Data protection level (CIA):** [not defined] (dropdown menu, currently open showing a '+ Create' button and a 'Selector' link with an external icon)
- Logo:** (image upload area)
- Documentation:** [not defined] (with 'Drive' and 'Upload' links)

On the right side of the form, there are several icons: a share icon, a refresh icon, a delete icon, and a close icon.

新しいアプリケーションタイプの作成

以下に示すように、メニューから を選択すると、新しいアプリケーションタイプを作成できます。

The screenshot shows a form for creating a new application type. The fields and their values are as follows:

- Identifier:** [auto-incremented value]
- Label:** CRM system
- Description:** Primary Customer Relationship Management (CRM) system. Central instance accessible from anywhere with two-factor authentication. No VPN connection required. Connect to this application to track sales operations and value events.
- Codename:** Primary CRM
- Type:** A dropdown menu is open, showing options: [not defined], CRM (selected), + Create, and Selector. The CRM option has an external link icon.
- Owner:** [not defined]
- Data protection level (CIA):** [not defined]
- Logo:** [not defined]

At the bottom of the form, there are two buttons: Drive and Upload, both with external link icons.

データエレメントの作成

- 概要
- 新しいデータエレメントの作成
- 新しいデータエレメントタイプの作成
- 新しいデータエレメントステータスの作成

概要

パースペクティブのメニューで、 を選択します。ワークスペースのフォーカスが変更され、既存のすべてのデータエレメントが表示されます。

新しいデータエレメントの作成

を選択して、新しいデータエレメントレコードを作成します。表示される画面で、フィールドを使用してデータエレメントに関する情報を提供できます。ツールヒントを使用して、疑問のあるプロパティの説明を表示します。アプリケーションの作成が完了したら、 を選択します。

新しいデータエレメントタイプの作成

データエレメントを作成するときに、メニューを使用して新しいデータエレメントタイプを作成できます。

Identifier 1

Label * Customer

Description A person who buys goods or services from a shop or business.

Documentation [not defined]
[Edit](#) [Upload](#)

Data element type [not defined]
 Attribute
 Object
 + Create Selector

Parent data element [not defined]

Logo

新しいデータエレメントステータスの作成

データエレメントを作成するときに、メニューを使用して新しいデータエレメントステータスを作成できます。

Status	[not defined]
Critical data	+ Create Selector 
Personal data category	[not defined]
Regular Expression	

プロビジョニング

- プロビジョニングワークフローの起動
- 最初のステップ: コンテンツ分析
- 第2ステップ: ビューエレメントの入力
- 第3ステップ: レビュー
- 第4ステップ: 承認
- 第5ステップ: ビューの確定
- 検証
- 公開されたビューの更新
 - 公開ビューへの受信フローの追加
 - 公開されたビューからの受信フローの削除

プロビジョニングワークフローの起動

プロビジョニングは、選択した資産またはデータセットを集約するビューを定義できるプロセスです。データセットの場合は、優先資産が使用されます。

含めるデータセットを選択し、ツールバーの アイコン (カート) をクリックしてプロビジョニングを開始します。

最初のステップ: コンテンツ分析

EBX® は、ショッピングカートの内容で何ができるかを表示します。

Shopping cart content analysis

The list of exploitable items from your shopping cart is displayed in the table below.

Assets are directly considered for provisioning when preferred asset are resolved from Datasets.

Item in cart	Corresponding asset name	Asset type	Supported	Enabled
Datasets > Active customers	-	-	-	-
Assets > orders	orders	DB Table	Yes	No
Assets > name_basics	name_basics	DB Table	Yes	Yes

By clicking on the start button you will be redirected to the declaration of the asset corresponding to the view to be published. It requires a name that will be used as publication name and an instance of TDV in which the view will be published.

エレメントを確認したら、を選択します。

第2ステップ: ビューエレメントの入力

Request provisioning

Name *

Owner *

Marketing

Instance

TDV - For Localhost test agent

Justification *

Need to access this data to build our yearly report up to date for 2020.

このページでは、ビューを作成するための重要な情報を入力する必要があります。このビューは、EBX® メタデータ管理アプリケーション 資産として作成されます。この資産には所有者が必要であり、ビューが作成される TDV のインスタンス内に配置されている必要があります。リクエストを承認するには、レビュー担当者と承認者による正当な理由が必要です。

利用可能な TDV のインスタンスは、エージェントの登録時に自動的に作成されるインスタンスです。

第3ステップ: レビュー

この手順は、管理者によってレビュー担当者が設定されている場合にのみ実行されます。

第4ステップ: 承認

この手順は、管理者によって少なくとも 1 人の承認者が設定されている場合にのみ実行されます。

第5ステップ: ビューの確定

すべてのエレメントが設定されたら、 を選択します。

Finalize provisioning > Report_Customers_2020

Send back to requester
Finalize

Main
Referenced assets
References
Incoming flows
Outgoing flows
Datasets

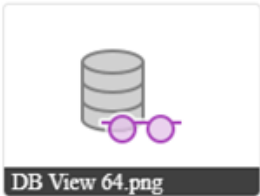
Identifier
67,393

Name
Report_Customers_2020

Label
Report_Customers_2020

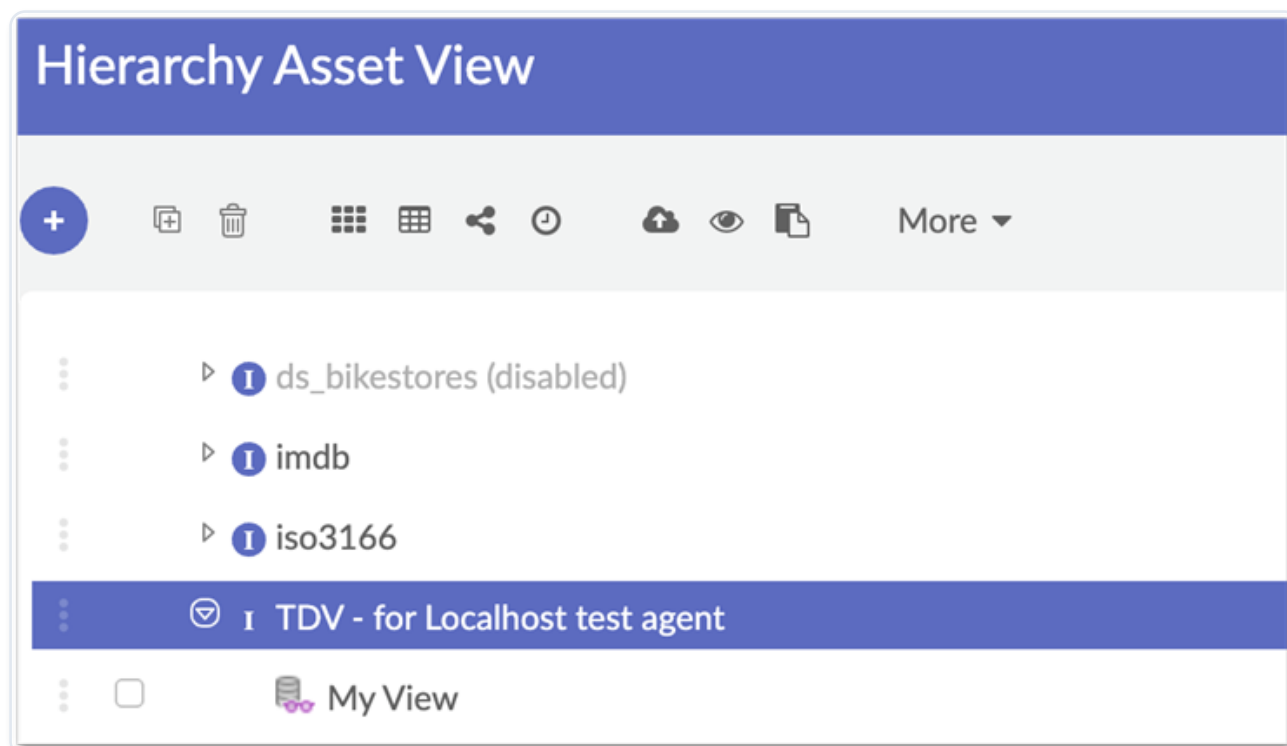
Description

Asset type
TDV Published View

Logo


検証

作成されたビューは、EBX® メタデータ管理アプリケーションの資産テーブルで確認できます。



TDV Studio でこの公開された TDV ビューの作成を確認することもできます。

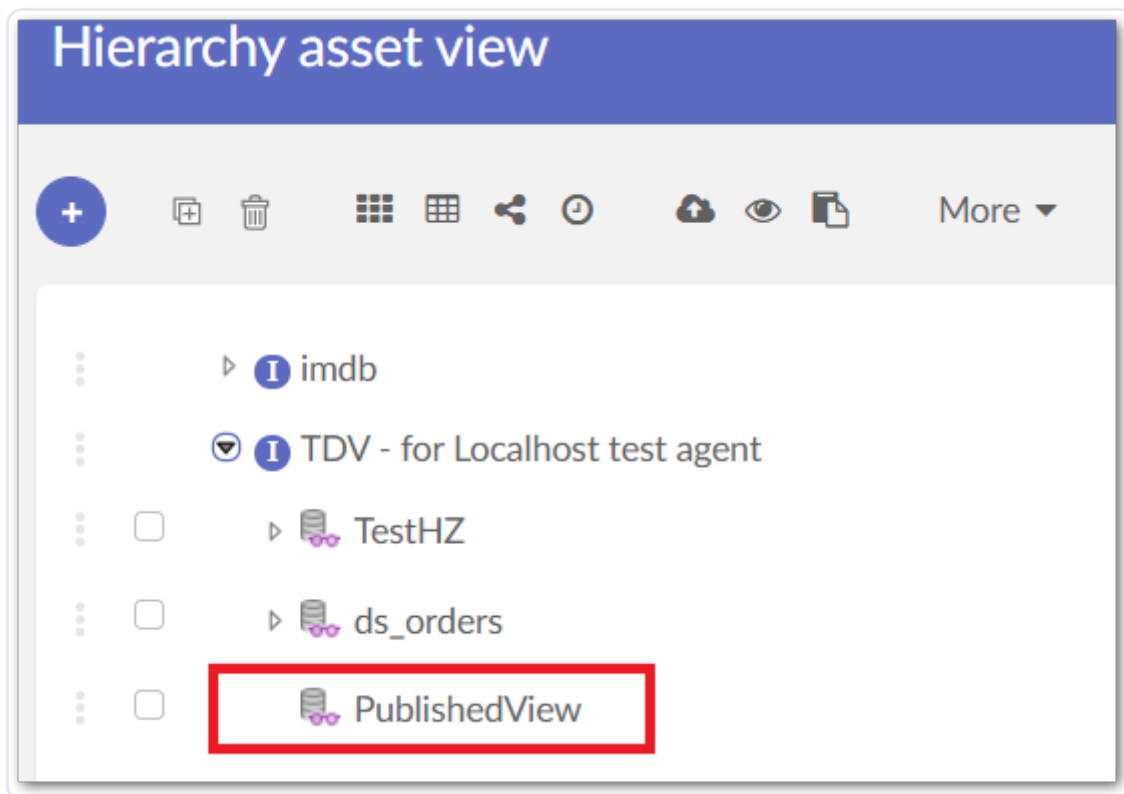
公開ビューの更新

プロビジョニングを使用して作成された TDV 公開ビューの場合、所有者は EBX®から更新操作を実行できます。

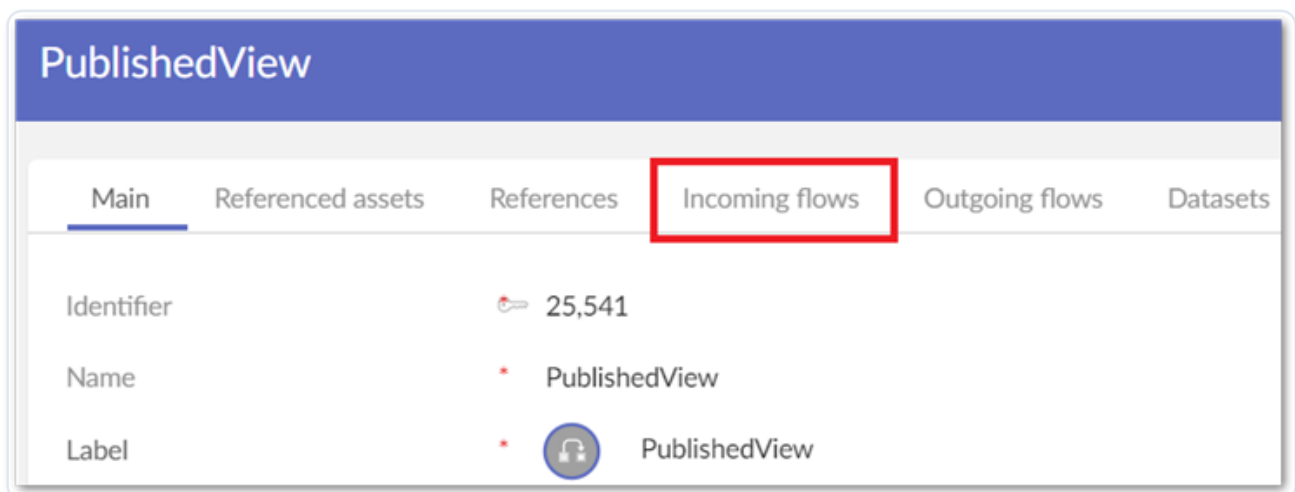
公開ビューへの受信フローの追加

TDV 公開ビュータイプの資産に着信フローを追加すると、TDV 内の対応するリソースの公開がトリガーされます。

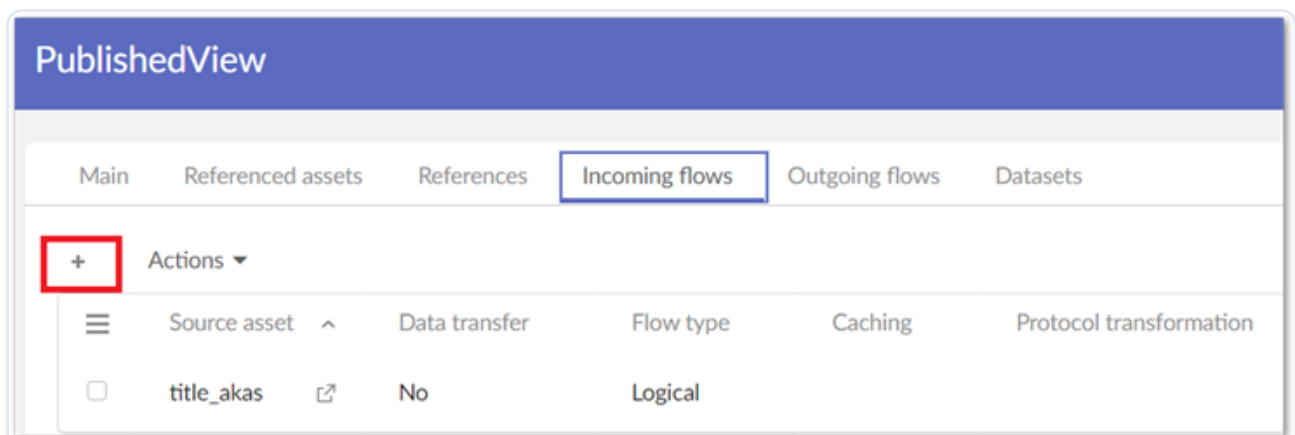
メイン コントロール パネルの メニューで、 を選択して、既存のすべての資産をリストするメイン ページを表示します。次に、編集する TDV 公開ビュー資産を開きます。



タブを選択します。



新しい技術フローを作成するには、左上隅の アイコンを選択します。



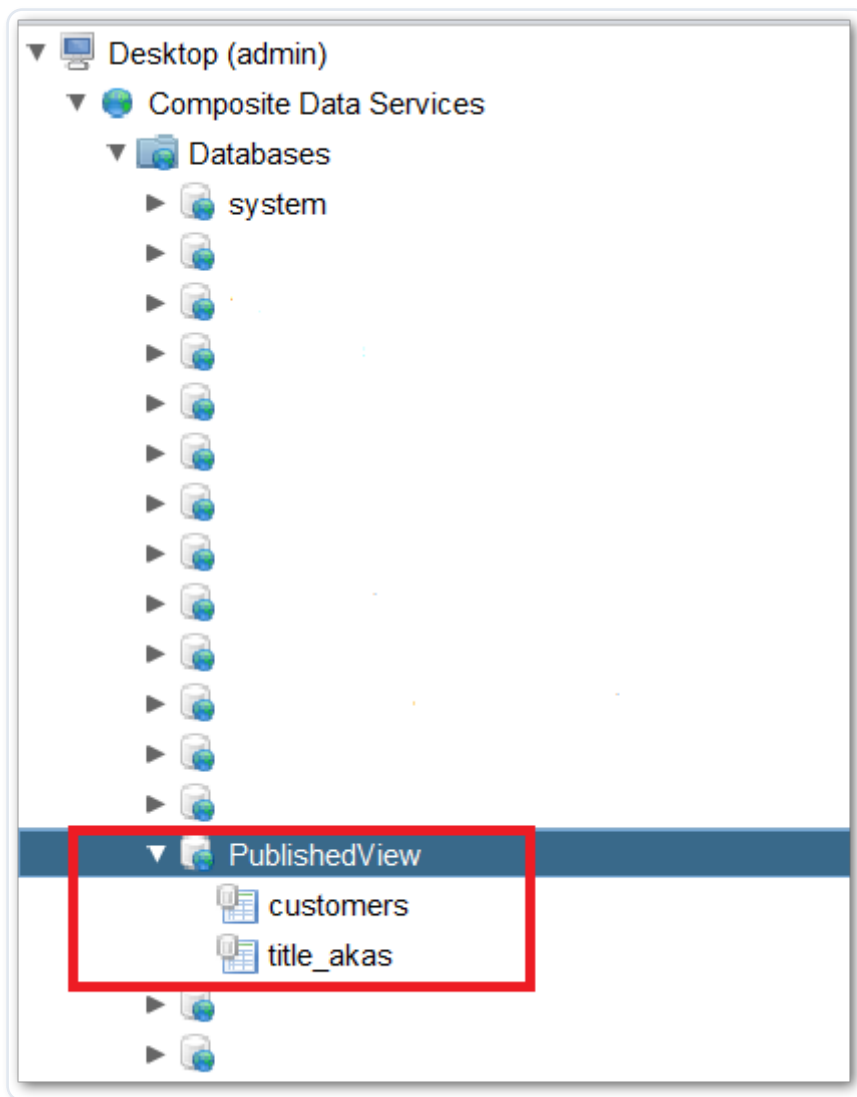
ソース資産を選択します。このリストには、収集の結果であり公開可能な資産のみが表示されます。

PublishedView > Technical flows : New record

Source asset	 customers 	
Target asset	 PublishedView 	
Data transfer	☐ Yes ☒ No 	
Flow type	Logical  	
Caching	☐ Yes ☒ No 	
Protocol transformation	☐ Yes ☒ No 	
Parent flow	[not defined] 	
Documentation	[not defined]	

 Edit  Upload

新しいフローの作成を検証するには、 を選択します。保存すると、TDV に対応するパブリケーションも作成されるため、保存には少し時間がかかる場合があります。



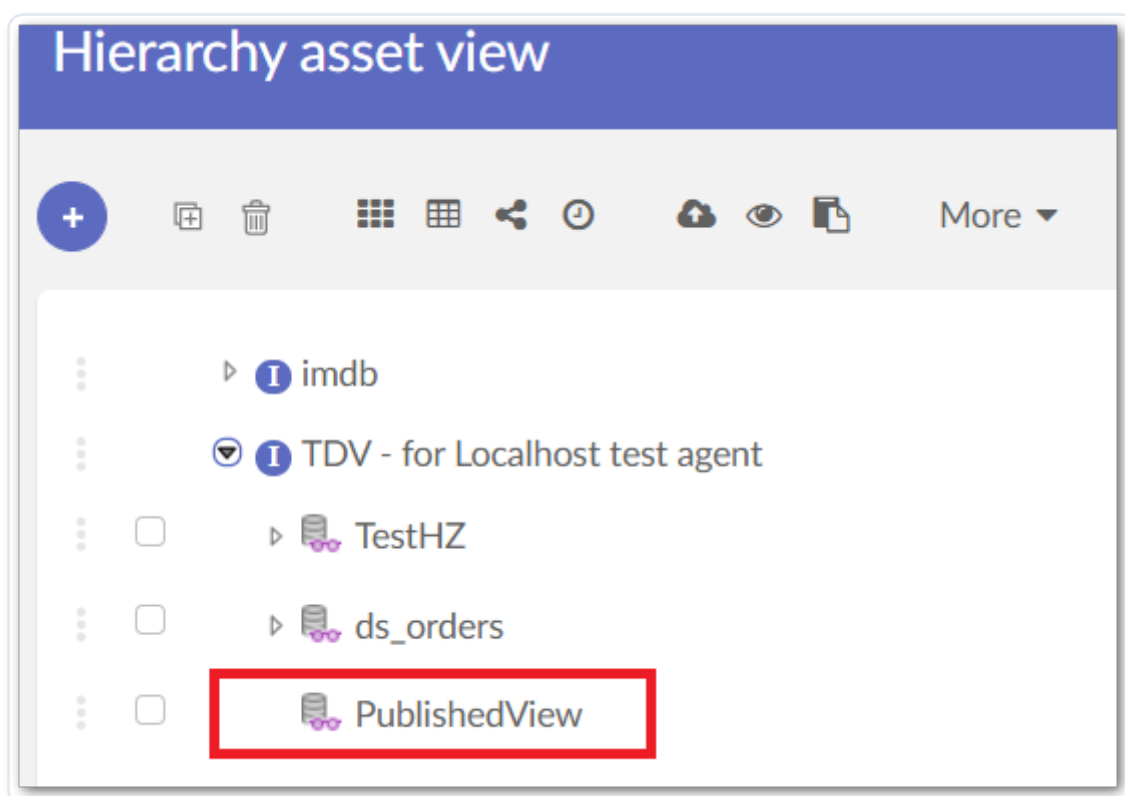
公開されたビューからの受信フローの削除

受信フローを削除

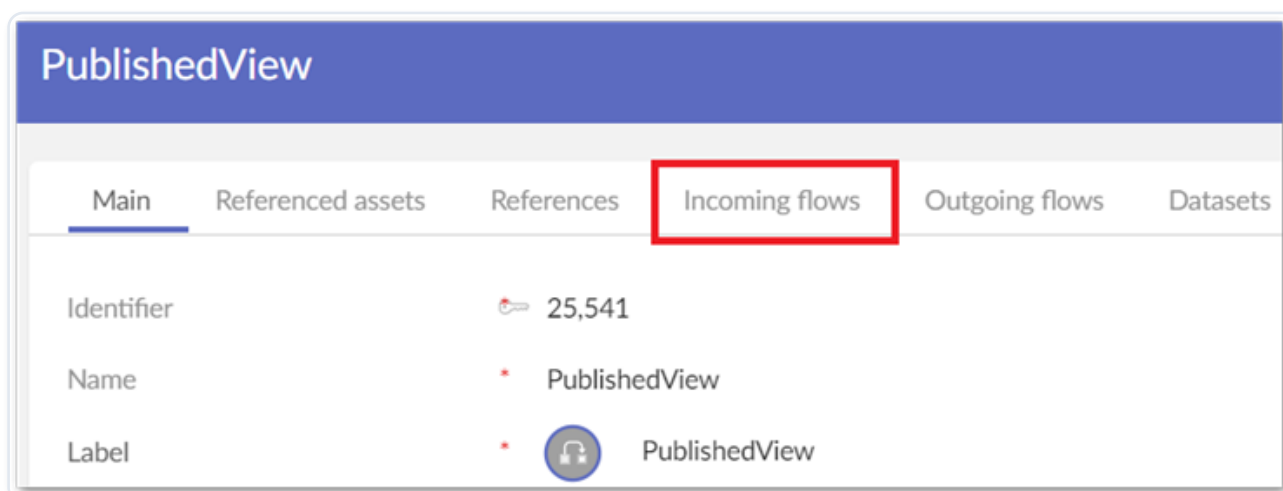
TDV 公開ビュータイプの資産から着信フローを削除すると、TDV 内の対応する公開リソースが削除されます。

メインコントロールパネルのメニューで、を選択すると、既存のすべての資産がリストされたメインページが表示されます。

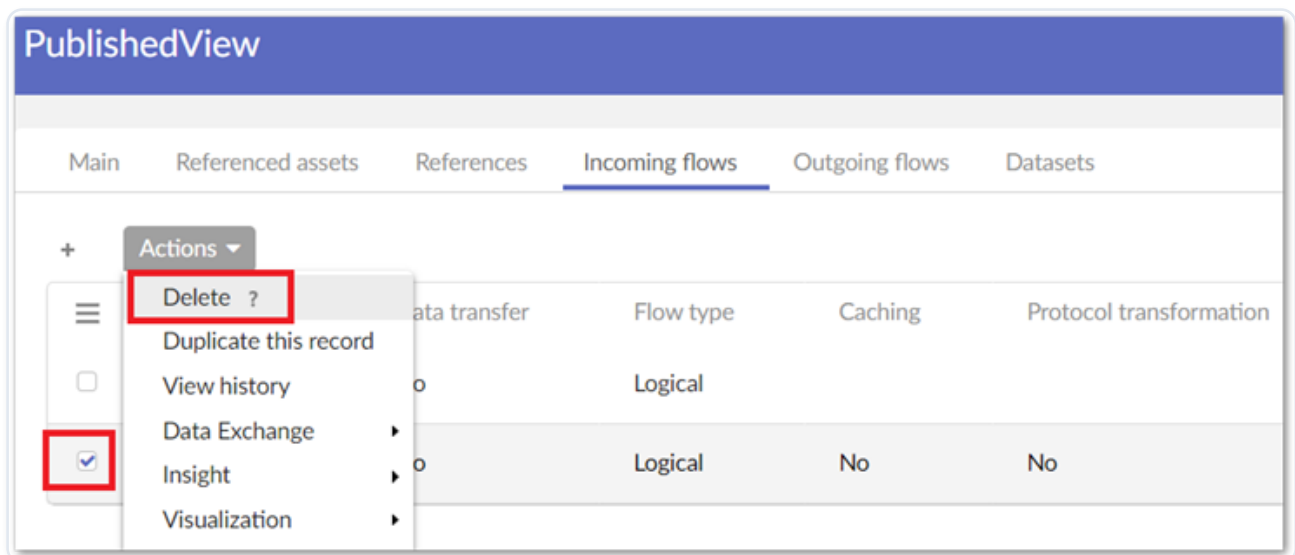
編集する TDV 公開ビュー資産を開きます。



を選択し、次に タブを選択します。



削除するフローを選択し、メニューで を選択します。



これにより、テクニカルフローが削除され、TDV 内の対応する公開エレメント (存在する場合) も削除されます。

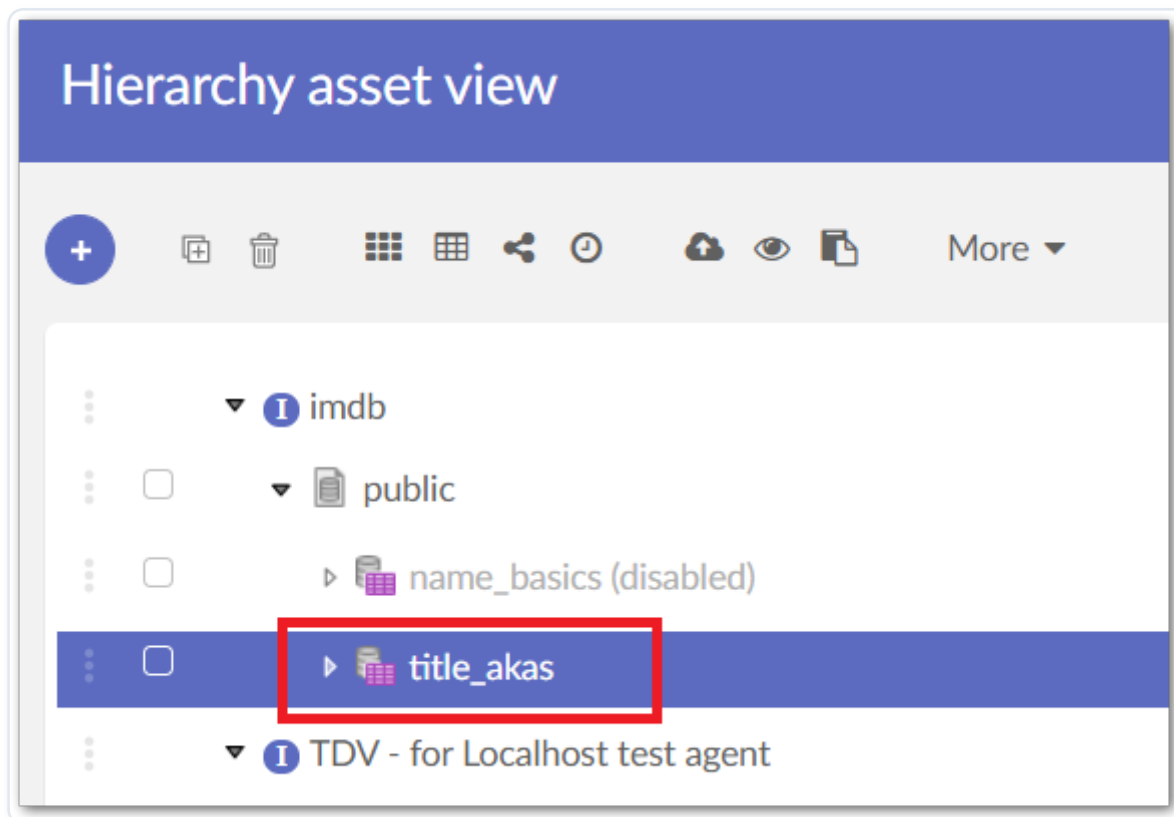


入力フローとして使用される資産の無効化

公開されたビューの着信フローとして使用されている資産を手動で無効にすると、TDV 内の対応する公開リソースが削除されます。

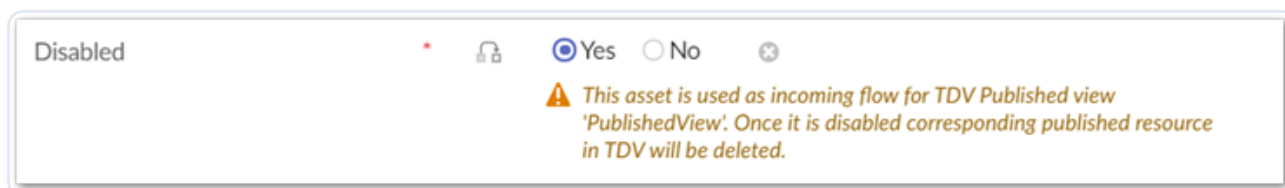
メインコントロールパネルのメニューで、を選択すると、既存のすべての資産がリストされたメインページが表示されます。

無効にする資産を選択して編集します。



フィールドを に設定します。

この資産が TDV 公開ビューの着信フローとして使用されていることを示す警告が表示されます。



これにより、TDV 内に対応する公開エレメントが存在する場合はそれが削除されます。

正規表現

- 概要
- 文字
- 文字クラス
- 定義済みの文字クラス
- POSIX 文字クラス (US-ASCII のみ)
- Unicode スクリプト、ブロック、カテゴリ、バイナリプロパティのクラス

- 境界マッチャー
- 最大量指定子
- 最小量指定子
- 絶対最大量指定子
- 論理演算子
- 後方参照
- 引用
- 特殊な構造 (名前付きキャプチャと非キャプチャ)

概要

正規表現は、資産のサンプルデータがデータエレメントに定義された正規表現と一致するかどうかに応じて、資産を分類し、データエレメントに関連付けるために使用されます。

文字

- `x` 文字 `x`
- `\\` バックスラッシュ文字
- `\0n` 8 進数値 `0n` の文字 ($0 \leq n \leq 7$)
- `\0nn` 8 進数値 `0nn` の文字 ($0 \leq n \leq 7$)
- `\0mnn` 8 進数値 `0mnn` の文字 ($0 \leq m \leq 3, 0 \leq n \leq 7$)
- `\xhh` 16 進数値 `0xhh` の文字
- `\uhhhh` 16 進数値 `0xhhhh` の文字
- `\x{h...h}` 16 進数値 `0xh...h` の文字 ($\text{Character.MIN_CODE_POINT} \leq 0xh...h \leq \text{Character.MAX_CODE_POINT}$)
- `\t` タブ文字 (`\u0009`)
- `\n` 改行文字 (`\u000A`)
- `\r` キャリッジリターン文字 (`\u000D`)
- `\f` フォームフィード文字 (`\u000C`)
- `\a` 警告 (ベル) 文字 (`'007'`)
- `\e` エスケープ文字 (`'\u001B'`)
- `\cx` `x` に対応する制御文字

文字クラス

- `[abc]` a、b、または c (単純クラス)
- `[!abc]` a、b、c 以外の任意の文字 (否定)
- `[a-zA-Z]` a から z または A から Z (範囲)
- `[a-d[m-p]]` a から d、または m から p: `[a-dm-p]` (結合)
- `[a-z&&[def]]` d、e、または f (交差)
- `[a-z&&[!bc]]` a から z、ただし b と c は除く: `[ad-z]` (減算)
- `[a-z&&[!m-p]]` a から z まで、m から p までではない: `[a-lq-z]` (減算)

定義済みの文字クラス

- `.` 任意の文字 (行末文字と一致または不一致)
- `\d` 数字: `[0-9]`
- `\D` 数字以外: `[!0-9]`
- `\s` 空白文字: `[\t\n\x0B\f\r]`
- `\S` 空白以外の文字: `[! \s]`
- `\w` 単語文字: `[a-zA-Z_0-9]`
- `\W` 単語以外の文字: `[! \w]`

POSIX 文字クラス (US-ASCII のみ)

- `\p{Lower}` 小文字のアルファベット: `[a-z]`
- `\p{Upper}` 大文字のアルファベット: `[A-Z]`
- `\p{ASCII}` すべての ASCII: `[\x00-\x7F]`
- `\p{Alpha}` アルファベット文字: `[\p{Lower}\p{Upper}]`
- `\p{Digit}` 10 進数の数字: `[0-9]`
- `\p{Alnum}` 英数文字: `[\p{Alpha}\p{Digit}]`
- `\p{Punct}` 句読点: `!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~` のいずれか
- `\p{Graph}` 表示可能文字: `[\p{Alnum}\p{Punct}]`
- `\p{Print}` 印刷可能文字: `[\p{Graph}\x20]`
- `\p{Blank}` スペースまたはタブ: `[\t]`
- `\p{Cntrl}` 制御文字: `[\x00-\x1F\x7F]`
- `\p{XDigit}` 16 進数: `[0-9a-fA-F]`

- `\p{Space}` 空白文字: `[ \t\n\r\x0B\f]`

UNICODE スクリプト、ブロック、カテゴリ、バイナリプロパティのクラス

- `\p{IsLatin}` ラテン文字 (スクリプト)
- `\p{InGreek}` ギリシャ語ブロック内文字 (ブロック)
- `\p{Lu}` 大文字 (カテゴリ)
- `\p{IsAlphabetic}` アルファベット文字 (バイナリプロパティ)
- `\p{Sc}` 通貨記号
- `\P{InGreek}` ギリシャ文字ブロックの文字を除く任意の文字 (否定)
- `[\p{L}&&[^\p{Lu}]]` 大文字以外の任意の文字 (減算)

境界マッチャー

- `^` 行頭
- `$` 行末
- `\b` 単語境界
- `\B` 非単語境界
- `\A` 入力の始まり
- `\G` 前の一致の終わり
- `\Z` 入力の終わり (最後の終端文字がある場合は除く)
- `\z` 入力の終わり

最大量指定子

- `X?` X、1 回またはまったくない
- `X*` X、0 回以上
- `X+` X、1 回以上
- `X{n}` X、正確に n 回
- `X{n,}` X、少なくとも n 回
- `X{n,m}` X、n 回以上 m 回以下

最小量指定子

- `X??` X、1 回 またはまったくない
- `X*?` X、0 回以上
- `X+?` X、1 回以上

- `X{n}?` X、正確に n 回
- `X{n,}?` X、少なくとも n 回
- `X{n,m}?` X、n 回以上 m 回以下

絶対最大量指定子

- `X?+` X、1 回 またはまったくない
- `X*+` X、0 回以上
- `X++` X、1 回以上
- `X{n}+` X、正確に n 回
- `X{n,}+` X、少なくとも n 回
- `X{n,m}+` X、n 回以上 m 回以下

論理演算子

- `XY` X の後に Y が続く
- `X|Y` X または Y のいずれか
- `(X)` X を捕捉グループとして

後方参照

- `\n` n 番目の捕捉グループに一致
- `\k<name>` 指定した捕捉グループに一致

引用

- `\` 何もしないが、次の文字を引用
- `\Q` 何もしないが `\E` までのすべての文字を引用符で囲む
- `\E` 何もしないが `\Q` で開始された引用を終了

特殊な構造 (名前付き捕捉と非捕捉)

- `(?<name>X)` X、捕捉グループとして指定
- `(?:X)` X、非捕捉グループとして指定
- `(?idsuxU-idsuxU)` 何もしないが一致フラグ `i d m s u x U` をオン/オフ
- `(?idsux-idsux:X)` X、指定されたフラグ `i d m s u x` をオン/オフを持つ非捕捉グループとして指定
- `(?=X)` X、ゼロ幅の正の前方参照
- `(?!X)` X、ゼロ幅の負の前方参照

- `(?<=X)` X、ゼロ幅の正の後方参照
- `(?<!X)` X、ゼロ幅の負の後方参照
- `(?>X)` X、独立した非捕捉グループ

ガバナンス

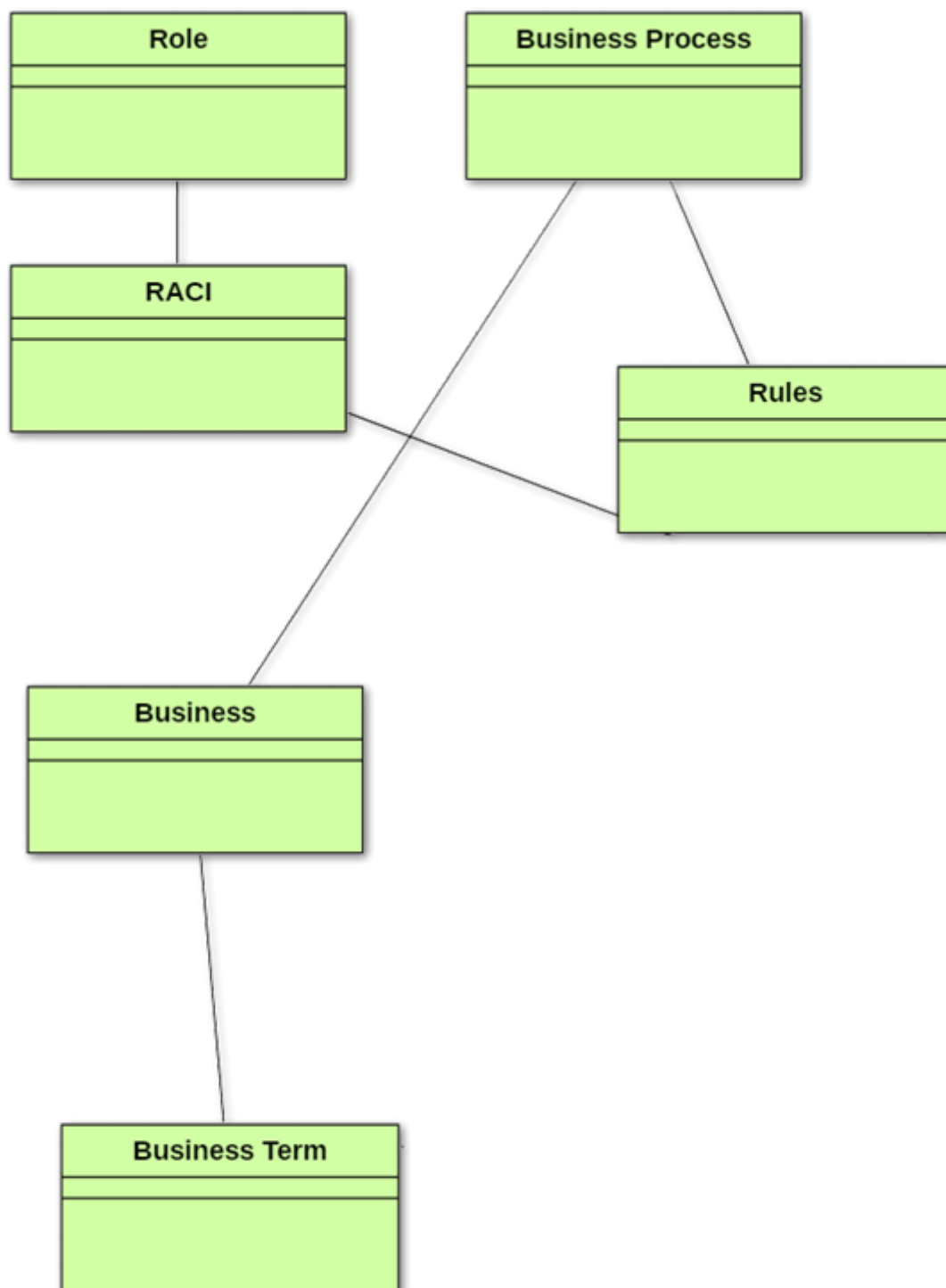
- [ガバナンスの概要](#)
- [ビジネス用語の定義](#)

ガバナンスの概要

このセクションでは、さまざまなビジネスドメインによってデータがどのように管理されるかについて説明します。

各ドメインには独自の用語と役割を設定できます。さらに、ユーザーグループとそのロール、およびデータに対する権限もここで定義されます。

データプログラムは、データに関するあらゆるプログラム、イニシアチブ、またはプロジェクトを登録することを目的としています。



ビジネス用語の定義

ビジネス用語を定義するには、のメニューを使用し、を選択して対応する画面を開きます。この画面には既存の用語がすべて表示され、リストに新しい用語を追加するための作成メニューにアクセスできます。

作成画面にアクセスするには、左上隅のアイコンをクリックします。

ビジネス用語の作成が完了したら、ボタンをクリックします。

メタデータ

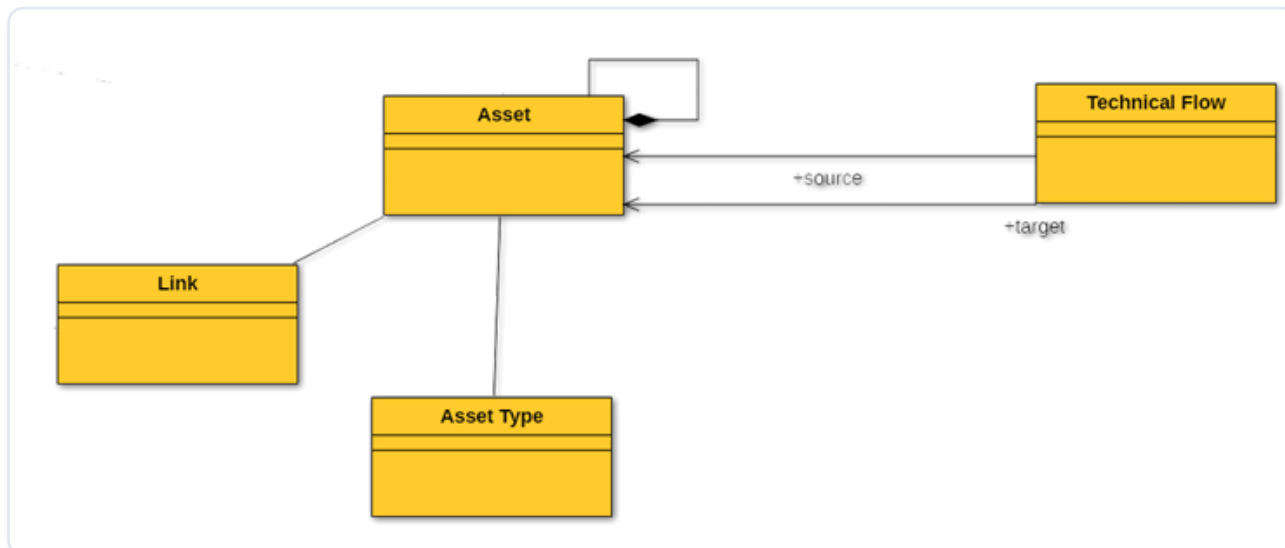
- メタデータセクションの概要
- 資産の管理

メタデータセクションの概要

カタログに記述されているデータセットの具体化については、**メタデータ** セクションに記載されています。

資産とは、データを含む、またはデータを提供する物理的なリソースです。

このセクションでは、タイプ別の技術的構造と特性について説明します。



資産の管理

- 資産の作成
- 資産の変更
- 複合データ型
- 資産の種類
- 新しい資産タイプの作成
- 重複した資産タイプ

資産の作成

パースペクティブの の下で、 を選択して資産を表示します。

左上隅にある ボタンをクリックして、資産作成画面にアクセスします。

資産の変更

現在の資産を変更する場合は、左上隅にある ボタンをクリックします。

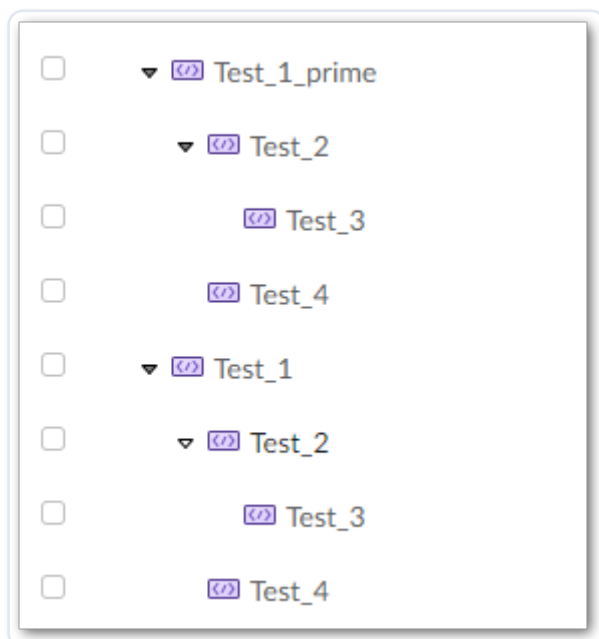
既存のデータが事前に入力された資産フォームが表示されます。必要に応じて変更することができます。

フォームビューでは、埋め込みテーブルが表示される場合があります。各埋め込みテーブルは関連付け (現在の資産を他のエレメントにリンクする方法) を表します。この場合、メニューを開いて、次の操作を実行できます。

- 1 つまたは複数のレコードを選択してデタッチします
- 1 つまたは複数のレコードを選択して削除します
- 新しいターゲットレコードを作成します
- 既存のレコードに関連付けます

複合データ型

一部の資産タイプは、データを含むものとして定義されています。これらの資産には、データタイプ属性も含めることができます。この属性が保持できる値は、データタイプとして列挙され、リストで選択できます。新しいオカレンスを作成することもできます。データタイプはラベルと説明で構成され、主に情報を提供します。ただし、フラグを設定してタイプを複雑として定義できます。データが複雑になると、このデータ型にリンクされているすべての資産は同じ構造を共有します。つまり、この資産から始まるサブツリーは、同じタイプのすべての資産と同期されます。



ここで、Test\1 と Test\1_prime は同じ複合型を持ちます。それらの構造は、Test\1 の作成後に発生した Test\1_prime の作成から同期されます。Test\1 の子の属性に対する変更は、

Test_1_prime の下位の対応する属性に影響します。削除または作成も、同じデータタイプの資産によってミラーリングされます。

資産の種類

名称	説明	資産タイプの親	含まれる特定のフィールド
DB テーブル	データベース内のテーブル形式で保持される関連データのコレクション。列と行で構成されます。 DB スキーマ	- Encoding- Partitioned DB table- TableSpace- Encryption- Sample file	N/A
DB 列	特定の単純なタイプのデータ値のセット。テキスト値、数値、またはオペレーティングシステム内のファイルへのポインターが含まれる場合があります。	DB テーブル	- Field type- Precision- Encryption- Column index- Sample file
DB スキーマ	データベース管理システムによってサポートされる形式言語で記述されたデータベース構造。	- Tablespace	N/A
DB ビュー	クエリによって定義されるデータベース内の検索可能なオブジェクト。	DB スキーマ	- Tablespace
DB 関数	合計、平均、カウントなどの基本的な操作を実行し、さらに条件引数を使用して、データベース内のレコードの指定されたサブセットに対してのみ計算を実行できます。	DB スキーマ	- Tablespace

名称	説明	資産タイプの親	含まれる特定のフィールド
DB 同義語	テーブル、ビュー、シーケンス、またはその他のスキーマオブジェクトのエイリアスまたは代替名を表します。	DB スキーマ	- Tablespace
DB シーケンス	ユーザーが一意的な整数値を生成できるようにするデータベースオブジェクト。	DB スキーマ	- Tablespace
REST サービス	REST Web サービスを使用すると、クエリシステムは、一連の統一されたステートレスな定義済み操作を通じて、テキスト表現を介して Web リソースを操作できます。	- Version- URL (エンドポイントまたは URI)	N/A
REST リソース	型、関連データ、他のリソースとの関係、およびそのオブジェクトを操作する一連のメソッドを持つオブジェクト。	REST サービス	- Field type- URL (エンドポイントまたは URI)
SOAP サービス	構造化された情報を交換するためのシンプルなオブジェクトアクセスプロトコルサービス。	- Version- URL (エンドポイントまたは URI)	N/A
SOAP 操作	SOAP サービス操作 (getRecord、count、Create など)	SOAP サービス	- URL (エンドポイントまたは URI)

名称	説明	資産タイプの親	含まれる特定のフィールド
SOAP リクエスト	SOAP 操作へのアクセスリクエスト。	SOAP 操作	N/A
SOAP 応答	特定の SOAP 操作への SOAP リクエストリンクから返された応答。	SOAP 操作	N/A
Excel ブック	ワークシートとも呼ばれる 1 つ以上のスプレッドシートのコレクション。	- Version- Encoding- Locale- Attribute value- Encryption- File name- File size- File extension- Header present- Protected- Sample file	N/A
Excel ワークシート	Excel で作成されたファイル内の 1 ページ。	Excel ブック	- Protected- Order- Sample file
Excel ワークシート配列	ワークシートの配列。	Excel ワークシート	- Number of lines- Start line- Protected- Sample file
CSV ファイル	データを表形式で保存できるカンマ区切り値ファイル。	- Version- Encoding- Encryption- File name- File size- File extension- Line separator- String delimiter- Header present- Start line- Protected- Sample file	N/A
列	CSV ファイルまたは Excel ワークシート内の列を表します。	- CSV file - Excel Worksheet	- Format- Field type- Default value- String delimiter- Column index- Row index- Formula-

名称	説明	資産タイプの親	含まれる特定のフィールド
			Orientation- Sample file
XML ファイル	拡張マークアップ言語データファイル。HTML ドキュメントとほぼ同じ形式ですが、カスタムタグを使用してオブジェクトと各オブジェクト内のデータを定義します。	- Version- Encoding- XML Encoding- Encryption- File name- File size- File extension- Protected- Sample file	N/A
XML ストリーム	XML コンテンツのストリーム。	- Encoding- XML Encoding	N/A
XML エlement	Elementの開始タグからElementの終了タグまでのすべてです。	- XML file - XML Stream	- Field type- Node type- Default value- TagName (attribute name)- Pattern- Range- Enumeration- Maximum length- Minimum length- Sample file
XML 属性	特定のElementに関連するデータが含まれます。	XML Element	- Field type- Default value- Sample file
JSON ファイル	標準データ交換形式である JavaScript Object Notation (JSON) 形式で単純なデータ構造とオブジェクトを保存するファイル。	- Version- Encoding- Encryption- File name- File size- File extension- Protected- Sample file	N/A
JSON Element	JSON のデータ構造。データは、JSON オブジェクト	- JSON file- JSON stream	- Format- Field type- Default value- Array length- Media

名称	説明	資産タイプの親	含まれる特定のフィールド
	ト、テキスト、数値、ブール値などになります。		encoding- Content media type- Pattern- Range- Enumeration- Maximum length- Minimum length- Sample file
JSON プロパティ	各 JSON エlement は、JSON プロパティによって識別されます。	JSON エlement	- Field type- Default value- Sample file
JSON ストリーム	JSON コンテンツのストリーム。	- Encoding	N/A
TDV 公開ビュー	特定の TIBCO® Data Virtualization 公開ビュー。	- Encoding- Sample file	N/A
未定義	不明な形式のデフォルトの資産タイプ。	N/A	N/A

新しい資産タイプの作成

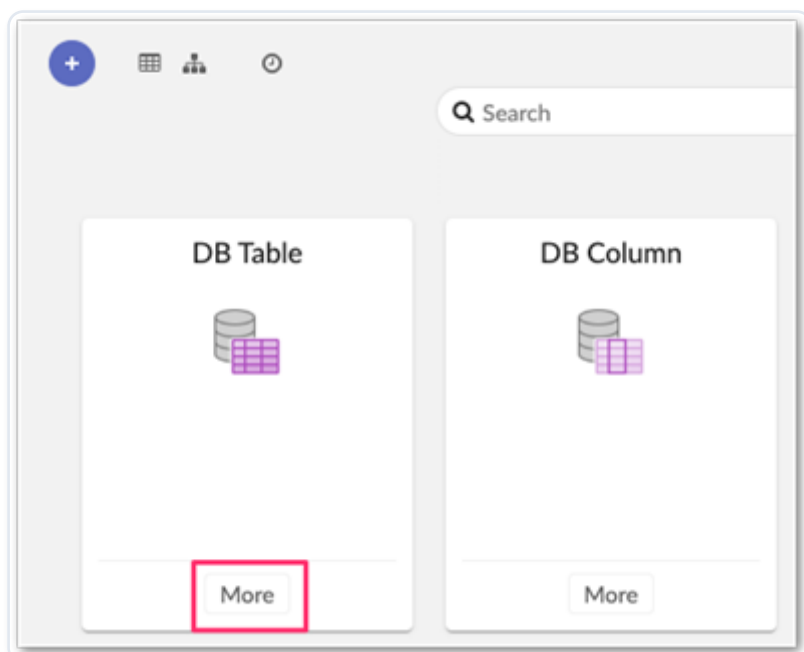
資産タイプのメインビューの左上隅にある ボタンをクリックすると、新しい資産タイプを作成できます。

入力するフィールドがあるフォームが表示されます。

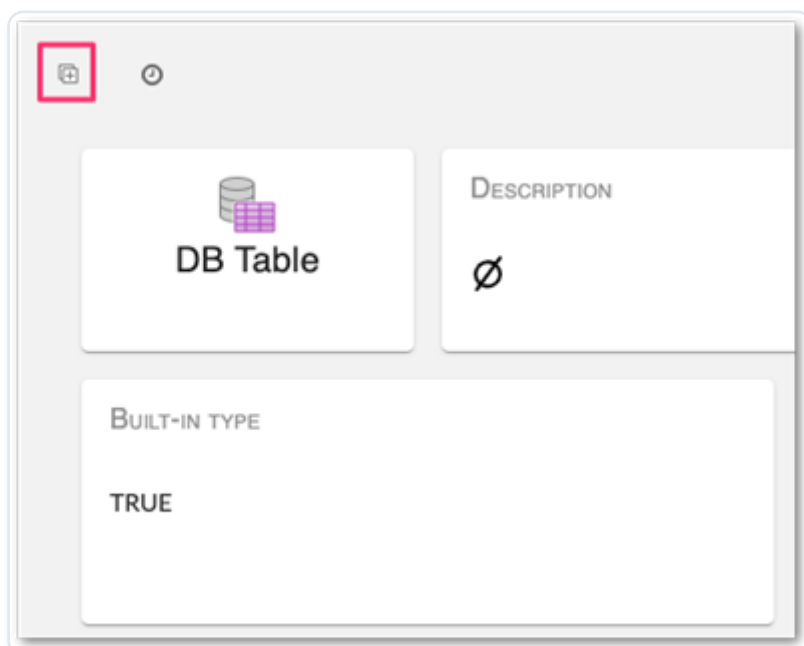
重複した資産タイプ

既存の資産タイプから独自の資産タイプを作成する場合は、複製アクションボタンをクリックできます。

階層ビューの場合は、最初に複製する資産タイプをクリックするか、カードビューモードの場合は ボタンをクリックして、複製する資産タイプを表示する必要があります。



資産タイプ表示フォームで、左上隅にある ボタンをクリックします。



更新を行うことができる、事前に入力されたフォームが表示されます。

テクノロジー

- [テクノロジーの概要](#)
- [インスタンスの作成](#)
- [システムの作成](#)
- [システムタイプの作成](#)
- [インフラストラクチャーの作成](#)
- [プロバイダーの作成](#)

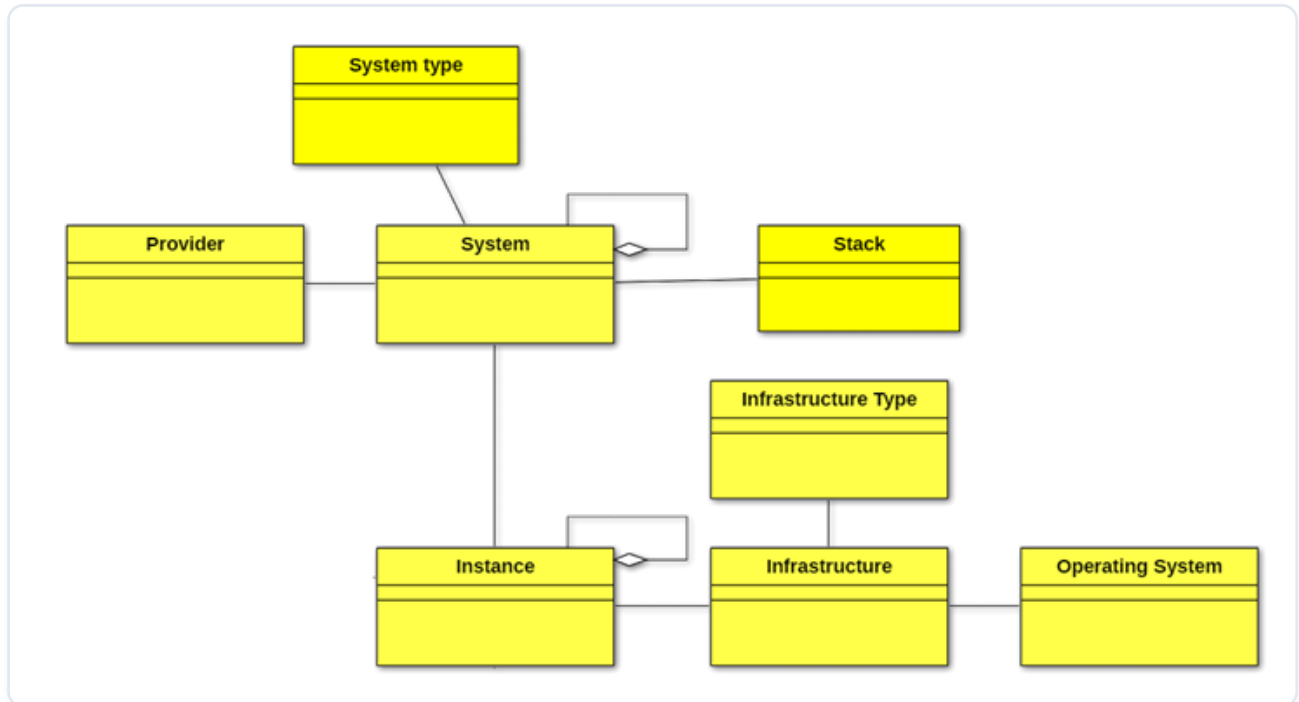
テクノロジーの概要

データ管理は、多くのシステムとして表されるインスタンスに依存します。

これは、データを公開する特定のインフラストラクチャー上で実行されているシステムのインスタンスです。

システムは、データ管理およびガバナンスのプロセスに関与するあらゆるシステムでもあります。

これらは、ベンダーやその他の外部または内部組織であるプロバイダーによって提供されます。



インスタンスの作成

メインパースペクティブのグループで、を選択してインスタンス画面にアクセスします。既存のインスタンスがすべてリストされたメインページが表示されます。

このページには、次のように、インスタンスを作成するための複数の選択肢があります。

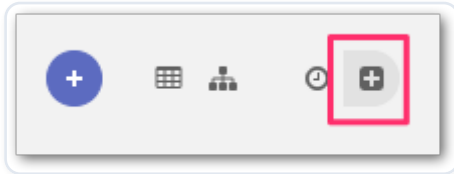
- 左上隅のアイコンをクリックして作成画面にアクセスし、インスタンスを最初から作成します。
- エージェントから作成画面にアクセスするには、左上隅にある四角いアイコンをクリックします。

最初から作成されたインスタンスの場合、TDV に対応するデータソースを作成し、メタデータを自動的に収集することができます。詳細については、[インスタンスからのデータソースの作成](#)を参照してください。

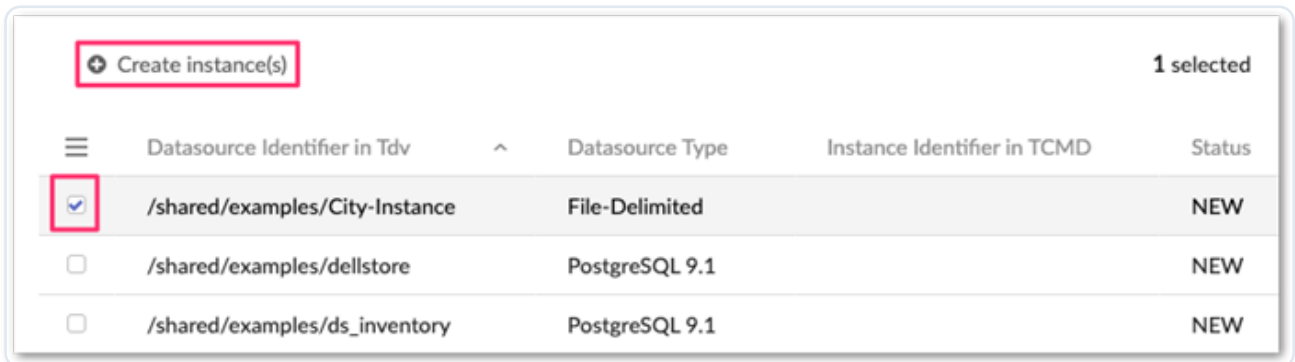
エージェントからのインスタンスの作成

エージェントからインスタンスを作成するには、TIBCO® Data Virtualization アプリケーションに接続された使用可能なエージェントが必要です。

インスタンスのメインページで、上部のアイコンをクリックします。



追加可能なすべてのデータソースを一覧表示するメインページが表示されます。このページでは、チェックボックスをクリックし、左上隅の を選択して、追加する 1 つ以上のデータソースを選択できます。



作成の最後に成功メッセージが表示され、インスタンスがインスタンスリストに自動的に追加されます。

データソース選択リストに表示される属性の説明

- TDV のデータソース識別子：アプリケーション内のデータソースの TDV 識別子
- データソースの種類：Oracle、PostgreSQL などのデータソースの種類
- EBX® メタデータ管理アプリケーションのインスタンスの識別子：EBX® メタデータ管理アプリケーションのインスタンスのテーブルで選択されたインスタンスの識別子

EBX® メタデータ管理アプリケーションに対応するインスタンスが存在しない場合、識別子は空になります。

- ステータス：EBX® メタデータ管理アプリケーション内の TDV データソースのステータス。NEW の場合、データソースはアプリケーション内に存在しません。STORED の場合、データソースはすでに存在します。
- 作成日：データソースの作成日
- 更新日：データソースの最終更新日

システムの作成

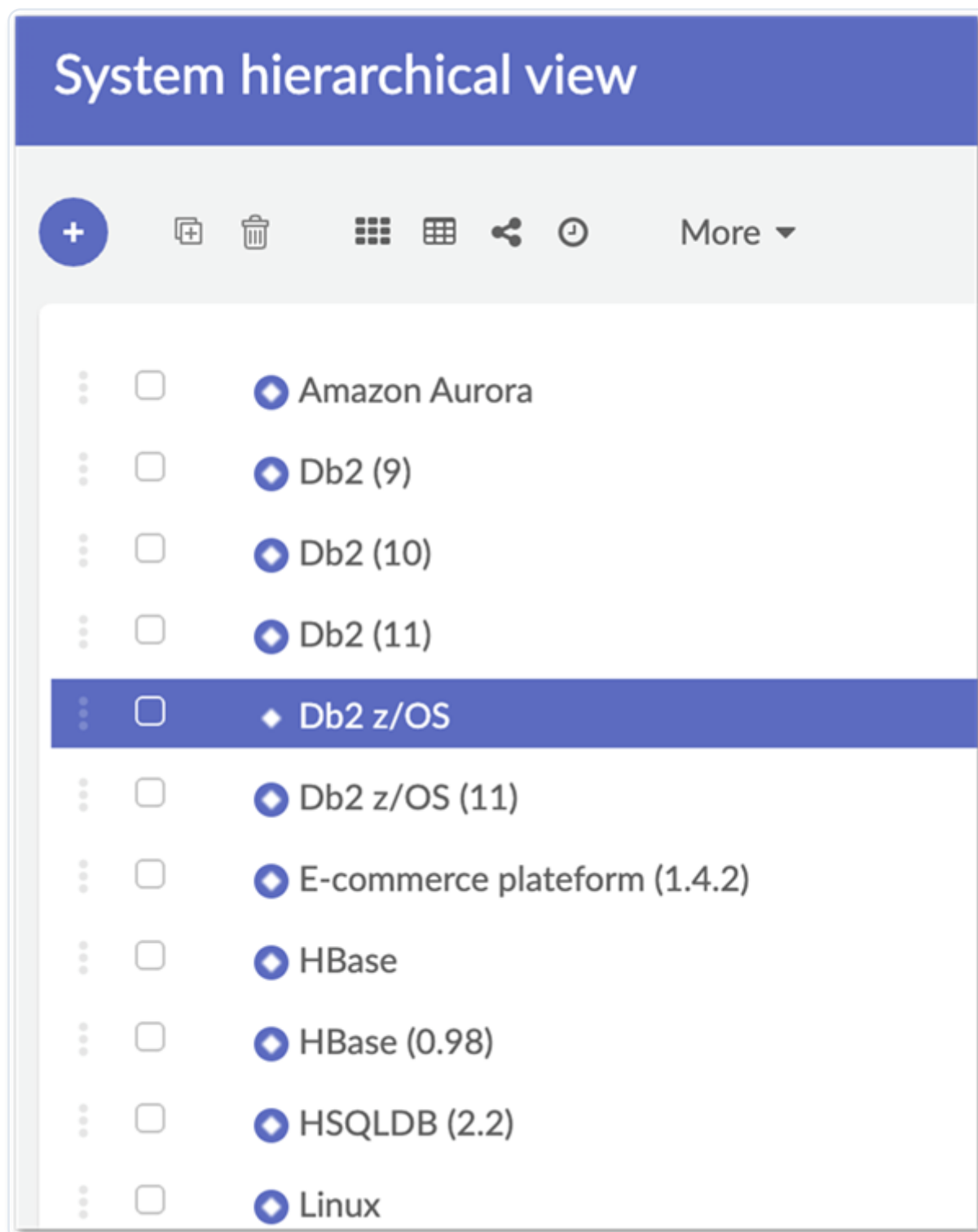
- 概要
- 新しいシステムの作成
- 新しいスタックの作成

概要

メインパースペクティブのグループで、を選択してシステム画面にアクセスします。既存のすべてのシステムをリストしたメインページが表示されます。

階層ビューをクリックすると、収集機能とプロビジョニング機能に必要ないくつかのビルトインシステムが表示されます。

- TIBCO Data Virtualization
- IBM DB2 (複数のバージョン)
- HBase (複数のバージョン)
- HSQLDB
- Microsoft Excel
- Microsoft SQL Server (複数のバージョン)
- Oracle MySQL (複数のバージョン)
- Oracle データベース (複数のバージョン)
- PostgreSQL (複数のバージョン)



これらのビルトインシステムは、収集およびプロビジョニング機能に必要なため削除できません。必要に応じて、他のシステムやシステム タイプを作成できます。

新しいシステムの作成

左上隅の アイコンをクリックすると、作成画面にアクセスできます。この画面で、システム情報を入力する必要があります。

新しいスタックの作成

スタックとは、さまざまなシステムを構造的に関連付け、それらの組み合わせを制限することで、情報システムと技術アーキテクチャのガバナンスを合理化する方法です。

EBX® メタデータ管理アプリケーションでは、スタックは別のシステムを参照するシステムとして定義されます。

新しいスタックを定義するには、System type = **Stack** を使用して新しいシステムを作成し、その他のシステム属性を入力します。

スタックコンポーネントを定義するには 2 つの方法があります。

- スタック自体の タブで、このスタックのコンポーネント (システム) を関連付けます (たとえば、Web アプリケーションスタックを定義するには、データベース + アプリケーションサーバーと Web サーバー)
- 各コンポーネントの タブで、このコンポーネントを目的のスタックに関連付けます。

システムタイプの作成

システムタイプには複数の用途があります。例えば、

- システムをファミリーに分類
- スタックを定義
- インフラストラクチャーのオペレーティングシステムを定義
- プロビジョニングを可能にするために TDV インフラストラクチャーを入力

メインコントロールパネルの メニューで、 を選択してシステム画面にアクセスします。システムを作成するときは、そのタイプを定義する必要があります。

いくつかのビルトインシステムタイプが定義されています。

- オペレーティング システム：これは、インフラストラクチャー定義ページで利用可能なオペレーティングシステムにフィルターを設定するために使用されます。
- データベースシステム
- ミドルウェア
- アプリケーションサーバー
- Web サーバー
- エンタープライズサービスバス
- インフラストラクチャーサービス
- セキュリティサービス

- ストレージシステム
- メッセージングシステム
- イベント管理システム
- キャッシュシステム
- ファイルシステム
- エージェント
- データウェアハウス
- データマート
- データレイク
- ETL
- Hadoop ベースのプラットフォーム
- レジストリ/ディレクトリ
- ゲートウェイ
- スタック： ./manage_systems.html) を定義するために使用されます。
- データ仮想化 (TIBCO® Data Virtualization など)
- クラウドプラットフォーム
- スプレッドシート
- マスターデータ管理
- ERP
- ビジネスインテリジェンス
- 未定義

新しいシステムタイプの作成

ビルトイン型が使用目的に十分でない場合は、他のシステム型を作成できます。

システム作成/更新ページのシステムタイプ選択で、新しいシステムタイプを作成できます。

Type *	[not defined]
Provider	Agent
Edition	Application Server
Version	Business Intelligence
Commercial version	Caching System
License	Cloud Platform
Usage policy	Data Virtualization
Parent system	Database System
Contact	Datalake
Documentation	Datamart
Built-in system	Datawarehouse
	Enterprise Service Bus
	ERP
	ETL
	Event Management System
	File System
	Gateway 
	+ Create Selector 

インフラストラクチャーの作成

- [インフラストラクチャータイプの作成](#)
- [新しいインフラストラクチャータイプの作成](#)

インフラストラクチャータイプの作成

インフラストラクチャータイプは、インフラストラクチャーをファミリー間で分類するために使用されます。メインパースペクティブのグループで、[インフラストラクチャータイプ](#)を選択してインフラストラクチャー画面にアクセスします。インフラストラクチャーを作成するときは、そのタイプを定義する必要があります。

いくつかのビルトインインフラストラクチャータイプが定義されています。

- SaaS アプリケーション

- Paas インスタンス
- IaaS インスタンス
- 物理サーバー
- 仮想サーバー
- コンテナ化：たとえば Docker テクノロジーの使用
- Serverless
- アプライアンス：たとえば Oracle Exadata
- メインフレーム
- IOT デバイス
- モバイルデバイス
- 仮想デスクトップ
- パソコン
- 未定義

新しいインフラストラクチャータイプの作成

ビルトインタイプが使用目的に十分でない場合は、他のインフラストラクチャータイプを作成できます。

ページのインフラストラクチャータイプの選択で、新しいインフラストラクチャータイプを作成できます。

Identifier	8
Label	Infrastructure
Description	
Type	[not defined] Appliance Containerized IaaS Instance IOT Device Mainframe Mobile Device PaaS Instance Personal Computer Physical Server SaaS Application Serverless Undefined Virtual Desktop Virtual Server + Create Selector
CPU number	
CPU type	
RAM number (GB)	
Drive type	
Drive capacity (GB)	
Operating system	
Cluster	
Business Continuity Plan	
Owner	
Documentation	

プロバイダーの作成

パースペクティブのメニューで、を選択してプロバイダー画面にアクセスします。既存のすべてのプロバイダーをリストするメインページが表示されます。

プライバシー

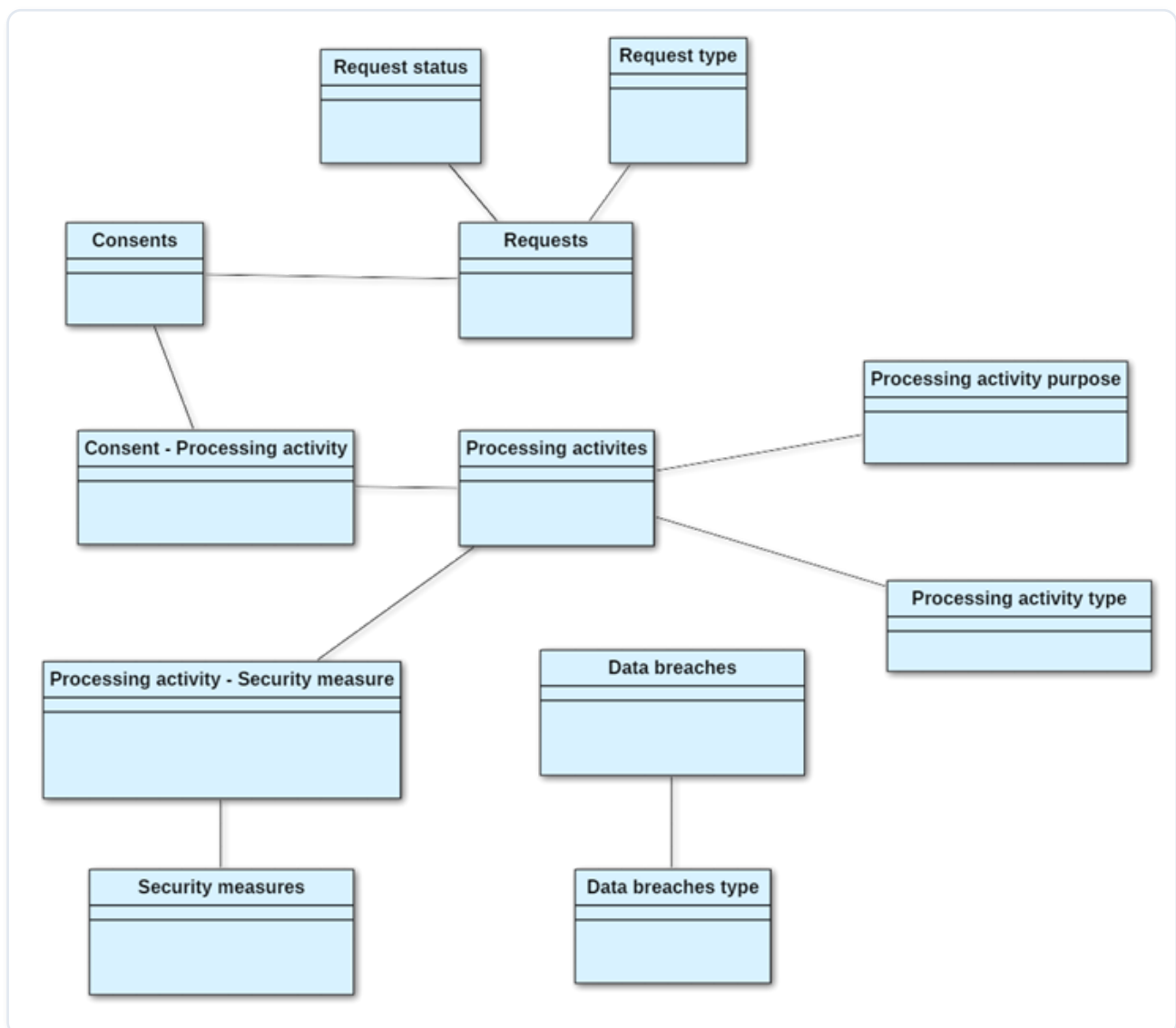
- プライバシーの概要

- 処理アクティビティ
- 同意
- データ侵害
- リクエスト

プライバシーの概要

多くの規制当局によって義務付けられているように、データプライバシーの管理とは、処理活動と個人の同意の一覧を保持することを意味します。また、個人データ所有者からのリクエストを登録し、データ侵害を伝えることも求められます。

すべての条件は、それらを含む物理的な資産、それら进行处理するシステム、およびそれらのネイチャーを定義する概念に接続され、それらの使用が暗示するルールへの準拠を保証するために統合されます。



処理アクティビティ

パースペクティブの ****にある テーブルには、企業のすべての処理アクティビティが登録されます。規制遵守のために、企業がこのアクティビティのレジストリを提供できることが重要です。これは、個人情報に関連するアクティビティに特に当てはまります。

各処理アクティビティでは、目的、責任、影響評価、関連データカテゴリ、個人データカテゴリなどの情報が収集されます。

同意

の にある テーブルを使用すると、特定の処理アクティビティと個人データに対する個別の同意を追跡できます。

データ侵害

の にある テーブルは、検出されたデータ侵害の報告を処理します。侵害は特定のインスタンスに関係します。ほとんどの規制では、侵害のステータス、発見日、提出日、影響を受ける顧客などを含む通知を監督当局に提出することが義務付けられています。

リクエスト

ほとんどのデータプライバシー規制では、個人に対して、企業が管理および処理している個人のデータを開示するよう企業に要求する権利が与えられています。たとえば、GDPR テキストの第 III 章には、個人のアクセス、訂正、移植性、消去、処理制限、オブジェクト権が記載されています。パースペクティブの にある テーブルを使用すると、このようなリクエストとその進行状況を追跡できます。

ワークスペース

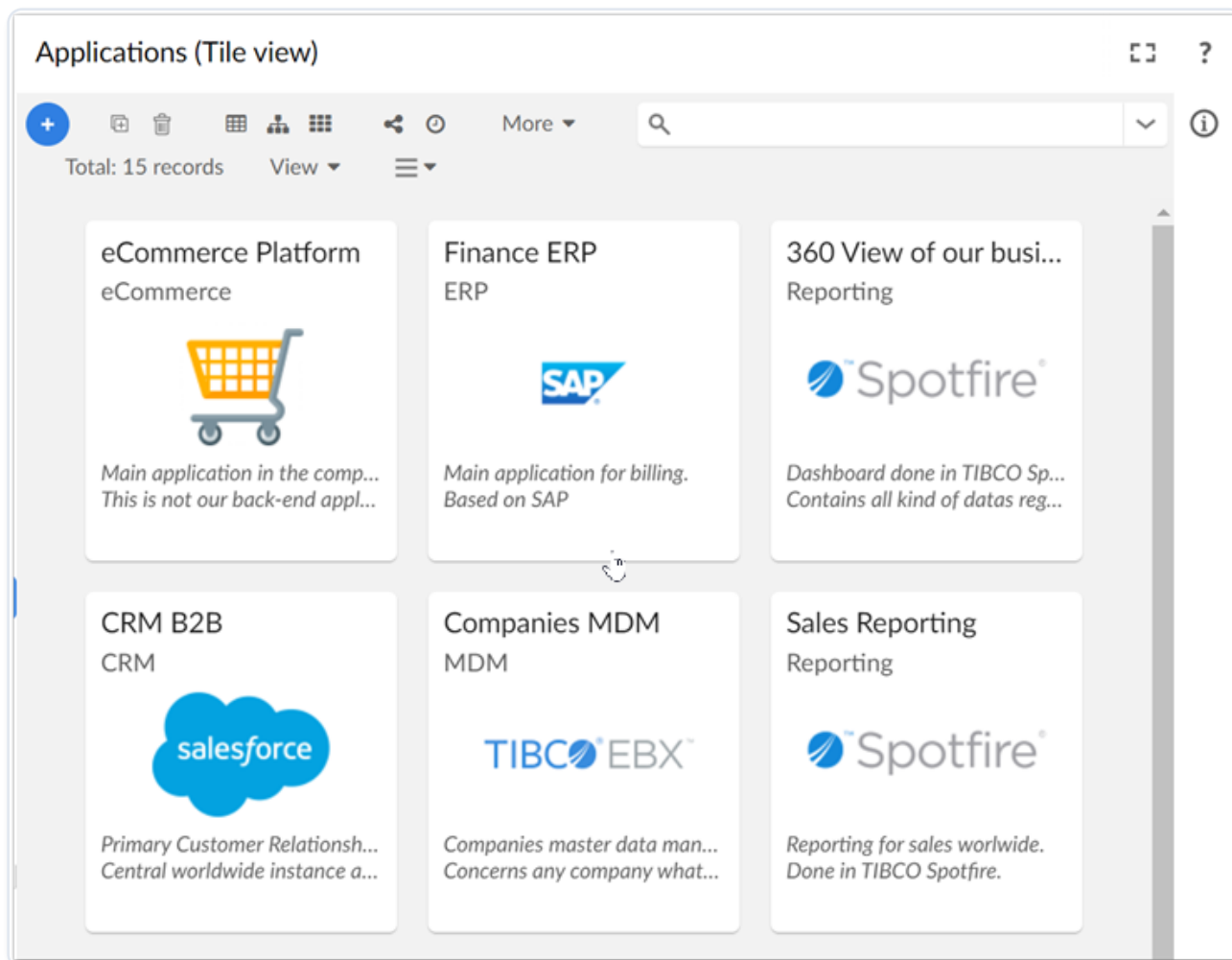
- [概要](#)
- [データビュー](#)

概要

ワークスペースにはメタデータが表示され、関連するアクションを実行できます。セクションを除くすべてのメニュー項目には、エンティティのリストにアクセスするためのデフォルトのビューが表示されます。グリッドには、アプリケーションやビジネス用語など、同じ種類のエンティティを表すカードが含まれています。階層ビューは、資産の既定のビューを提供するために使用されます。

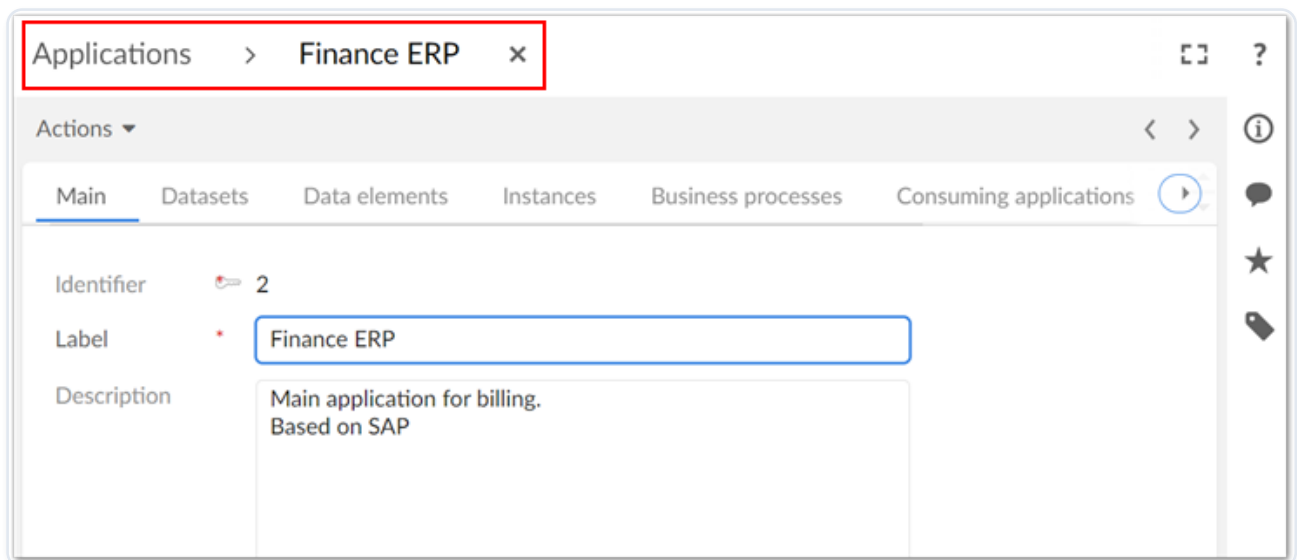
ワークスペースの左上隅にツールバーが表示されます。承認されたユーザーは、アイコンをクリックしてレコードを作成できます。作成された項目がワークフローの対象となる場合、このボタンをクリックすると作成ワークフローが起動します。このツールバーでは、デフォルトの表形

式ビューまたはデフォルトの階層ビューに切り替えることもできます。現在のエンティティの変更履歴にアクセスするには、時計アイコンを選択します。



データビュー

最初のビューから、他のビューやサービスにアクセスして、メタデータを表示したり、アクションを実行したりできます。ビュー内を参照するときに、ビルトインのパンくずリストを使用して前のページに移動できます。ブラウザのナビゲーションボタンは完全にはサポートされていないため、このナビゲーション方法が推奨されます。



パースペクティブでは、すべての標準 EBX® データビューにアクセスできます。

- 表形式ビュー
- 階層ビュー
- タイルビュー
- グラフィカルビュー
- 詳細ビュー
- 作成したカスタムビュー

ホームページ

- 概要
- ホームページウィジェット
 - ウィジェットの種類
 - ウィジェット「ライブラリ」と「マイウィジェット」
 - ウィジェットの作成/複製/更新/削除
- ホームページのカスタマイズ
 - ウィジェットの位置変更
 - 構成済みウィジェットの追加/削除
 - 自分で作成したウィジェットの追加/更新/削除

概要

ホームページには、サービスやデータにすばやくアクセスできるウィジェットを使用して、グローバルメタデータビューが表示されます。

以下に示すように、ホームページに初めてアクセスしたときに、次のいずれかを選択できます。

- デフォルトホームページの使用：このオプションを選択すると、デフォルトのホームページが表示されます。これは後でカスタマイズできます。詳細については、[ホームページのカスタマイズ](#)を参照してください。
- ホームページの作成：を選択すると、何も入力されていないホームページと、左上隅にボタンが表示されます。このボタンを使用して、利用可能なウィジェットを選択し、ホームページに追加します。ボタンを使用して変更を保存します。

ホームページウィジェット

ホームページにはウィジェットの情報が表示されます。ウィジェットタイプから独自のウィジェットを作成することも、構成済みウィジェットを使用することもできます。次のセクションでは、使用可能なウィジェットのタイプについて説明します。

ウィジェットの種類

複数のウィジェットタイプが利用可能です。たとえば、グローバル検索にすばやくアクセスするためのウィジェットや、特定のテーブルにすばやくアクセスするためのウィジェットがあります。次のセクションでは、利用可能なウィジェットについて説明します。

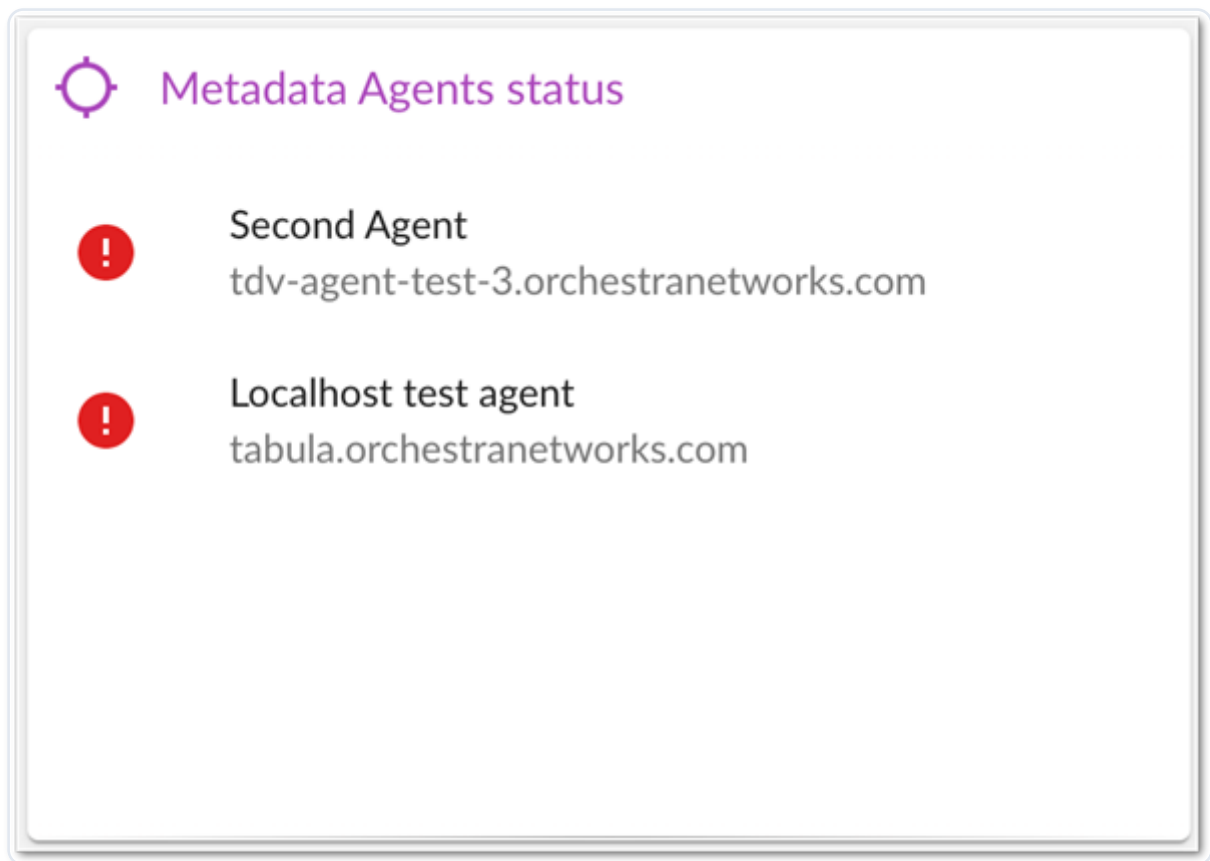
グローバル検索ウィジェット

このウィジェットには、グローバル検索サービスを使用して検索するための検索語を入力できる検索バーが表示されます。このウィジェットでは構成オプションは使用できません。

メタデータエージェントステータスウィジェット

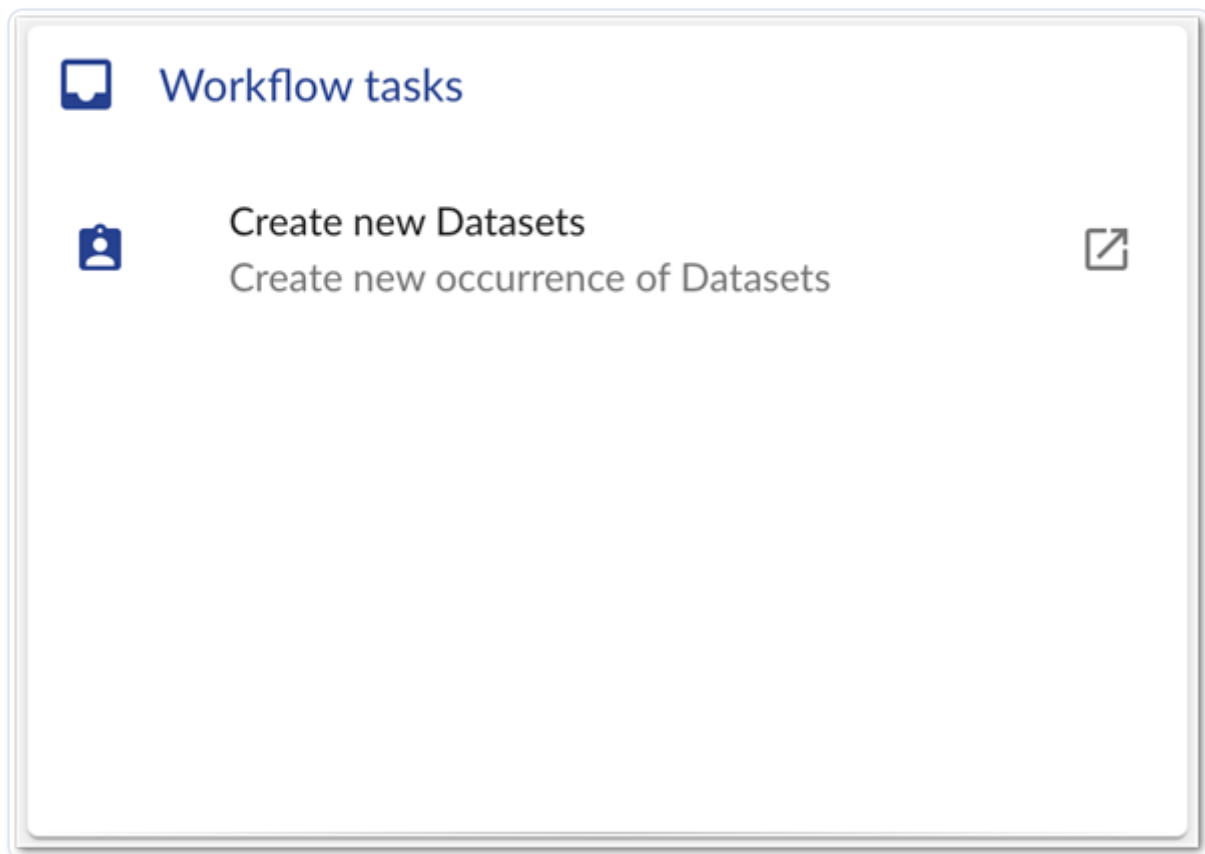
このウィジェットは、メタデータエージェントのステータスを表示します。エージェントが接続されているかどうか、また接続されていない場合はその理由を確認できます。このウィジェットには、次の構成オプションが含まれています。

- タイトル：ウィジェットのタイトル
- 説明：ウィジェットの説明
- エージェント：表示されるエージェントを 1 つまたは複数の選択されたエージェントに制限するオプションパラメーター



ワークフロータスクウィジェット

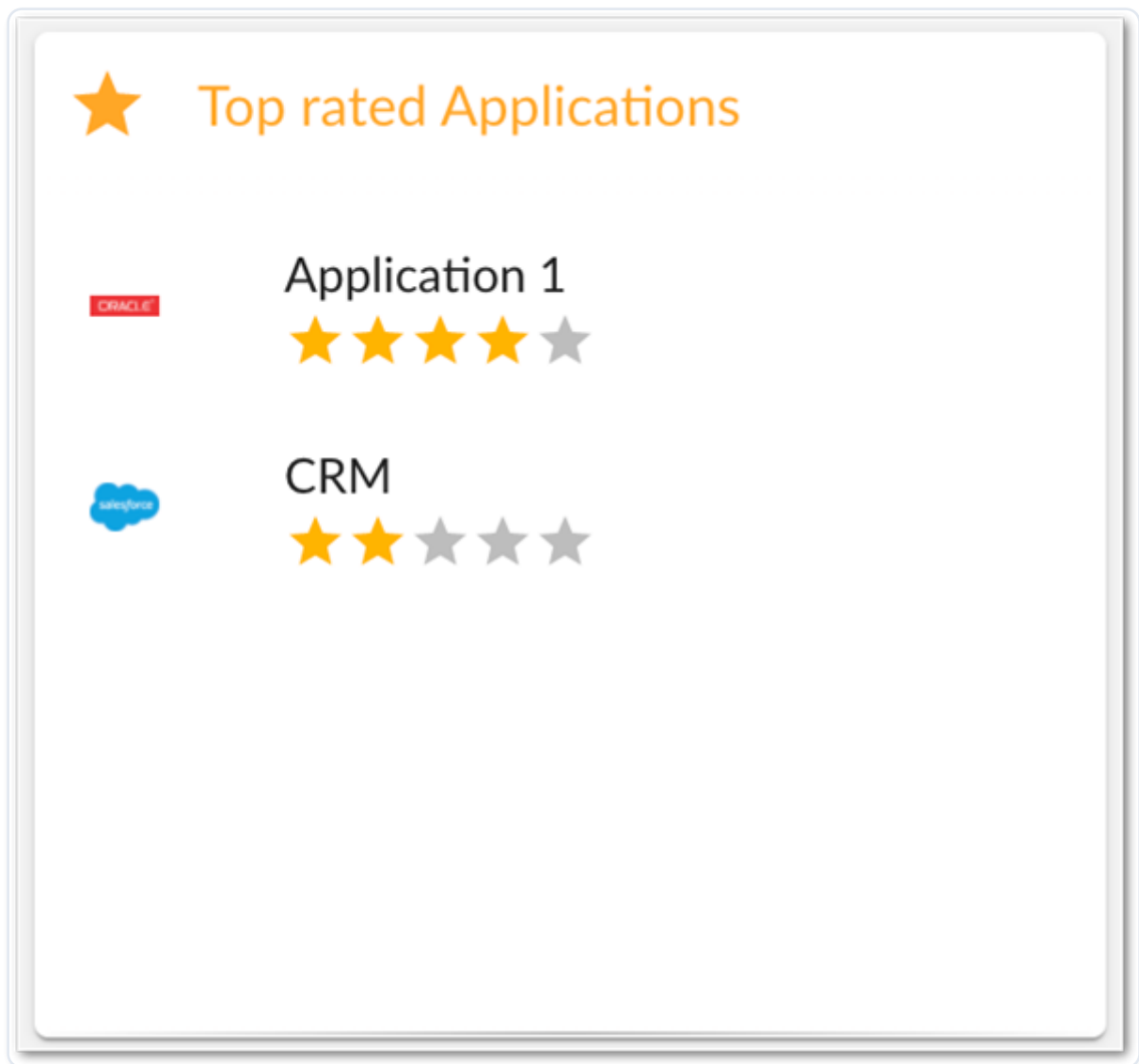
このウィジェットには、待機中または提供されているワークフロータスクが表示されます。このウィジェットからタスクをクリックすると、ワークフロービューまたはワークフロージャブビューにアクセスできます。タスクが提供されていてまだ割り当てられていない場合は、割り当てられることを通知するポップアップメッセージが表示されます。このウィジェットでは構成オプションは使用できません。



エレメント数ウィジェット

このウィジェットには、エンティティの最も評価の高いエレメントが表示されます。エレメントをクリックすると、そのエレメントにアクセスできます。評価されているエレメントがない場合、このウィジェットには空の記号が表示されます。このウィジェットには、次の構成オプションが含まれています。

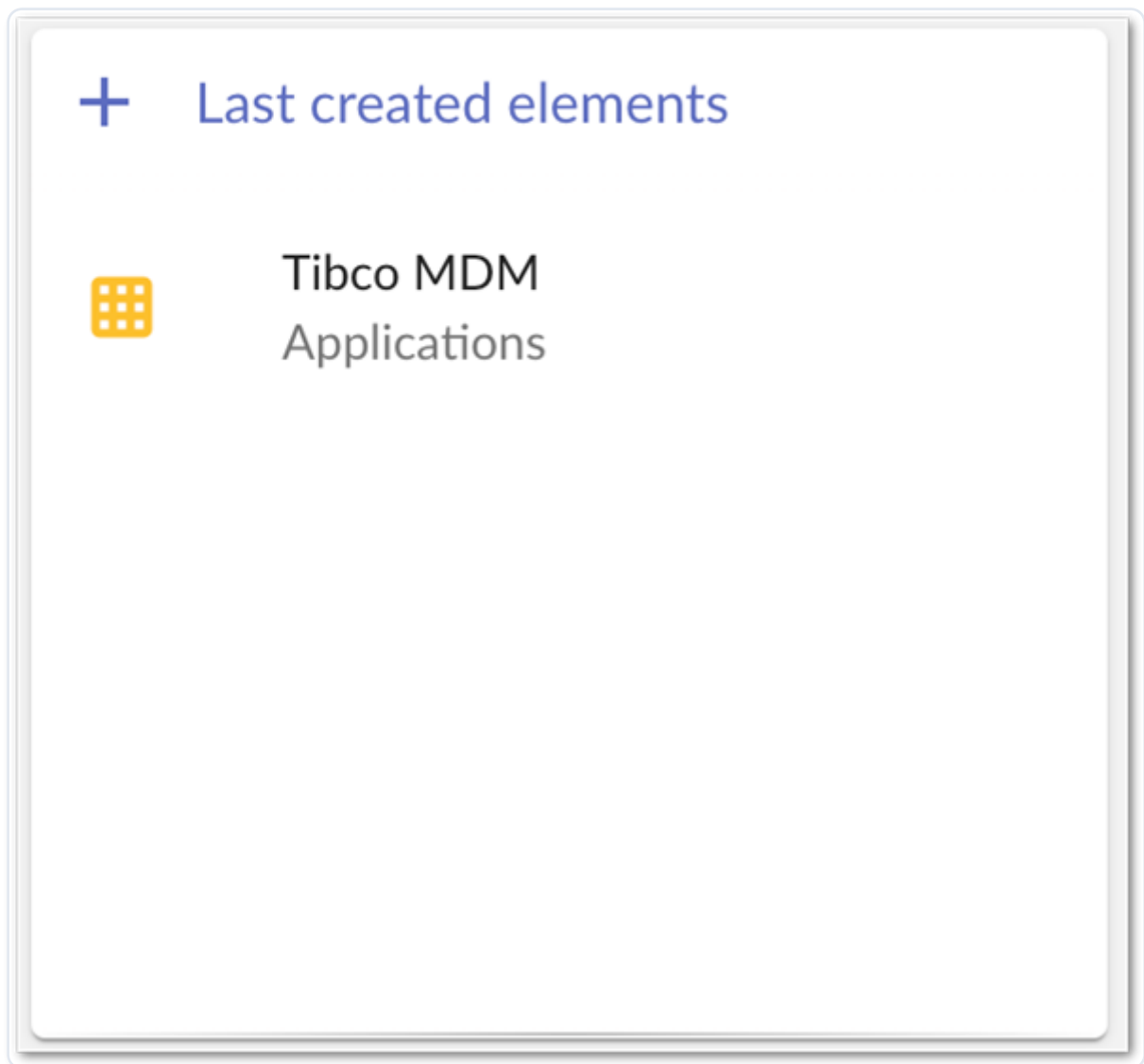
- タイトル：ウィジェットのタイトル
- 説明：ウィジェットの説明
- エンティティ：表示するエンティティを選択
- エレメント数：表示する評価の高いエレメントの数
- 述語：選択したエンティティをフィルタリングするための Xpath 述語形式



最後に作成されたエレメント

このウィジェットには、選択したエンティティから最後に作成されたエレメントが表示されます。エレメントをクリックすると、そのエレメントにアクセスできます。エレメントの上にマウスを移動すると、作成時間と作成者のユーザー名を含むツールヒントが表示されます。このウィジェットには、次の構成オプションが含まれています。

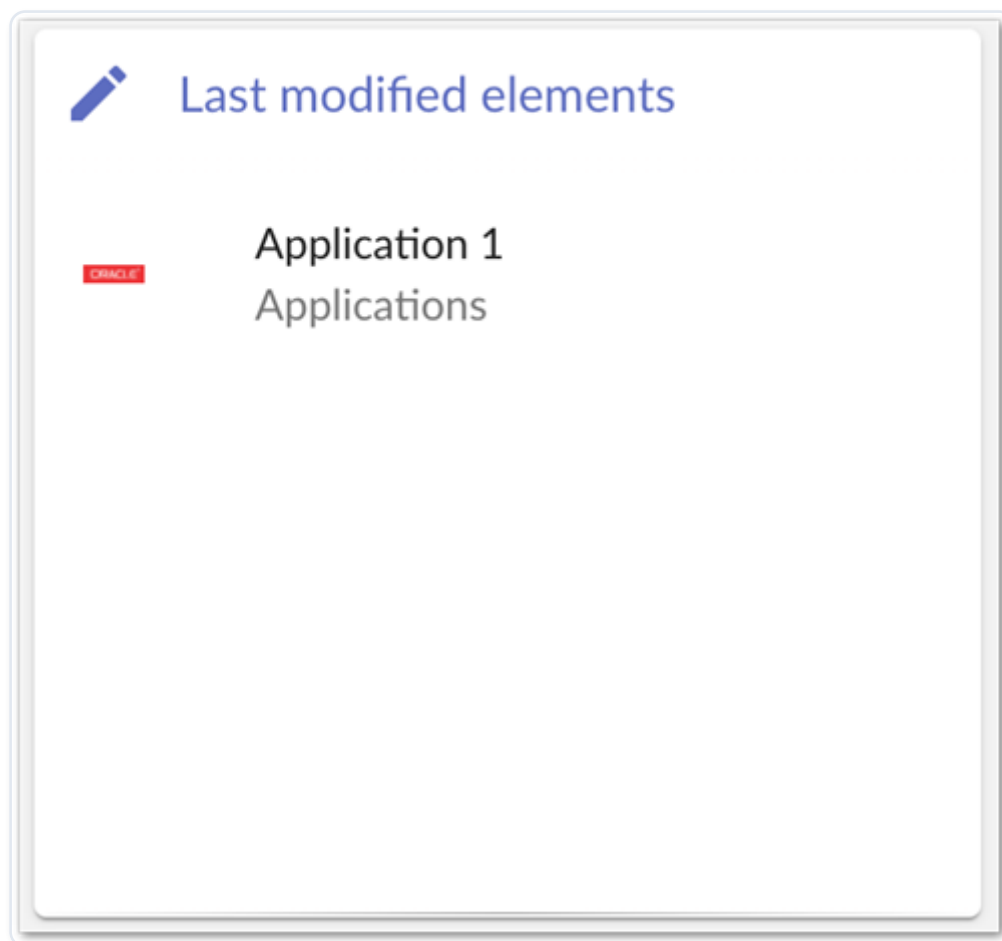
- タイトル：ウィジェットのタイトル
- 説明：ウィジェットの説明
- エンティティ：表示するエンティティを 1 つ以上選択
- エレメント数：表示するエレメントの最大数



最後に変更されたエレメント

このウィジェットには、選択したエンティティから最近変更されたエレメントが表示されます。エレメントをクリックすると、そのエレメントにアクセスできます。エレメントの上にマウスを移動すると、変更時刻と変更を行ったユーザー名を含むツールヒントが表示されます。このウィジェットには、次の構成オプションが含まれています。

- タイトル：ウィジェットのタイトル
- 説明：ウィジェットの説明
- エンティティ：表示するエンティティを 1 つ以上選択
- エレメント数：エンティティごとに表示するエレメントの最大数



ウィジェット「ライブラリ」と「マイウィジェット」

ダイアログでは、以下を選択できます。

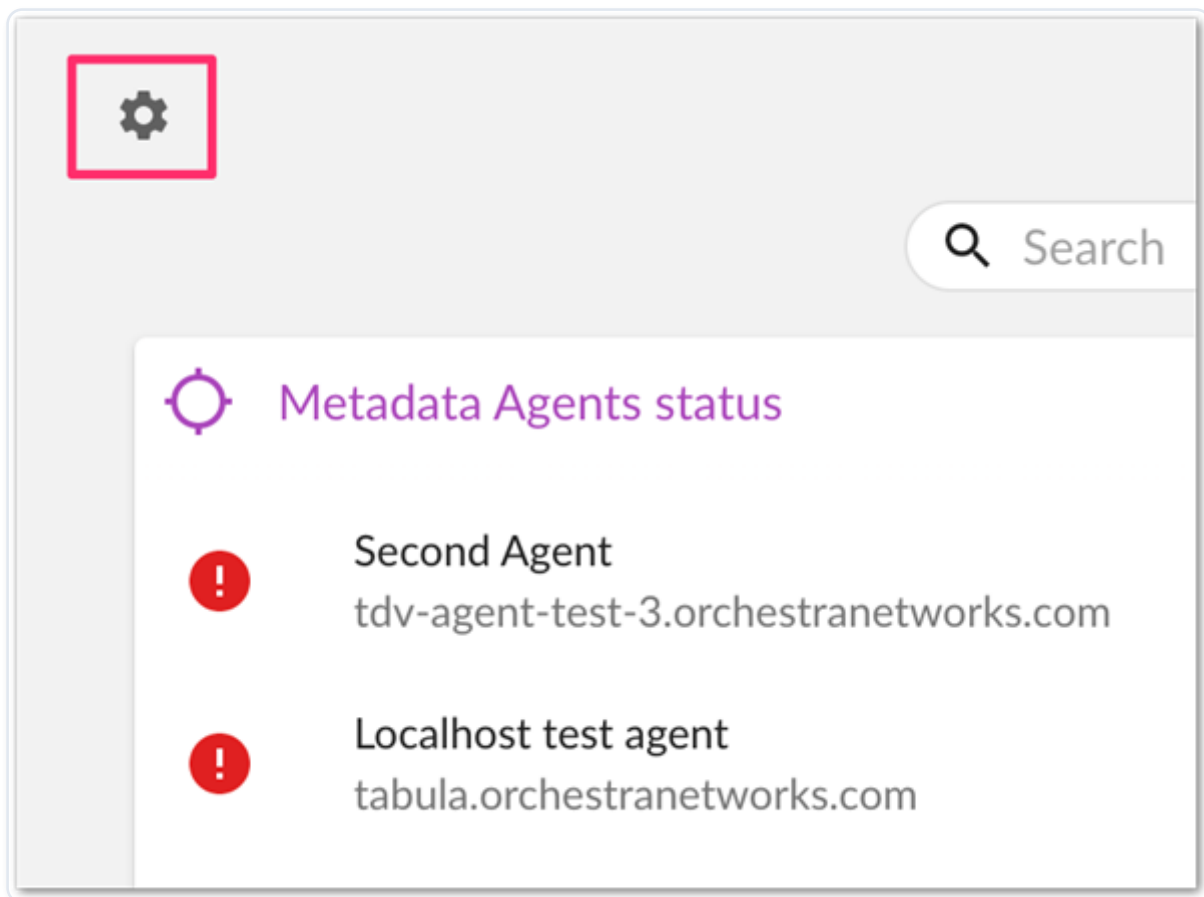
- 構成済みウィジェットまたは共有ウィジェットを表示する タブ。
- ウィジェットを表示するための タブ。

ウィジェットの作成/複製/更新/削除

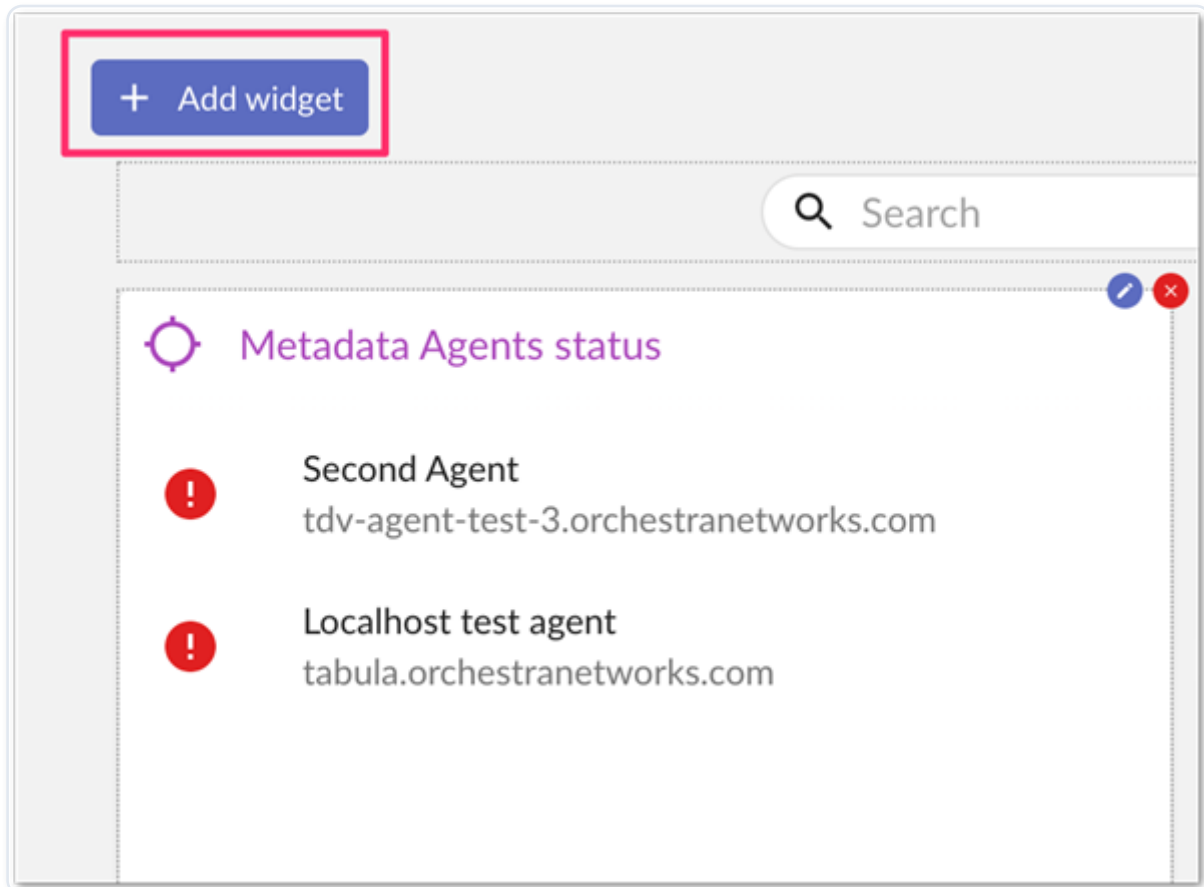
必要なウィジェットタイプを自由に作成できます。ウィジェットライブラリまたは共有ウィジェットから既存のウィジェットを複製できます。更新または削除できるのは、自分のウィジェットのみです。

ウィジェットの作成

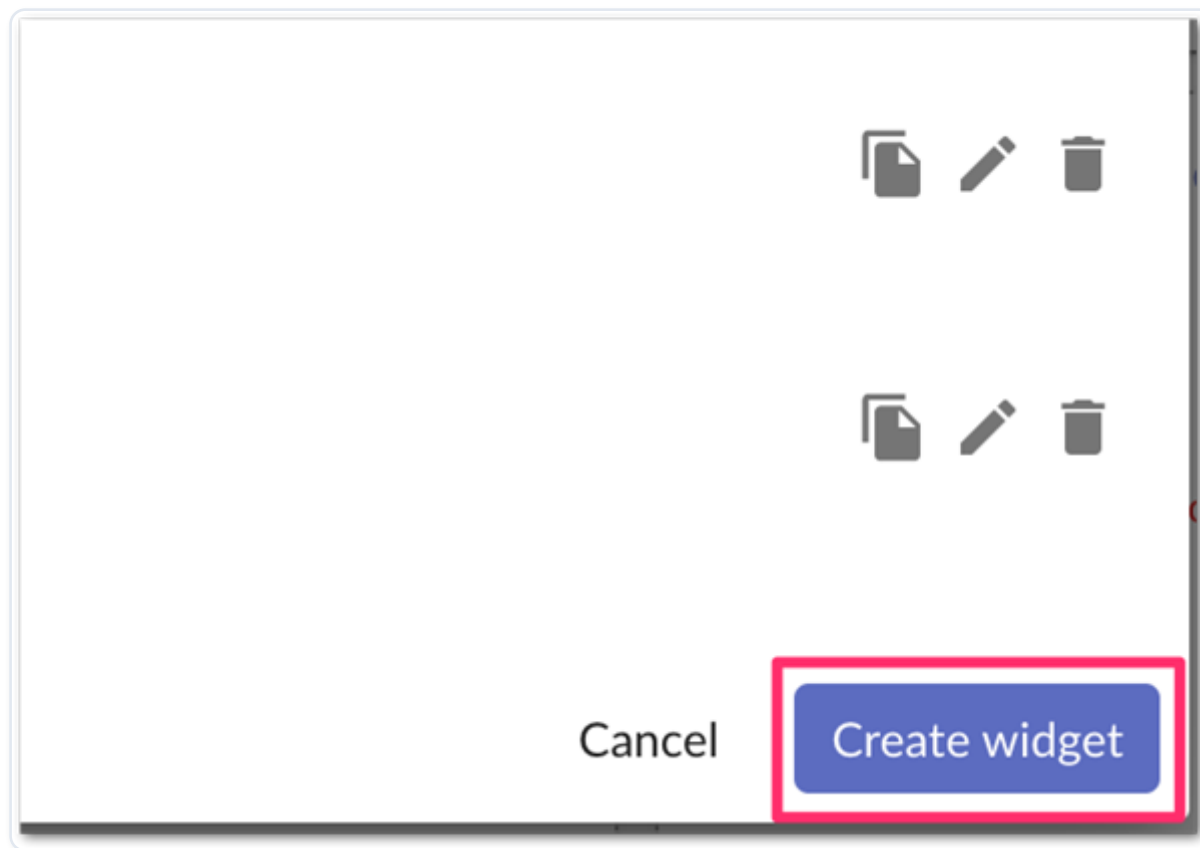
新しいウィジェットを作成するには、ホームページの左上隅にある歯車アイコンをクリックします。



これで、になりました。左上隅のをクリックします。



ウィジェットを作成できるダイアログが表示されます。を選択してウィジェット構成ウィザードを開き、作成プロセスを開始します。



ウィジェット構成ウィザードには次の手順が含まれます。

- ウィジェットの種類を選択します。
- ウィジェットの構成：ウィジェットの値と特定のウィジェットタイプの属性を構成します。
- ウィジェットの共有：ウィジェットの共有パラメーターを設定します。共有ウィジェットは他のユーザーが利用できます。
- ウィジェットの検証：ウィジェットの情報とプロセスの作成を確認します。

注意:

次のステップに進むには、各ステップで必要な情報をすべて入力する必要があります。入力するまで、ボタンは無効になります。

次の一連の画像は、ウィジェットの作成例を示しています。

Add widget

1 Select widget type

Please select the widget type you want to use for your custom widget

Widget type

Global Search

Next

2 Configure widget

3 Share widget

4 Create and add

Add widget

✓ Select widget type

2 Configure widget

Fill widget main information

Widget name*

Widget description

Previous

Next

3 Share widget

4 Create and add

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

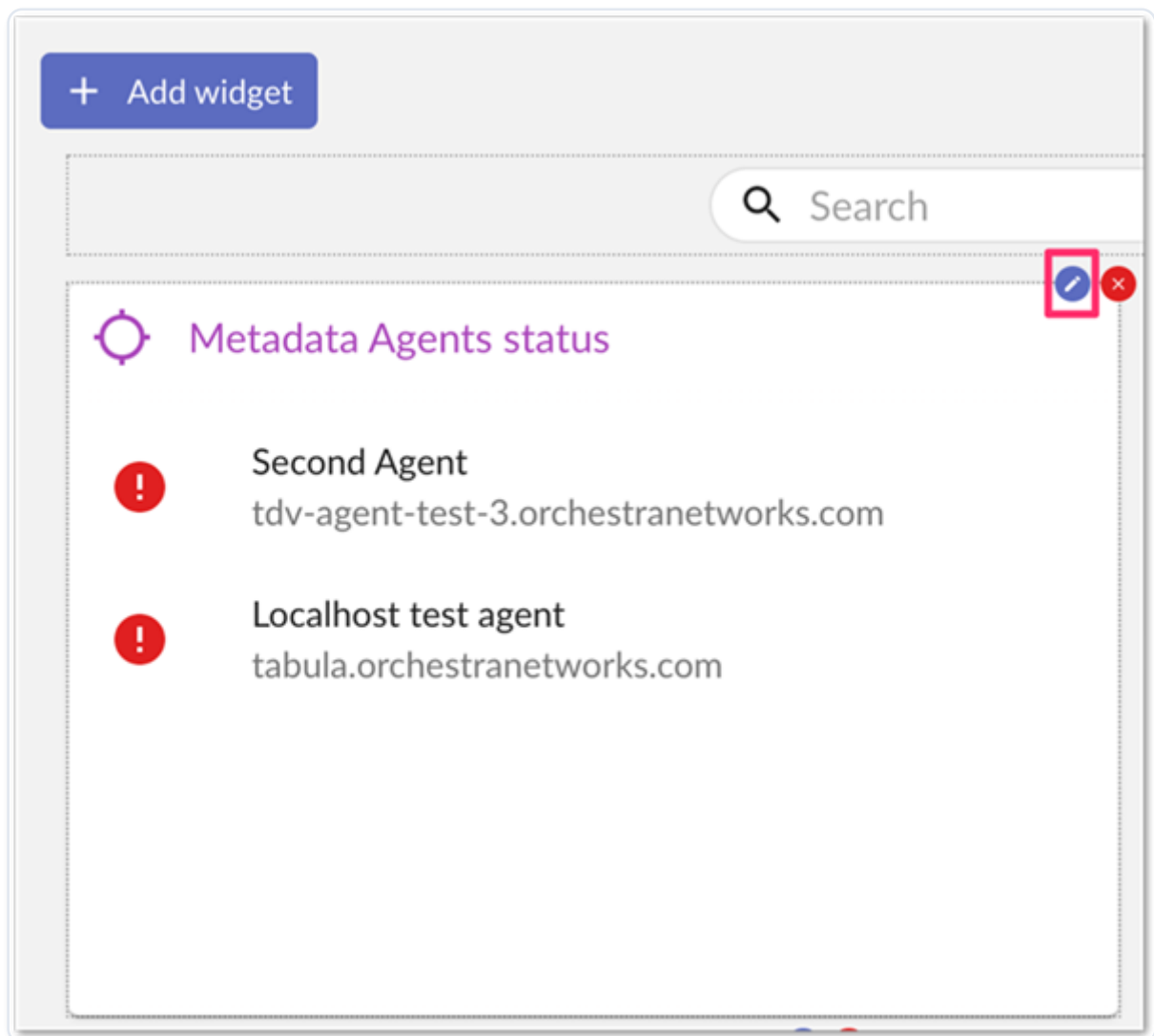
ウィジェットの変更

ウィジェットを変更する場合、ウィジェット構成ウィザードの簡易バージョンを使用して、タイプ属性、共有構成を変更し、変更を確認して適用することができます。

ウィジェットを変更するには、次の手順を実行します。

編集したいウィジェットが **ホームページ** にある場合

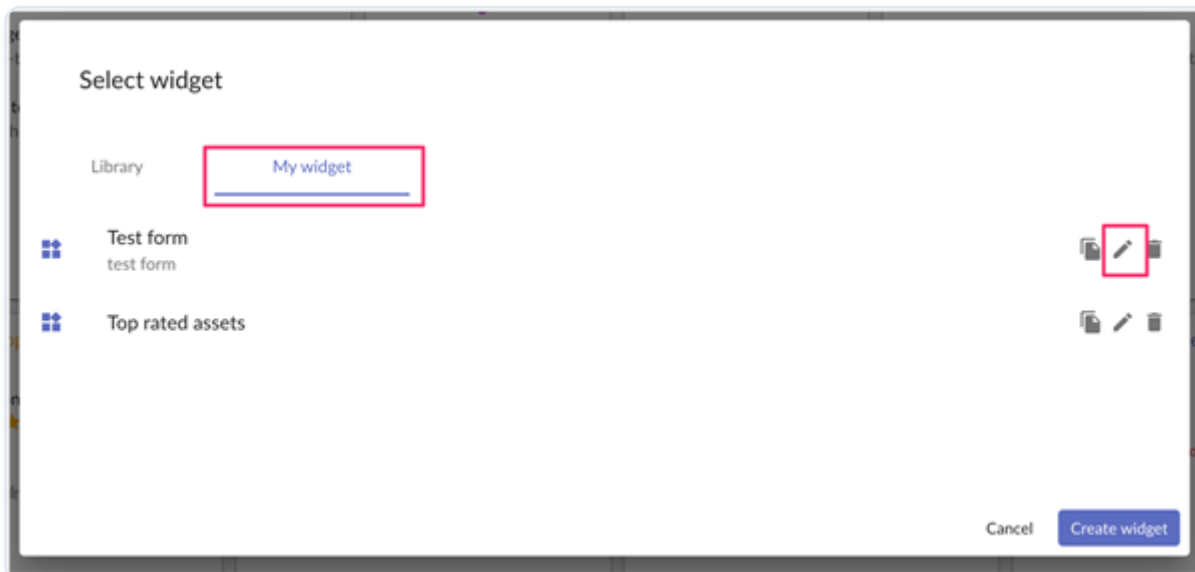
1. 歯車アイコンを選択して、**ホームページ** を編集します。
2. 変更したいウィジェットの アイコンを選択します。
3. ウィザードの手順を完了します。



編集したいウィジェットが **ホームページ** にはない場合は、次の手順に従います。

4. 歯車アイコンを選択して、**ホームページ** を編集します。
5. を選択し、表示されるダイアログで **タブ**を選択します。
6. 変更したいウィジェットの アイコンを選択します。

7. ウィザードの手順を完了します。



次の一連の画像はウィジェットの変更プロセスを示しています。

1 Modify widget**Fill widget main information**

Widget name *

Top rated Applications

Widget description

Display top rated Applications**Specifics parameters for widget type : Top rated elements**

Title *

Top rated Applications

Description

Display top rated Applications

Entity *

Applications

Number of elements *

5

Not empty and not negative

Predicate

**Next**

✓ Modify widget

2 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

3 Apply modifications

✓ Modify widget

✓ Share widget

3 Apply modifications

Are you sure to apply modification ? If you click on "Validate" the widget will be updated. If you are not sure, you can review your modifications by clicking on the "Previous" button.

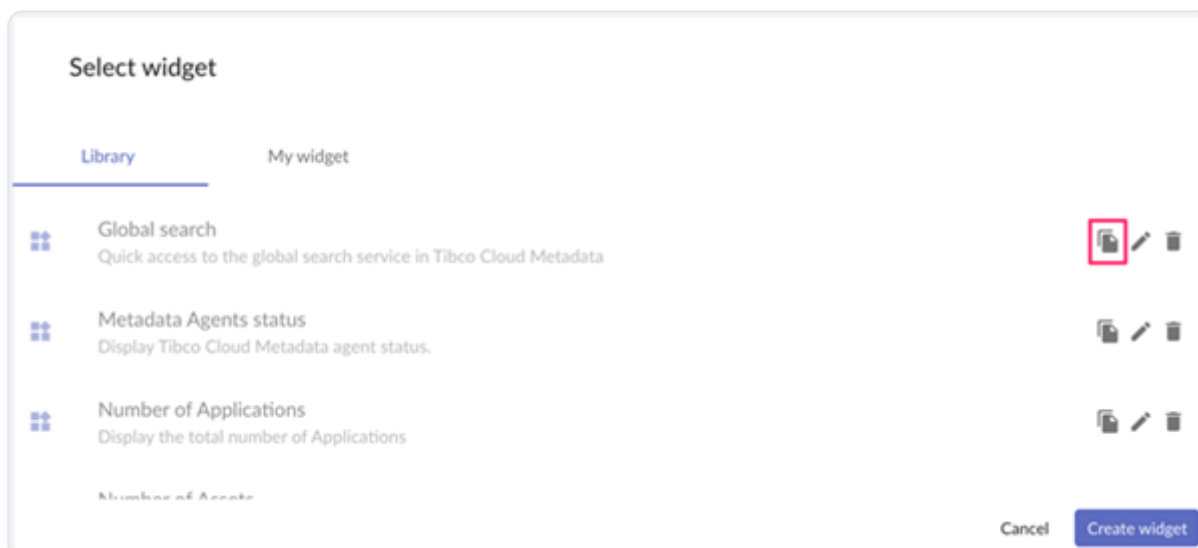
Previous

Validate

ウィジェットの複製

既存のウィザードを複製するには、次の手順を実行します。

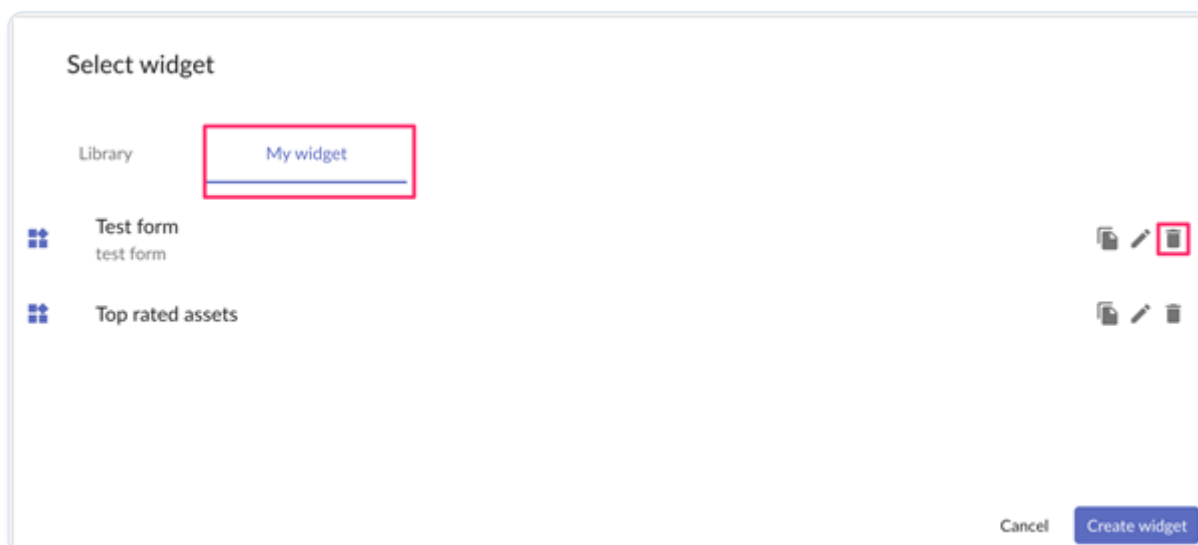
1. 歯車アイコンを選択して、**ホームページ** を編集します。
2. を選択します。
3. ライブラリまたはコレクションから目的のウィジェットを選択し、その複製アイコンを選択します。構成ウィザードが表示されます。
4. ウィザードの手順を完了してウィジェットを複製し、現在のホームページに追加します。



ウィジェットの削除

削除できるのは自分が作成したウィジェットのみです。ウィジェットを削除するには、次の手順を実行します。

1. 歯車アイコンを選択して、**ホームページ** を編集します。
2. を選択します。
3. タブを選択します。
4. 目的のウィジェットの削除アイコンを選択します。アクションを確認するとウィジェットが削除されます。



****注意：***ウィジェットが共有されていたり、ホームページで使用されていたりする場合は、この操作により、自分と他のすべてのユーザーからウィジェットが削除されます。

ホームページのカスタマイズ

別のウィジェットではなく特定のウィジェットを表示したい場合は、ホームページのカスタマイズによってそれを行うことができます。

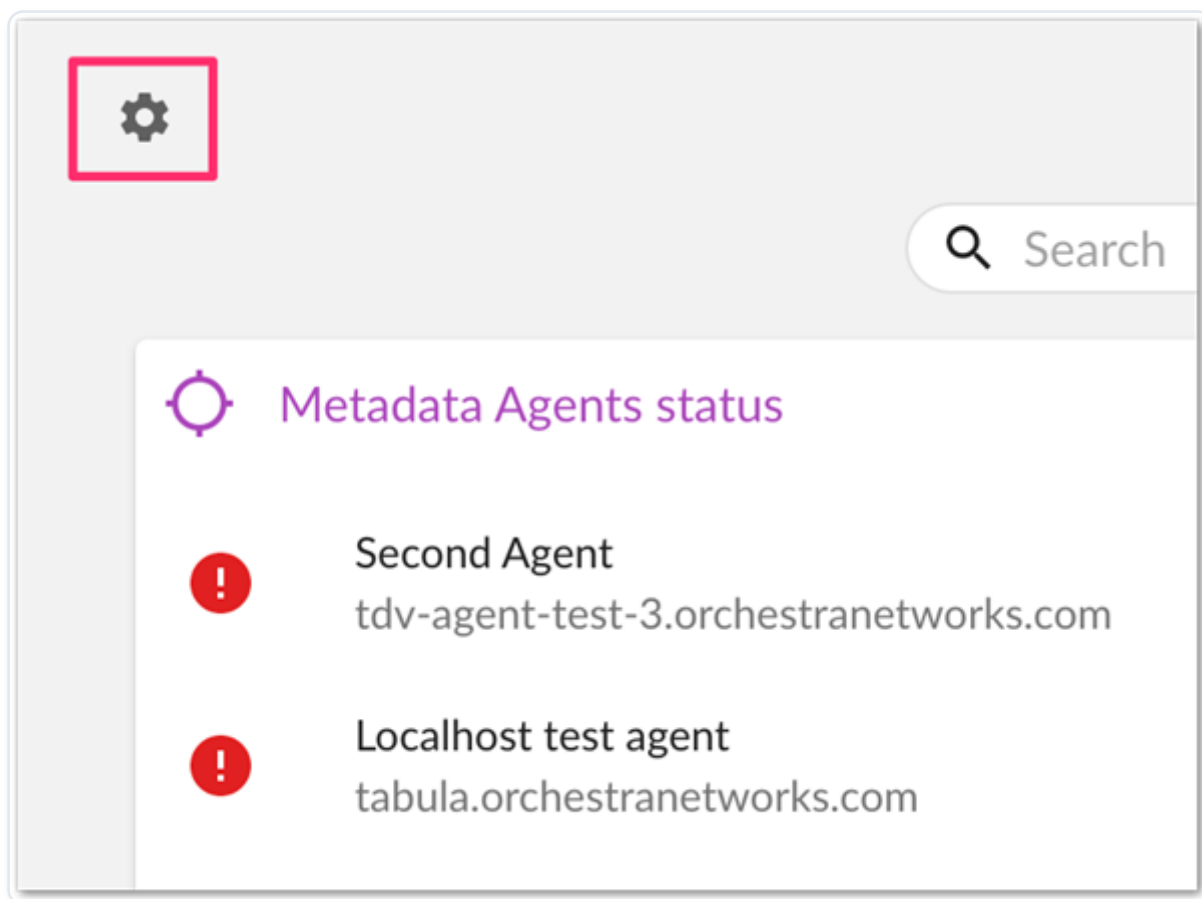
デフォルトのホームページまたは空白のホームページから開始する場合は、次の操作を実行できます。

- [ウィジェットの位置変更](#)
- [構成済みウィジェットの追加/削除](#)
- [自分で作成したウィジェットの追加/更新/削除](#)

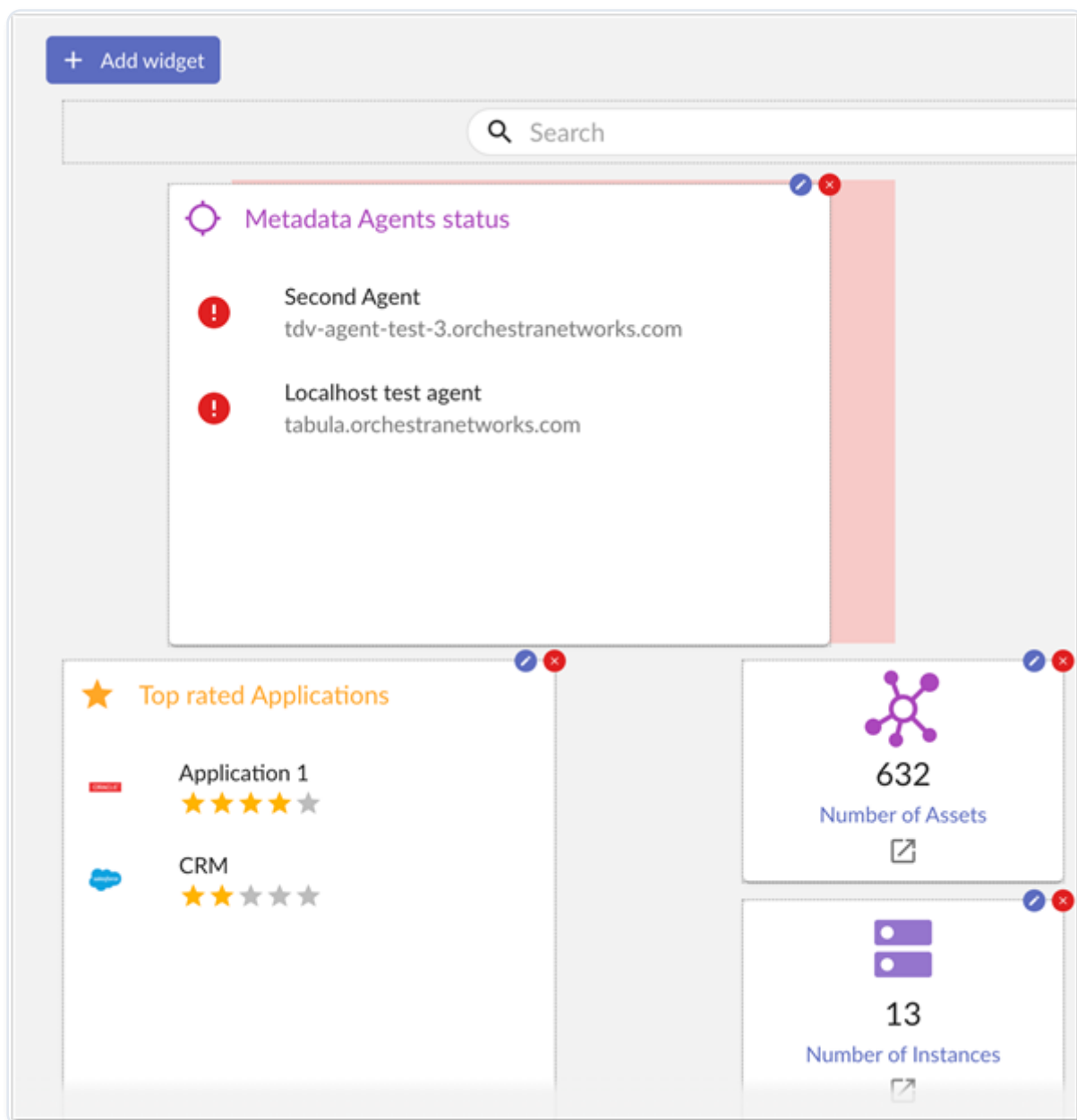
ウィジェットの位置変更

ウィジェットを移動するには、次の手順を実行します。

1. アイコンを選択します。



2. ウィジェットを選択して、目的の場所にドラッグします。



3. 変更内容を保存します。

構成済みウィジェットの追加/削除

ウィジェットを追加および削除するには、設定アイコンを選択して **ホームページ 編集モード**に入ります。

ウィジェットを削除するには、アイコンを選択します。

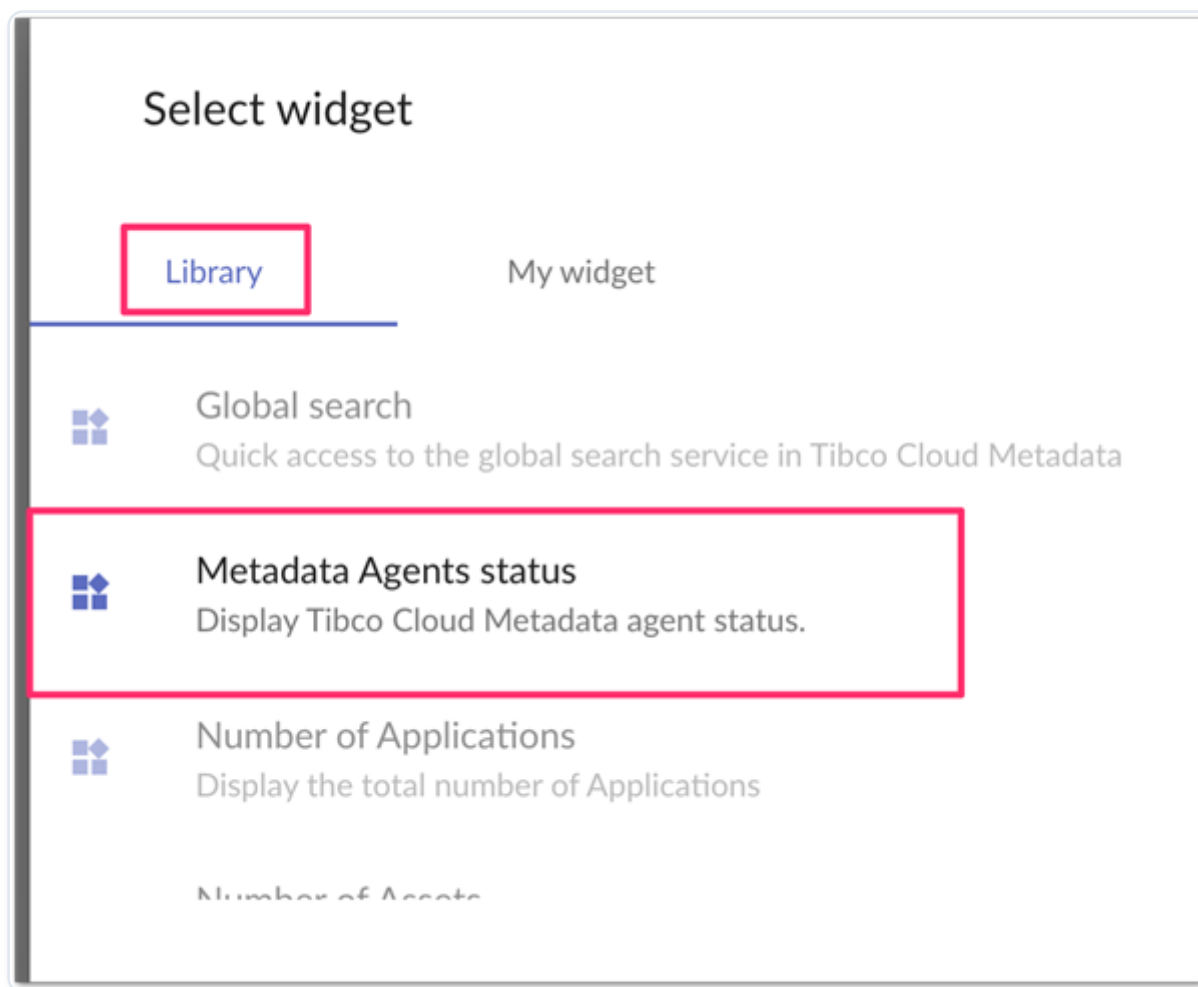
The screenshot displays a dashboard with the following widgets:

- Metadata Agents status**: Shows two agents with status icons (red exclamation marks).
 - Second Agent: `tdv-agent-test-3.orchestranetworks.com`
 - Localhost test agent: `tabula.orchestranetworks.com`
- Number of Assets**: Displays the value 632.
- Number of Instances**: Displays the value 13.
- Top rated Applications**: Lists applications with star ratings.
 - Application 1: ★★★★★
 - CRM: ★★★☆☆
- Top rated Assets**: Lists assets with star ratings.
 - public: ★★★★★

A red box highlights the top-right corner of the 'Metadata Agents status' widget, indicating it is the target for the next step in the procedure.

ウィジェットを追加するには、次の手順を実行します。

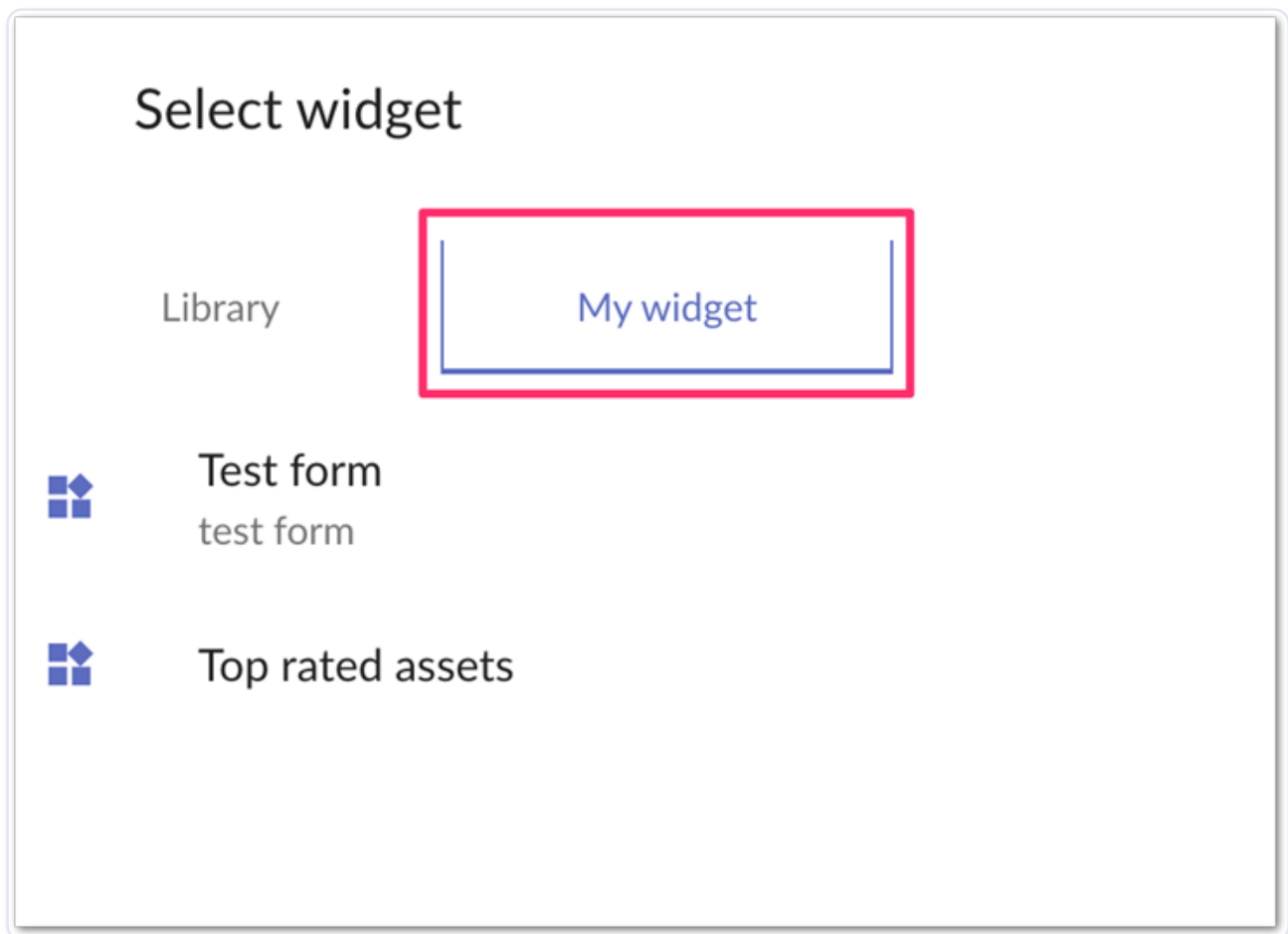
1. を選択します。
2. タブから、追加するウィジェットを選択します。

**注意:**

ウィジェットはホームページの最後に追加されます。追加したら、目的の場所に移動できます。ホームページにまだないウィジェットのみを追加できます。

作成したウィジェットの追加/更新/削除

前のセクションの手順を使用しますが、ダイアログボックスで タブを選択します。



カスタムビュー

- 概要
- 表形式のビュー
 - 新しいビューの作成
 - 既存のビューの適用
- 階層ビュー
 - 新しい階層ビューの作成

概要

EBX® メタデータ管理アプリケーションでは、対象ユーザーの要件に基づいてビューをカスタマイズできます。次のタイプのビューを作成できます。

- 表形式：表示されたレコードを並べ替えたりフィルタ設定したりするためのテーブルビュー
- 階層ビュー：関係に基づいて異なるテーブルのデータをリンクするツリービュー

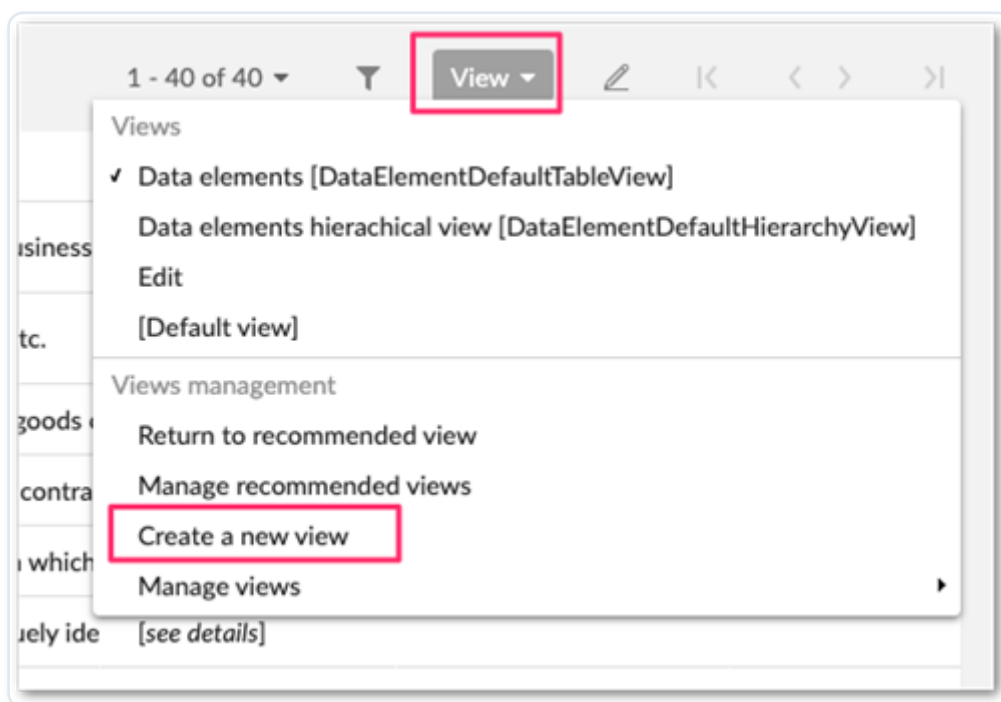
表形式のビュー

テーブルビューモードで、右上隅の ボタンを選択します。表示されるメニューでは、次の操作を実行できます。

- ビューの名前をクリックして既存のビューを適用します
- 十分な権限がある場合は、既存のビューを管理します
- 新しいビューを作成します

新しいビューの作成

新しいビューを作成するには、 を選択し、次に を選択します。



表示されるダイアログボックスでは、次の操作を実行できます。

- 名前の設定
- 所有者の設定
- ビューを共有するプロファイル/ユーザーの設定
- ビューモードの設定

新しい表形式ビューを作成するには、ビューモードを に設定します。ビューを他のユーザーと共有する場合は、次の操作を実行できます。

- プロファイルを設定します (****で識別されます)。ビューはプロファイルに含まれるすべてのユーザーと共有されます
- ユーザーを設定します。ビューは指定されたユーザーとのみ共有されます

作成プロセスを続行するには、ボタンをクリックします。次の画像に示すオプションと、以下で説明するオプションが表示されます。

The screenshot displays a configuration window with three main sections: **Displayed columns**, **Sorted columns**, and **Filter**.

Displayed columns: This section contains a list of columns under the heading "Displayed". The columns are: Identifier, Label, Description, Documentation, Data element type, Parent data element, and Logo. The "Identifier" column is selected and highlighted with a blue circle. To the right of this list is an empty box labeled "Displayable". Between the two boxes are four arrow icons: a double left arrow, a single left arrow, a single right arrow, and a double right arrow.

Sorted columns: This section contains a list of columns under the heading "Sorted". The columns are: Identifier, Label, Description, Data element type, Status, Critical data, Personal data category, and Regular Expression. The "Identifier" column is selected and highlighted. To the right of this list is an empty box labeled "Sortable". Between the two boxes are four arrow icons: a double left arrow, a single left arrow, a single right arrow, and a double right arrow.

Filter: This section contains two dropdown menus. The first dropdown is labeled "All criteria match" and the second is labeled "Add a criterion". Below these are two buttons: "Add a logical block" and "Remove all criteria".

At the bottom of the window are three buttons: "Back", "Cancel", and "Create".

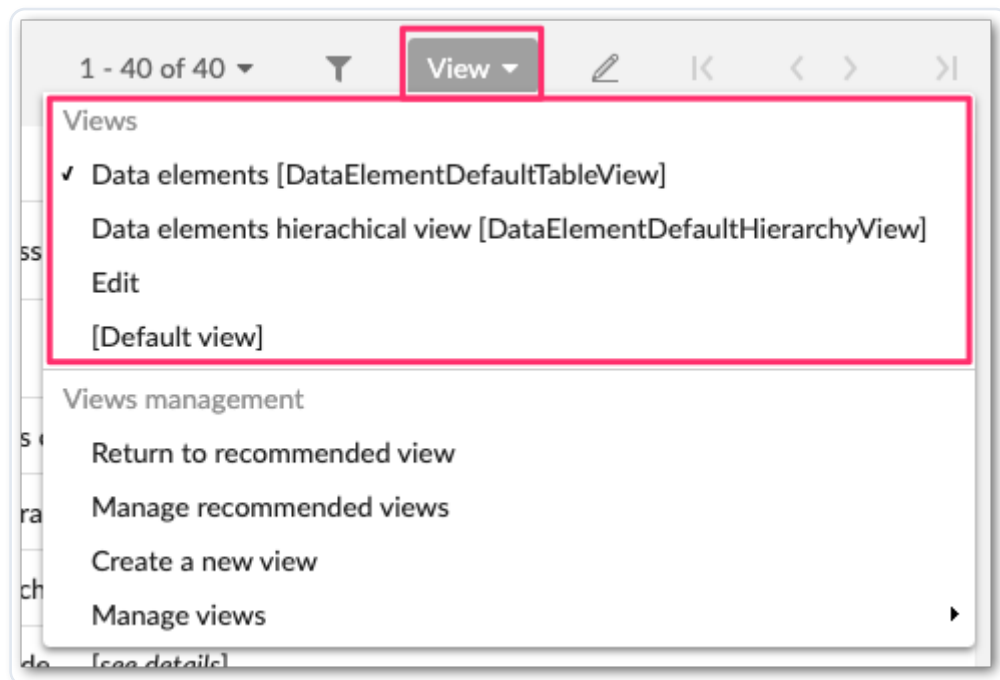
次のビューでは以下のことができます。

- 表示する列を選択します
- 列の順序を設定します
- フィルターを追加します
- ページ付け (ページあたりのレコード数) を設定します
- グリッド編集を有効にし、ツールバーを設定します。ツールバーは、テーブルの上部にある利用可能なアクションのリストに対応します。設定できる項目は次のとおりです。
 - テーブルビュー用の軽量汎用ツールバー
 - テーブルビュー用の汎用ツールバー

最後に、新しいビューを保存するか、 を選択して保存し、新しいビューが適用されたテーブルビューに戻ります。

既存のビューの適用

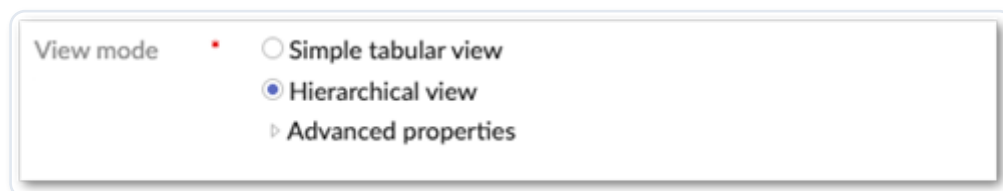
既存のビューを適用するには、メニューを開き、表示するビューを選択します。



階層ビュー

新しい階層ビューの作成

新しい階層ビューを作成するには、上記の表形式ビューを作成する手順に従います。ただし、ビューモードを選択する手順では、 を選択します。



を選択して、階層のディメンションとその他のオプションを定義します。次の画像とリストは、いくつかのオプションを示しています。

Dimension

Data elements

Asset (Assets)

Consent (Consents)

Consent date

Person (Persons)

Processing activity (Processing activities)

Display records in a new window ☐ Yes ☒ No

Prune hierarchy ☐ Yes ☒ No

Display orphans ☒ Yes ☐ No

Display root node ☒ Yes ☐ No

Root node label

English (United States)

French (France)

Toolbar on top of hierarchy [not defined]

Advanced properties

- 新しいウィンドウ内にレコードを表示：*** の場合、レコードを含む新しいウィンドウが開きます。それ以外の場合は、同じウィンドウの新しいページに表示されます*
- 階層の整理：*** の場合、子を持たず、ターゲットテーブルに属していない階層ノードは表示されません*
- 親なしの表示：*** の場合、親のない階層ノードが表示されます*
- ルートノードの表示：*** の場合、階層のルートノードはビューに表示されません*
- ルートノードラベル：階層ルートノードのローカライズされたラベル
- 階層の上部のツールバー：階層の上部にツールバーを設定できます
- 一致しない子を表示する：再帰的なケースで、検索フィルターが適用されると、検索中に一致するノードの一致しない子を表示できます
- 再帰的ルートリーフを削除：再帰的なケースでは、検索フィルターが適用されるか、モードが整理の場合、ルートリーフを表示から削除します
- サイクルの検出：再帰的なケースでサイクルの検出と表示を許可します。最も古いノードレコードがサイクルルートとして選択されます。これは検索モードやプルーニングモードでは機能しないことに注意してください

を選択して、階層の各ノードをどのように表示するかを定義します。

The screenshot shows a web interface for configuring data elements. At the top, there are two tabs: 'Consent' (selected) and 'Data elements'. Below the tabs, there's a 'Labels' section with two rows. The first row is for 'English (United States)' and the second for 'French (France)'. Both rows have a text input field containing the path '\$./id) - \$./person) - \$./consentDate' and a small icon to the right. Below the labels, there's a 'Filter' section with a dropdown menu currently showing 'All criteria match' and a button 'Add a criterion'. At the bottom of the filter section are two buttons: 'Add a logical block' and 'Remove all criteria'.

このビューでは次のことができます。

- ノード名をクリックしてノードに切り替えます
- ラベルを定義します
- フィルターを追加します

最後に、ビューを作成するには [ここ](#) を選択し、作成後に適用するには [ここ](#) を選択します。

高度なデータサービス

- [高度なデータサービスの概要](#)
- [メタデータの収集](#)
- [資産のサンプリング](#)
- [資産分類](#)
- [リネージ](#)

高度なデータサービスの概要

高度なデータ サービスを使用すると、TIBCO® Data Virtualization (TDV) ツールに接続して、システムにすでに存在するデータソースからメタデータを収集したり、EBX® メタデータ資産を使用して TDV で集約ビューを作成したりできます。この機能を使用するには、システムに TIBCO® メタデータエージェントをインストールして構成し、登録しておく必要があります。

管理者アクセス権がある場合、詳細については、[TIBCO® メタデータエージェントのインストール](#) を参照してください。

メタデータの収集

- [概要](#)
- [収集サービスの開始](#)

- 収集のメインビュー
- 新しい構成の作成
 - データプロファイリング
- 実行構成
- 既存の構成の使用
- 収集のスケジュール
- インスタンスからのデータソースの作成
 - 前提条件
 - 作成と収集
- 具体的な事例
 - REST リソースの収集

概要

メタデータ収集は、システム内に存在し、TDV によってサポートされているデータソースからメタデータを収集し、EBX® メタデータ管理アプリケーション 資産として保存するプロセスです。

TDV ですでに構成されているデータソース、または EBX® メタデータ管理アプリケーションで定義された新しいソースに対して収集を実行できます。

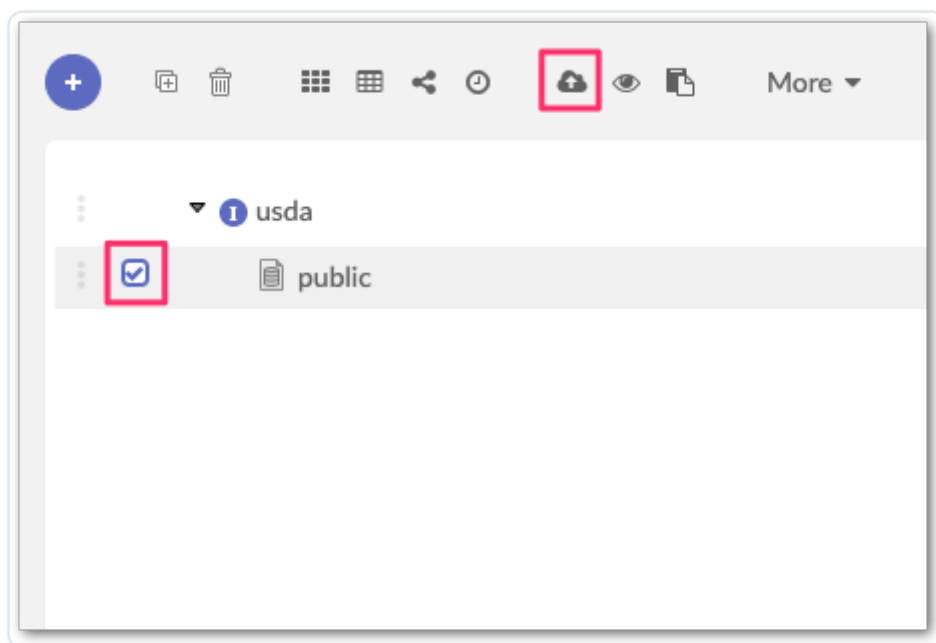
資産または資産の選択から **収集** サービスを開始する場合は、すべての資産がメタデータエージェントによって以前に収集されていることを確認してください。

インスタンスまたは選択したインスタンスから **Harvesting** サービスを開始すると、まだ存在しないデータソースを TDV に作成できます。詳細については、[インスタンスからのデータソースの作成](#) を参照してください。

Harvesting サービスは、無効な資産またはインスタンスでは実行できません。

収集サービスの開始

資産階層ビューまたはインスタンス ビューで、メタデータを収集する 1 つ以上の資産またはインスタンスを選択します。クラウドアイコンを選択して収集を開始します。収集のメインビューが表示されます。



収集メインビュー

収穫のメインビューには、現在の選択内容と既存の構成が表示されます。

既存の構成とは、現在の選択に関連する 1 つまたは複数のエレメントに対してすでに実行された収集です。既存の構成が利用できない場合は、以下のスクリーンショットのようなメインビューが表示されます。

Existing configurations relating to current selection

Harvesting configurations might already exist for your selection.

A configuration will be proposed here as soon as it shares at least one asset with a selected element.

Your current selection :

 Instances (0) ▼

 Assets (1) ▼

No existing configuration found for your current selection

Close

New configuration

既存の構成がある場合は、次のスクリーンショットのようなものが表示されます。

Existing configurations relating to current selection

Harvesting configurations might already exist for your selection.

A configuration will be proposed here as soon as it shares at least one asset with a selected element.

Your current selection :

 Instances (1) ▼

 Assets (0) ▼

Existing configuration related to your current selection :

Label	Instances	Assets	Status	First execution date	Last execution date	Next execution date	Duration of last run	Actions
Movie DB	0	0	Created	-	-	-	-	    

Close

New configuration

既存の構成から、次のアクションを実行できます。

- 実行

- スケジュール
- 構成
- 複製
- 現在の選択範囲を拡張

詳細については、[既存の構成の使用](#) を参照してください。

新しい構成の作成

メインビューから、右下の ボタンをクリックします。新しい構成画面が表示されます。

New configuration

Options :

☐ Include data sampling ☐ Force sampling

☐ Automatic classification suggestions ☐ Force refresh of classification suggestions

☐ Classify assets automatically ☐ Include lineage

☐ Include profiling

Select item(s) you want to harvest :

10 ▼ rows per page

☒ public

この画面では、まず構成名を追加します。これは、より多くのアクションを表示するために必要です。

New configuration

My configuration

Options :

☐ Include data sampling ☐ Force sampling

☐ Automatic classification suggestions ☐ Force refresh of classification suggestions

☐ Classify assets automatically ☐ Include lineage

☐ Include profiling

Select item(s) you want to harvest :

10 ▼ rows per page

☒	▶		public
-------------------------------------	---	--	--------

Back

Save Save and Run

この画面の上部に、次のオプションが表示されます。

- データサンプリングを含める：サンプリングの対象となる資産については、利用可能な場合はソースシステムからデータサンプルが取得されます。サポートされているタイプの詳細については、[資産サンプリング](#) を参照してください。
- 強制サンプリング ([データ サンプリングを含める](#) を自動的に有効にします)：資産に対してデータ サンプリングを強制します (既存のサンプルを上書きします)。
- 自動分類の提案：収集と同時に資産の分類も開始されます。
- 分類提案の強制更新 ([自動分類提案](#) を有効化)：資産にすでに分類提案がある場合、このオプションを選択すると、収集によって分類提案が強制的に更新されます。
- 資産の自動分類 ([自動分類提案](#) を有効化)：一致する可能性のあるリストを表示する代わりに、資産に分類提案を適用します。
- リネージを含める：収集に TDV 公開ビュータイプの資産が含まれている場合、リネージが抽出され、それらの資産が保存されます。リネージは非常に複雑で、多数のリソースに関係する可能性があるため、このオプションは即時収集実行ではなく、スケジュールされた実行に対してのみ有効にすることをお勧めします。
- プロファイリングを含める：対象となる資産 (DB テーブル、CSV ファイル、Excel ワークシート) のカーディナリティ統計およびその他のプロファイリング情報が収集結果に含まれます。統計のクエリは大規模なデータソースでは時間がかかる可能性があるため、このオプション

ンは即時収集実行ではなく、スケジュールされた実行に対してのみ有効にすることをお勧めします。

階層ビューでは、複数のアクションを実行できます。



1)	このアクションを使用すると、すべての子ノードの選択を解除し、親ノードを選択したままにすることができます。これは、ノードに子がある場合にのみ使用できます。
2)	収集するノードを選択または選択解除します。ノードに子がある場合、このアクションによりすべての子も選択/選択解除されます。
3)	子要素をさらに展開するか、展開する子要素を減らします。このアイコンをクリックすると、子要素が存在する場合は表示されます。存在しない場合は、データなしメッセージが表示されます。TDV との接続に問題がある場合は、ここでエラーが表示されることがあります。



これはアクションではなく、ロゴ付きのインスタンス/資産の名前です。サブタイトルには、次のストレージ情報が含まれます: - 新規：これは新しいインスタンス/資産です - 保存済み：インスタンス/資産はすでに EBX® メタデータ管理アプリケーションにあります。

設定に名前を付け、収集対象として少なくとも 1 つの要素を選択すると、下部に ボタンとボタンが表示されます。

- 保存：設定のみを保存します。保存した後、必要に応じてスケジュールを設定できます。詳細については、[収集のスケジュール](#) を参照してください。
- 保存して実行：設定を保存して収集を実行します。このアクションにより、収集の画面が表示されます

データプロファイリング

データプロファイリングは、次の資産タイプで利用できます。

- DB テーブルとその列
- CSV ファイルとその列
- Excel ワークシートとその列

そのような資産の 1 つが収集範囲内 (ルート 資産または子孫として) に存在する場合は、その資産のデータプロファイリングをアクティブ化できます。

データプロファイリングは物理データソースから収集されるまでに時間がかかる可能性があるため、このオプションはスケジュールされた構成に対してのみ有効にし、直接実行では使用しないことをお勧めします。

データソースから次の情報を取得できます。

- 完了率：合計値のうち null 以外の値の割合を表示します。
- 一意性率：合計値のうち一意の値の割合を表示します。
- 最も頻度の高い値：この列で最も一般的な上位 5 つの値をリストします。
- 最小値：この列の最小値を表示します。
- 最大値：この列の最大値を表示します。

false	false	∅
DATA TYPE	COMPLETENESS RATE	UNIQUENESS RATE
numeric	1	0.72
MOST FREQUENT VALUES	MINIMAL VALUE	MAXIMAL VALUE
22.00 7%	14.95	215.99
129.99 2%		

実行構成

構成ビューから を選択すると、収集が開始されます。最初にアプリケーションはTDVと通信する必要があるため、進行状況ビューが表示されます。

その後、進捗状況表が表示されます。

Gathering harvested metadata					
You can see the progression of your job during the harvesting. At the end, you can preview results before validation.					
Job ID	Instances	Assets	Status	Results	Action
1ceb12f8-437a-45ea-97e1-7042dd686efe		/shared/examples/movieDatabase/public ...	✓ Success	Preview	☒ Accept results

[Back](#)
[Validate and return to main view](#)

このテーブルは、収穫のステータスを表示するために自動的に更新されます。次のように、実行可能なアクションは複数あります。

- または 列で、 を選択すると、インスタンス/資産の完全なリストが表示されます。リストはポップアップに表示されます。ポップアップの外側をクリックすると、ポップアップが閉じます。
- 列で、ステータスが になったら、 を選択して収集結果を表示できます。ステータスが または の場合、このボタンは使用できません。
- 結果が正しい場合は、チェックボックスをオンにして承認できます。このチェックボックスはデフォルトでオンになっています。
- を選択すると、次の操作を実行できます。
 - がチェックされている場合、結果を受け入れて資産テーブルに追加します。メインビューにリダイレクトされます。
 - がチェックされていない場合は結果を拒否します。メインビューにリダイレクトされます。
- メインビューに戻るには、 ボタンをクリックします。収集がまだ検証されていない場合は、メインビューからこの進行状況ビューに戻ることができます。

結果を検証すると、メインビューで構成ステータスが更新されたことが表示されます。

既存の構成の使用

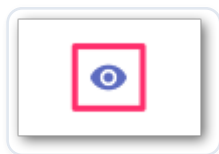
メインの収集ビューから、既存の構成がある場合は、次のアクションを実行できます。



1)	構成を実行します。進行状況ビューが表示され、次に進行状況テーブルが表示されます。
2)	構成をスケジュールします。
3)	現在の構成を構成します。構成画面が表示されます。
4)	現在の選択内容で構成を拡張します。構成画面が表示され、収集範囲が現在の選択内容と結合されます。

5)

現在の構成を複製します。この構成のコピーが構成リストに表示されます。



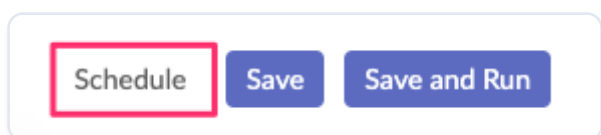
進捗状況テーブルビューを参照してください。このオプションは、構成ステータスが **承認待ち** の場合にのみ使用できます。

を選択すると、構成画面が表示されます。この画面は、**新しい構成の作成** セクションで呼び出された画面に似ていますが、インスタンス/資産階層、構成名、構成オプションは事前に入力されています。これらを変更して保存することができます。

収集のスケジュール

既存の収集構成の実行を 1 回または複数回スケジュールできます。スケジュールするには 2 つの方法があります。

- 設定ビューから、設定を保存した後、下部にある **スケジュール** を選択します。



- メインビューから **カレンダー** を選択します。



ボタンをクリックすると、スケジュールを構成できるダイアログボックスが表示されます。

Please fill the form below to create your scheduled task

Date 

Date is empty

Interval Unit

Repeat* ☐ Infinite

スケジュールを保存するには、日付を挿入する必要があります。日付：このフィールドは必須です。ここに日付と時刻を入力します。右側の矢印を使用してカレンダーを表示できます。繰り返しが設定されている場合の複数のスケジュール間の間隔：間隔の単位。デフォルトでは分単位ですが、時間または日に変更できます。繰り返し：スケジュールの繰り返し回数。この収集を無限に繰り返すには、チェックボックスをオンにします。

例：毎月の収集をスケジュールします。日付は今年 01/01、間隔は 28 日、繰り返しは 12 です。

収集をキャンセルする場合は、構成がすでにスケジュールされていることを確認する必要があります。メインビューから確認すると、構成のステータスが で、次の実行日が示されています。

Existing configuration relates to your current selection :

Label	Instances	Assets	Status	First execution date	Last execution date	Next execution
USDA harvesting	0	78	Scheduled	2020-04-07 12:38:38.313	2020-04-07 13:25:34.052	2020-04-07 20:30

設定の近くにあるカレンダーアイコンを選択すると、スケジュール ダイアログボックスに ボタンが表示されます。これをクリックすると、スケジュールがキャンセルされます。

Please fill the form below to create your scheduled task

Date

Date for the next harvesting execution

Interval Unit

Repeat*

Close Remove Schedule

インスタンスからのデータソースの作成

TDV でまだ構成されていないデータソースのメタデータを収集する場合は、EBX® メタデータ管理アプリケーションから直接構成できます。

前提条件

そのためには、まず、インスタンスの属性に必要な情報がすべて提供されていること、およびインスタンス タイプが EBX® メタデータ管理アプリケーションによる作成でサポートされていることを確認する必要があります。

現在、次のものが作成できます。

- リレーショナルデータソース
- Excel、XML、WSDL、CSV などのファイルベースのデータソース
- SOAP データソース
- REST データソース (これらのデータソースの場合、データソースを作成した後、TDV で操作を手動で作成する必要があります)

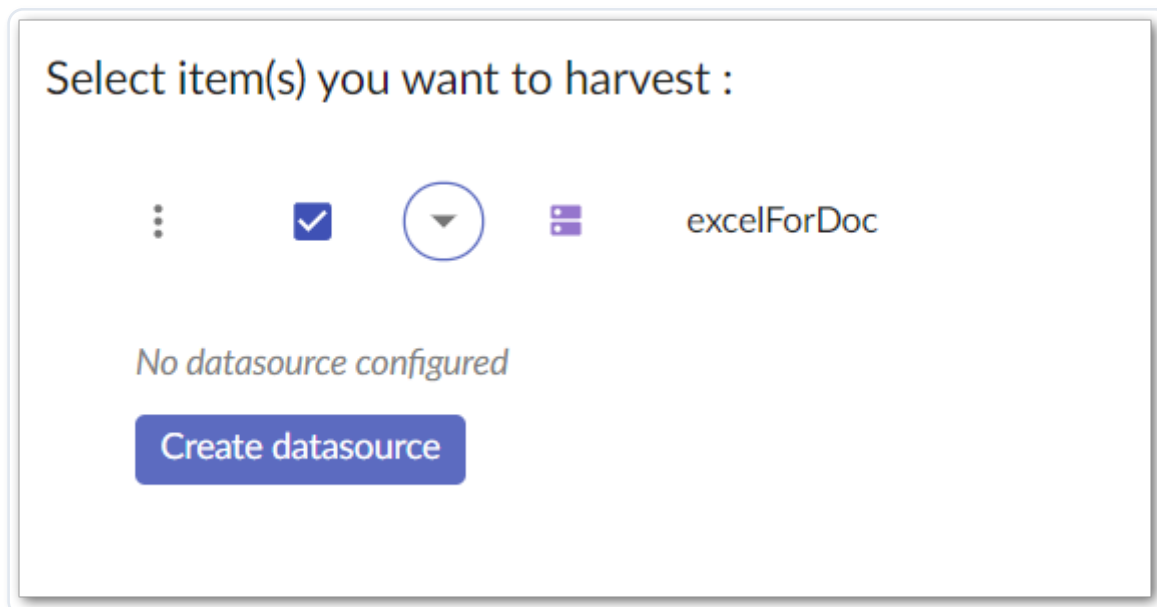
サポートされているアダプタの完全なリストは、- のテーブル属性 **Creatable?** = **Yes** で確認できます。

TDV でデータソースを作成するために使用するインスタンスは、TDV から物理データソースに接続するために必要なプロパティを提供する必要があります。以下は、データソースの種類に応じた必須属性のリストです。

作成と収集

[新しい構成の作成](#) セクションの説明に従って、インスタンスの新しい収集構成の作成を開始します。

クリックしてインスタンスの子を展開すると、ボタンが表示されます。



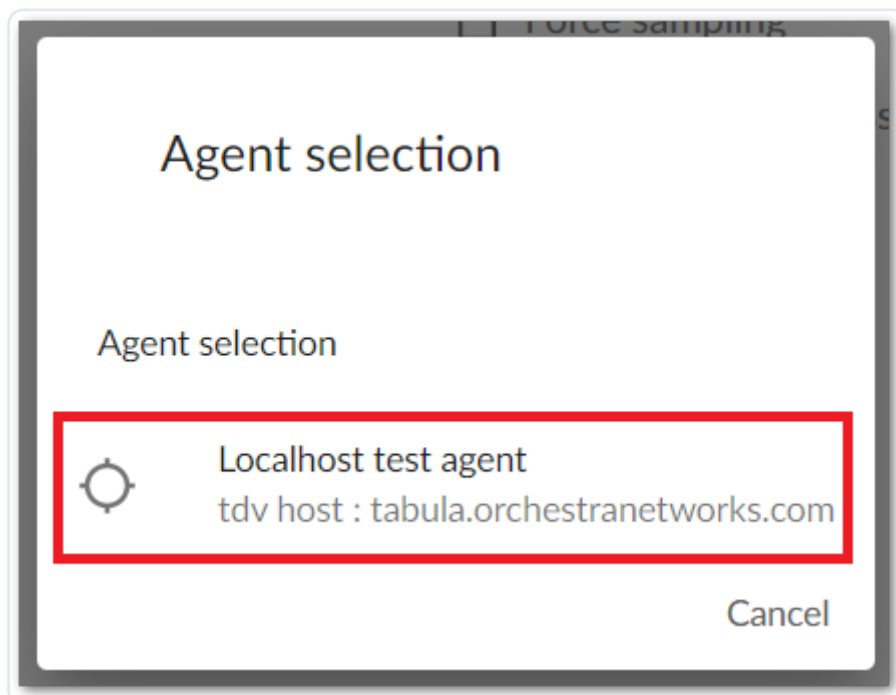
Select item(s) you want to harvest :

⋮ ☒ ▼  excelForDoc

No datasource configured

Create datasource

これをクリックすると、使用可能なメタデータエージェントのリストが表示されます。データソースの作成に使用するエージェントをクリックして、プロセスを起動します。



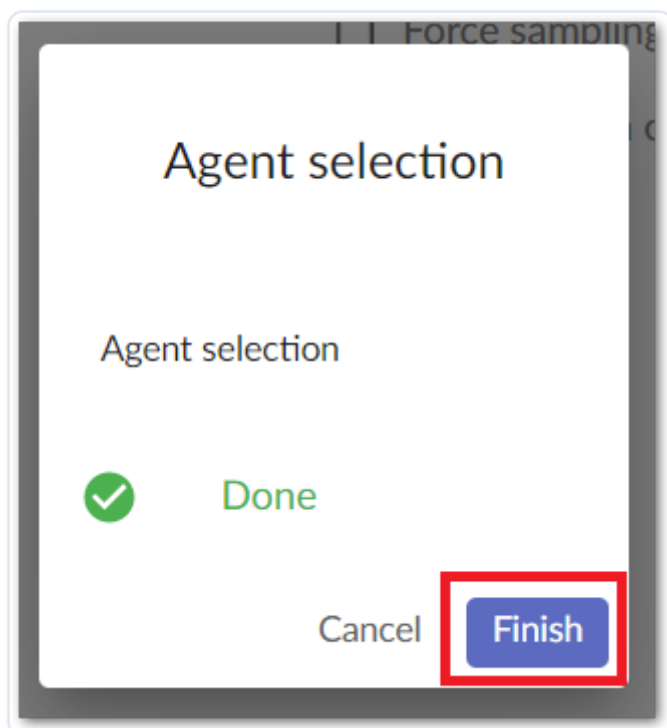
Agent selection

Agent selection

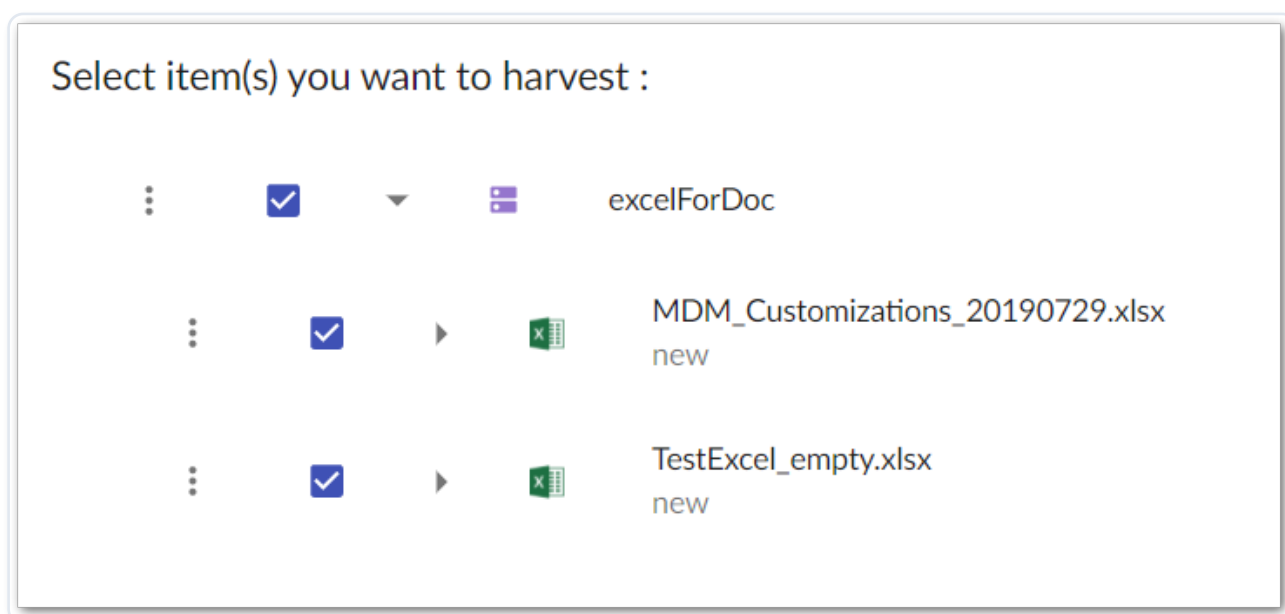
 Localhost test agent
tdv host : tabula.orchestranetworks.com

Cancel

プロセスが完了するまで待ちます。作成が成功した場合は、緑色で「完了」というメッセージが表示されます。 を選択してプロセスを終了します。



これにより、収集画面が再読み込みされ、インスタンスの直接の子を確認できるようになります。



前のセクションで説明したように、収集構成を続行できます。

具体的な事例

REST リソースの収集

EBX® メタデータ管理アプリケーションは、TDV で手動で文書化された REST リソースの収集をサポートします。TDV でリソースが文書化される方法により、特定の動作が発生します。実際、EBX® メタデータ管理アプリケーションは、TDV とは異なり、RESTful アーキテクチャに従って REST リソースを表します。

REST リソースの表示方法は次のとおりです。

- REST Datasource :
 - REST Resource :
 - REST Operation
 - REST Request
 - Request elements
 - REST Response
 - Response elements

階層内の の下にある資産の構成を収集することはできません。これらのリソースはエージェントによって作成され、TDV には存在しないためです。

REST データソースからの収集の構成中に、TDV に登録されたすべての操作が表示されますが、EBX® メタデータ管理アプリケーションによって作成されるため、REST リソース 資産は表示されません。

資産のサンプリング

- [概要](#)
- [サンプリング可能な資産タイプ](#)
- [サンプルのリクエスト方法](#)
- [サンプリング資産選択の誤り](#)
- [XML エLEMENTのサンプリング](#)

概要

資産サンプリングを使用すると、ファイル内のサンプルデータを要求して資産の内容を確認できます。デフォルトでは、サンプルにはターゲットソースの上位 100 行が含まれます。

資産ビューから、資産がサンプリングに使用できる場合はサンプルファイルをリクエストできます。

サンプリング可能な資産タイプ

次の資産タイプのサンプルをリクエストできます。

- DB テーブル (データベーステーブル)
- Excel ワークシート
- CSV ファイル
- TDV 公開ビュー

- XML エlement (オブジェクトルートとしてマーク)

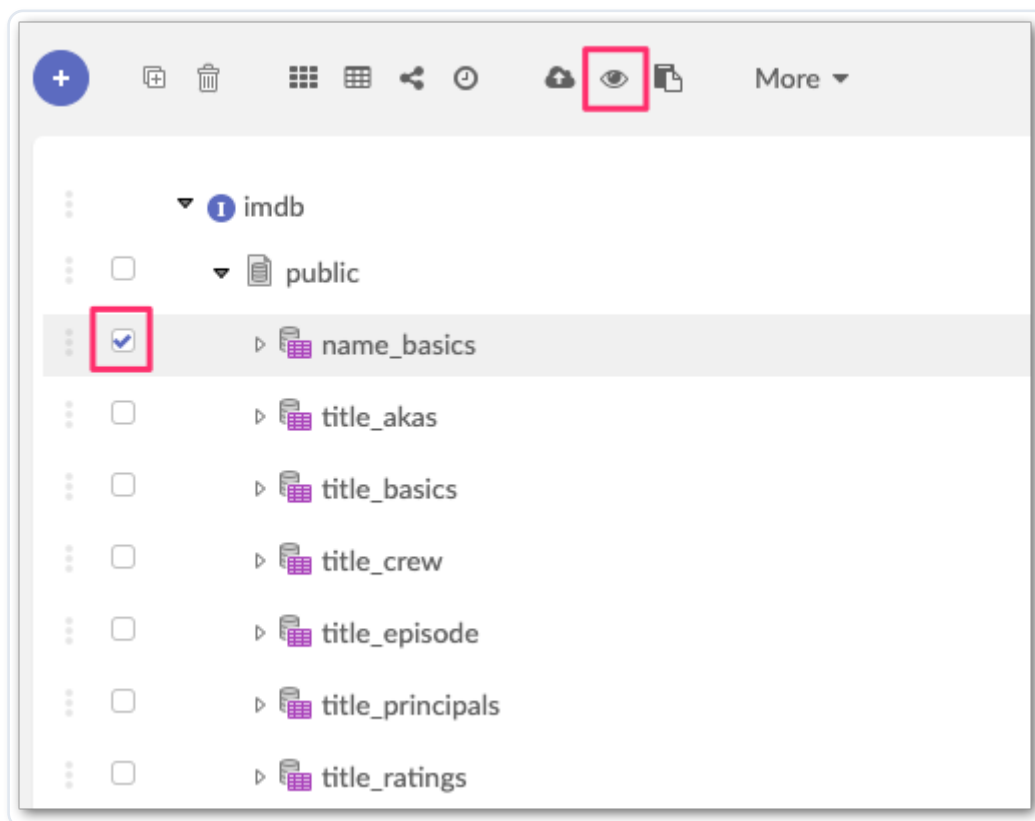
無効な資産ではサンプリングを実行できません。

サンプリングは、[メタデータ収集](#) サービスを使用して作成された資産に対してのみ実行できます。

サンプルのリクエスト方法

まず、メニューで [サンプル](#) を選択して、資産のビューを表示します。

階層ビューで、サンプルファイルが必要な資産を 1 つ以上選択します。完了したら、上部の目のアイコンを選択してサンプルをリクエストします。



進行状況メッセージと、完了すると成功メッセージが表示されます。

サービスを終了して資産に戻り、サンプルファイルをダウンロードできます。資産ビューで、フィールドを見つけて、マウスポインタをファイルの上に置きます。ツールヒントに ボタンが表示されます。

name_basics

INSTANCE: IMDB

DISABLED: false

MANDATORY: ∅

ENCODING: ∅

PARTITIONED DB TABLE: ∅

TABSPACE: ∅

ENCRYPTION: ∅

SAMPLE FILE

180272_name_basics_Sample.csv

BACK NEXT

OWNER: ∅

CREATOR: ∅

CREATION DATE: ∅

UPDATE DATE: ∅

Close

これで、ファイルを開くことができます。

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

サンプリング資産選択の誤り

サンプリングに使用できない資産を選択した場合は、エラーメッセージが表示されます。

Type 'DB Schema' of asset 'public' is not supported. (Supported types are DB_TABLE, EXCEL_WORKSHEET, CSV_FILE and TDV_PUBLISHED_VIEW)

XML エLEMENTのサンプリング

XML 構造の場合、XML ELEMENT タイプの資産に基づいて XML 形式のサンプルを取得できます。

XML ドキュメントの構造はオープンであり、さまざまな角度から分析できるため、XML 内のどのオブジェクトをサンプリングするかを定義できます。

これを実現するには、対応する資産を編集用を開き、プロパティまで下にスクロールして、に設定します。

Main	Referenced assets	References	Incoming flows	Outgoing flows
Owner			[not defined]	
Instance		*	DemoXml	
Data element			[not defined]	
			[No suggestions]	
Data element automatically assigned *		☒ Yes ☐ No		
Classification Ratio		0 %		
Object root		☒ Yes ☐ No		

変更を適用するには資産を保存します。

これで、[収集プロセス](#) でこの資産のサンプリングをリクエストできるようになりました。

資産分類

- [概要](#)
- [手動分類](#)
- [分類コンソールの使用](#)
 - [データエレメントを検索して選択](#)

- 。 自動分類の実行
- 。 資産のロック

概要

資産をデータエレメントと一致させることができます。これは、ビジネスオブジェクトが技術オブジェクトにどのようにリンクされているかを説明する主な機能です。

このリンクを独自に定義することは可能ですが、多くの資産では時間がかかる可能性があります。分類機能は、資産 (技術) とデータエレメント (ビジネス) の一致を見つけるのに役立ちます。

手動分類

ビューで、左上隅にある編集アイコンをクリックします。

The screenshot shows a configuration window for an asset named 'name_basics'. The window has a toolbar at the top with a red square icon highlighted. The main area contains several fields:

- DESCRIPTION:** A large text area containing the symbol $\emptyset$.
- NAME:** A field containing 'name_basics'.
- ASSET TYPE:** A field containing 'DB TABLE'.
- PARENT:** A field containing 'PUBLIC'.
- DATA ELEMENT:** A field containing the symbol $\emptyset$.
- INSTANCE:** A field containing 'IMDB'.
- DISABLED:** A field containing 'false'.
- MANDATORY:** A field containing the symbol $\emptyset$.

次に、データエレメントリストからデータエレメントを選択します。

The screenshot shows the 'Main' tab of the TIBCO EBX configuration interface. The 'Data element' field is highlighted with a red box and contains '[not defined]'. Below it, there are radio buttons for 'Data element automatically assigned' (set to 'No') and 'Is disabled?' (set to 'No'). The 'Mandatory' field has radio buttons for 'Yes' and 'No'. A large text area for 'Comment' is at the bottom.

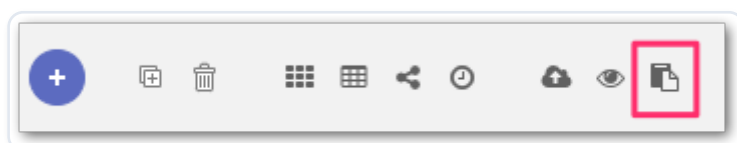
データエレメント名を入力して検索できます。更新を保存することを忘れないでください。

分類コンソールの使用

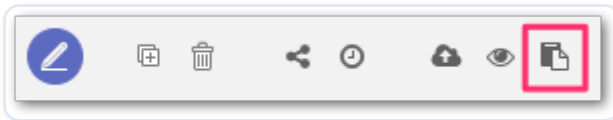
自分でデータエレメントを検索して選択する代わりに、**分類コンソール** を使用して、現在の資産にリンクする適切なデータエレメントに関する提案を表示できます。

このオプションは事前定義されたアルゴリズムを使用しており、効率を高めるにはデータが必要です。

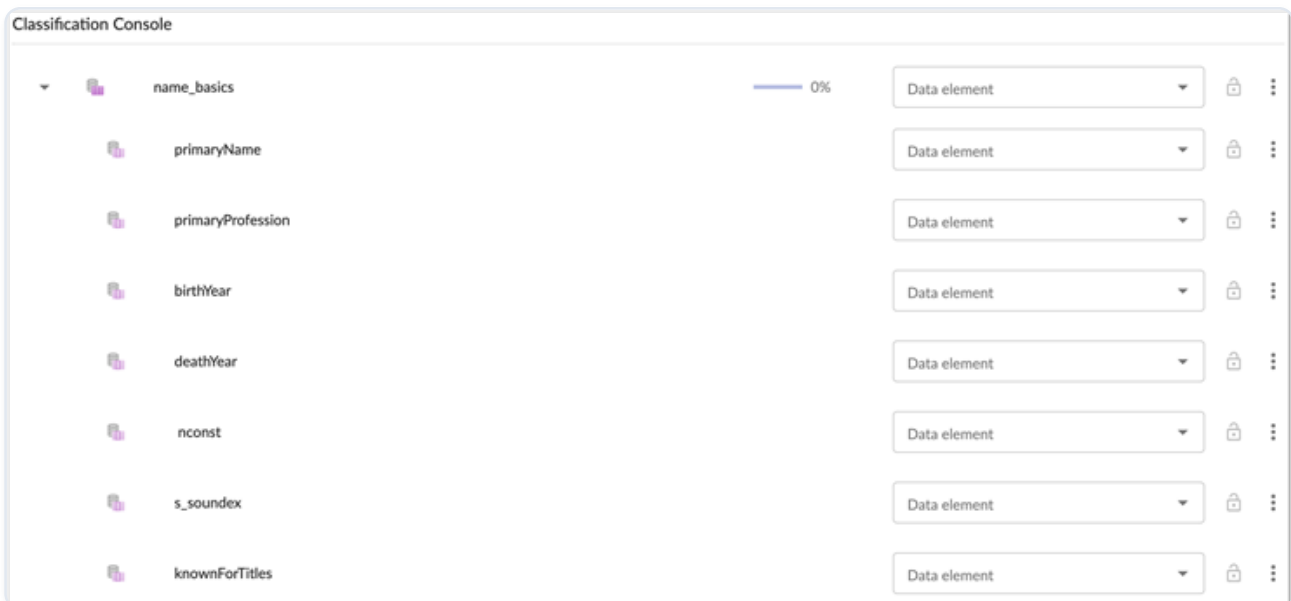
- 階層ビューから、分類する資産を 1 つ以上選択し、アイコンを選択します。



- 単一資産ビューから、分類アイコンを選択します。



新しい画面が表示され、現在選択されている資産とその子 (存在する場合) が表示されます。

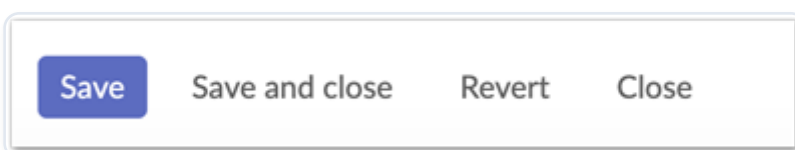


この画面では次のことができます。

1. 階層を展開または折りたたみます (子を持つ資産でのみ使用可能)。
2. 詳細を表示するには、資産名をクリックします。
3. 資産分類比率を確認します (資産とその子が分類されている場合)。
4. 資産にリンクするデータエレメントを検索して選択します。
5. 自動分類のためにデータエレメントをロックまたはロック解除します。
6. 自動分類の実行などの追加オプションにアクセスします。

下部のバーでは次のことができます。

- 現在の変更を保存します。
- 保存して分類コンソールを閉じます。
- 現在の変更を元に戻します。
- 分類コンソールを閉じます (現在保存されていない変更がある場合は確認ポップアップが表示されます)。



データエレメントを検索して選択

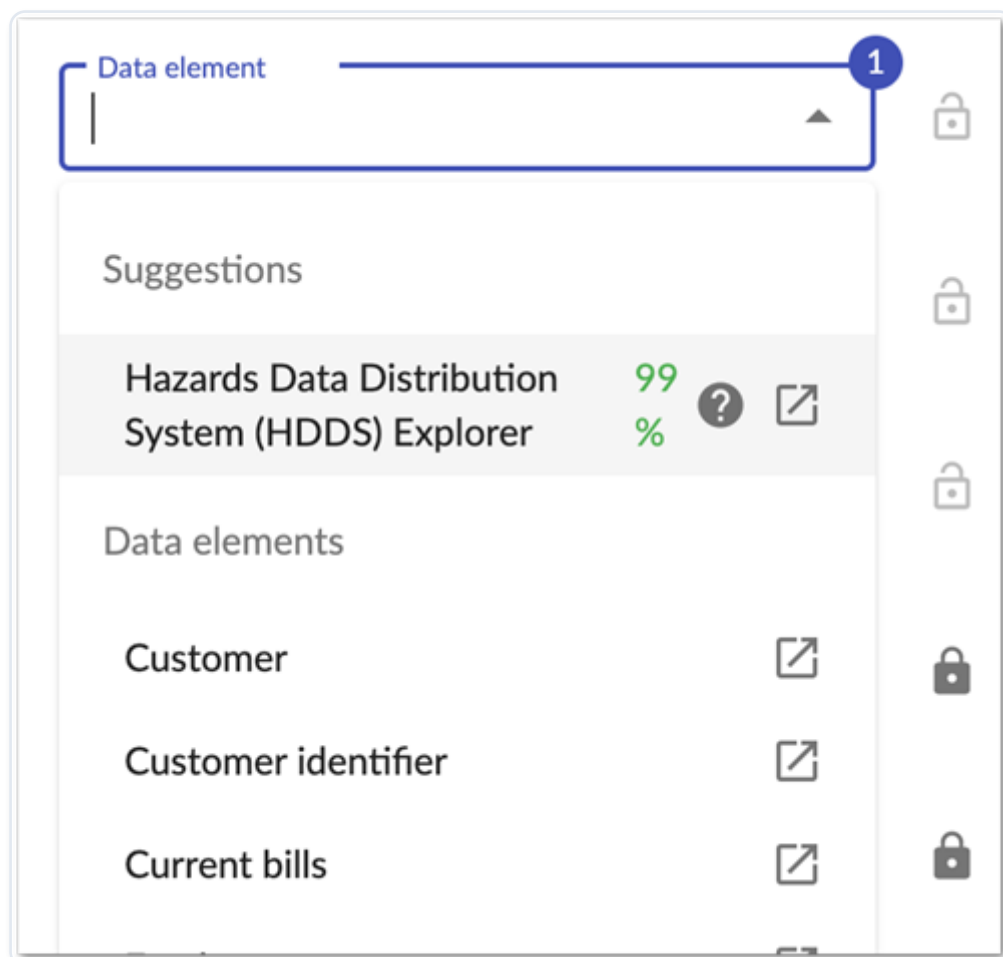
表示するデータエレメントボックスを選択します。

- データエレメントの提案リスト (存在する場合)。
- データエレメントリスト。

テキストを入力してデータエレメントを検索できます。 アイコンをクリックすると、データエレメントの詳細が表示されます。

The screenshot shows a user interface for searching data elements. At the top, there is a search input field with the placeholder text "Data element". Below this field, a dropdown menu is open, displaying a list of data elements. The list is titled "Data elements". The items in the list are: "Customer", "Customer identifier", "Current bills", "Employee", "Prototype", "Employee ID", and "Product ID". Each item has a magnifying glass icon to its right, indicating that clicking on the item will show more details. The "Customer" item is currently selected and highlighted. At the bottom of the dropdown menu, there is a label "Data element" and a downward-pointing arrow.

データエレメントの選択ボックスに提案が含まれている場合は、その近くに通知が表示されます。ボックスをクリックすると、提案とすべてのデータエレメントが表示されます。



提案には、一致した結果がパーセンテージで表示され、使用された分類構成に関する情報も表示されます。

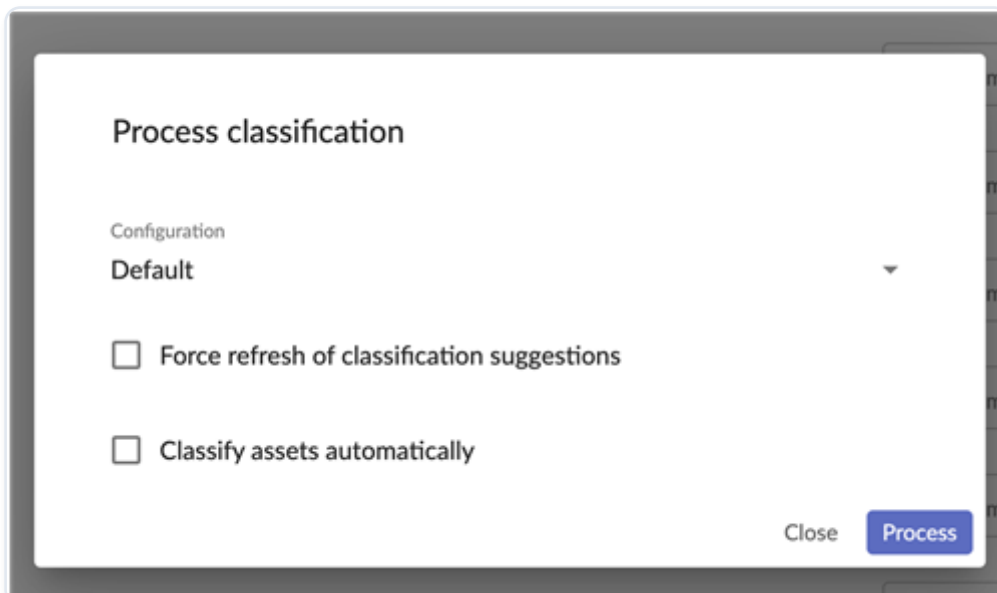
変更を行った場合は、データエレメント選択ボックスの左側に青い感嘆符が表示されます。

マウスポインタをその上に置くと、現在の変更の詳細 (データエレメントの変更、資産のロックまたはロック解除) が表示されます。

自動分類の実行

追加オプションにアクセスするには、オプションの右側にある縦方向のメニューボタンをクリックし、を選択します。ポップアップが表示され、次の操作を実行できます。

- 分類構成を選択します。
- 分類の提案を強制的に更新します。
- 資産を自動的に分類します (これにより、最も一致するデータエレメントが選択されます。保存されていない変更がある場合は、処理前に変更を保存しないと変更が失われる可能性があることを知らせる警告メッセージが表示されます)。



分類を開始するには、アイコンをクリックします。分類の実行中は、現在の資産とその子に対して新しい分類を実行することはできません。

分類コンソールを終了した後は、必ず変更を保存します。

資産のロック

資産にリンクされた既存のデータエレメントがある場合は、データエレメント選択ボックスの右側にあるロックボタンをクリックしてロックできます。オプションを使用して分類プロセスを実行する場合、ロックされた資産は無視されます。データエレメントのないロック解除された資産はロックできません。

リネージ

- [概要](#)
- [リネージの表示](#)
 - [資産ビューから](#)
 - [インスタンスビューから](#)
- [リネージの参照](#)
 - [概要ツールボックス](#)
 - [ノードの展開/折りたたみ](#)
 - [ノードリンクの表示](#)
 - [ノードの完全なリネージの表示](#)
 - [ノードの移動](#)

概要

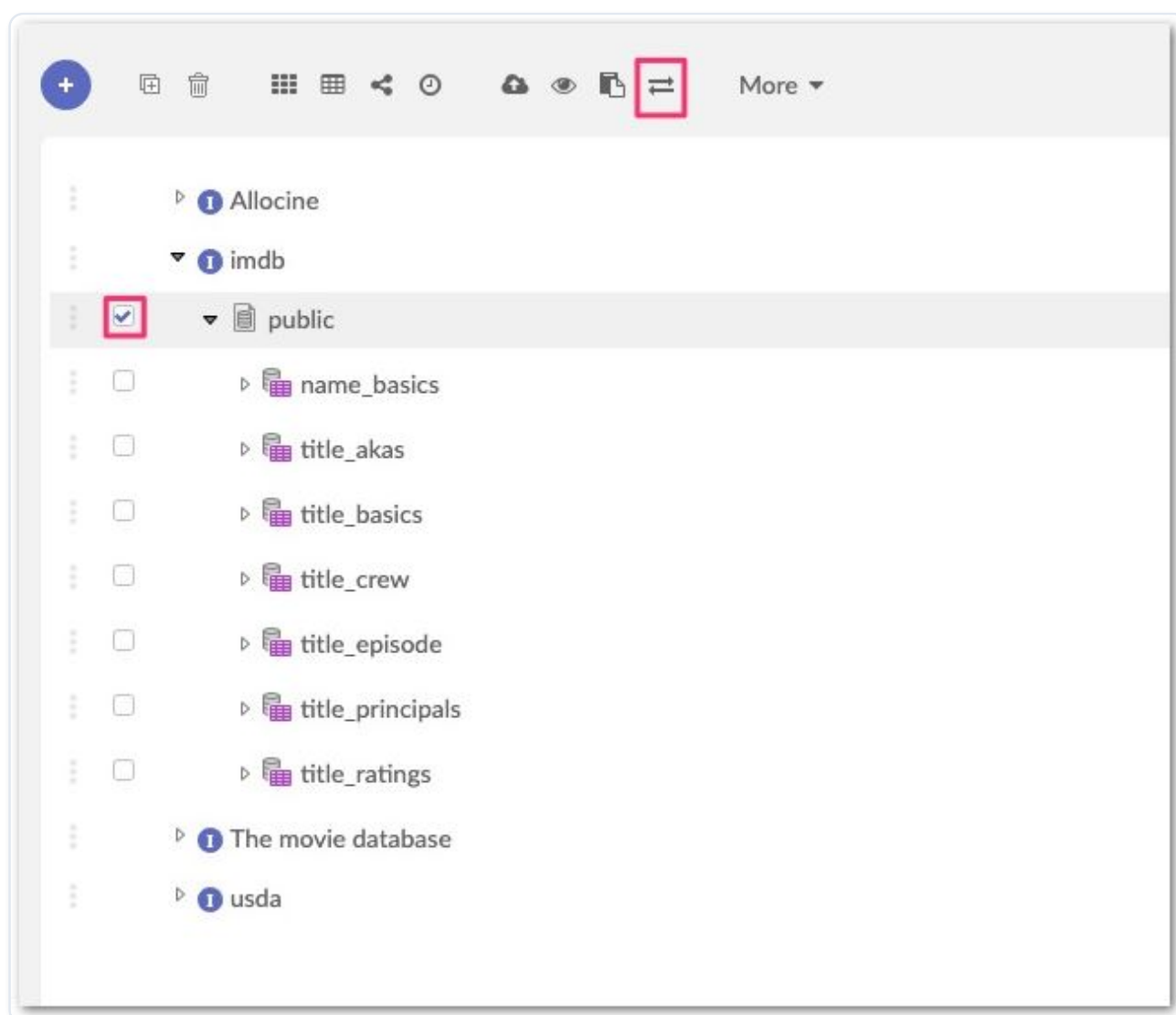
資産またはインスタンスのリネージを表示できます。これは、資産またはインスタスがどのように構成され、他の資産やインスタンスにリンクされているかを説明する主な機能です。

リネージの表示

資産ビュー

階層ビューまたはテーブルビューで資産を選択して、リネージを開始できます。ビューの上部にあるアイコンをクリックするだけです。

階層ビューから:



テーブルビューから:

	Label	Description	Asset type	Parent asset
☒	public		DB Schema	
☐	name_basics		DB Table	public
☐	primaryName		DB Column	name_basics
☐	primaryProfession		DB Column	name_basics

資産ビューフォームの読み取り中にリネージを開始することもできます。資産ビューフォームの上部にある アイコンをクリックします。

public

DESCRIPTION
Ø

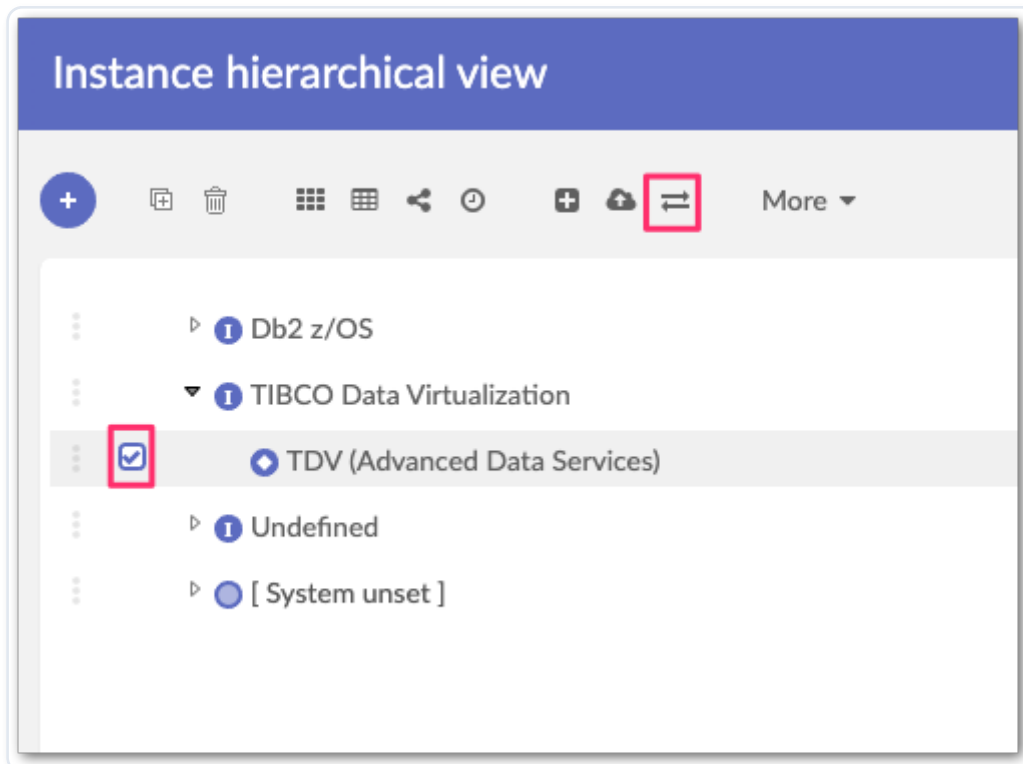
NAME
public

ASSET TYPE
DB SCHEMA

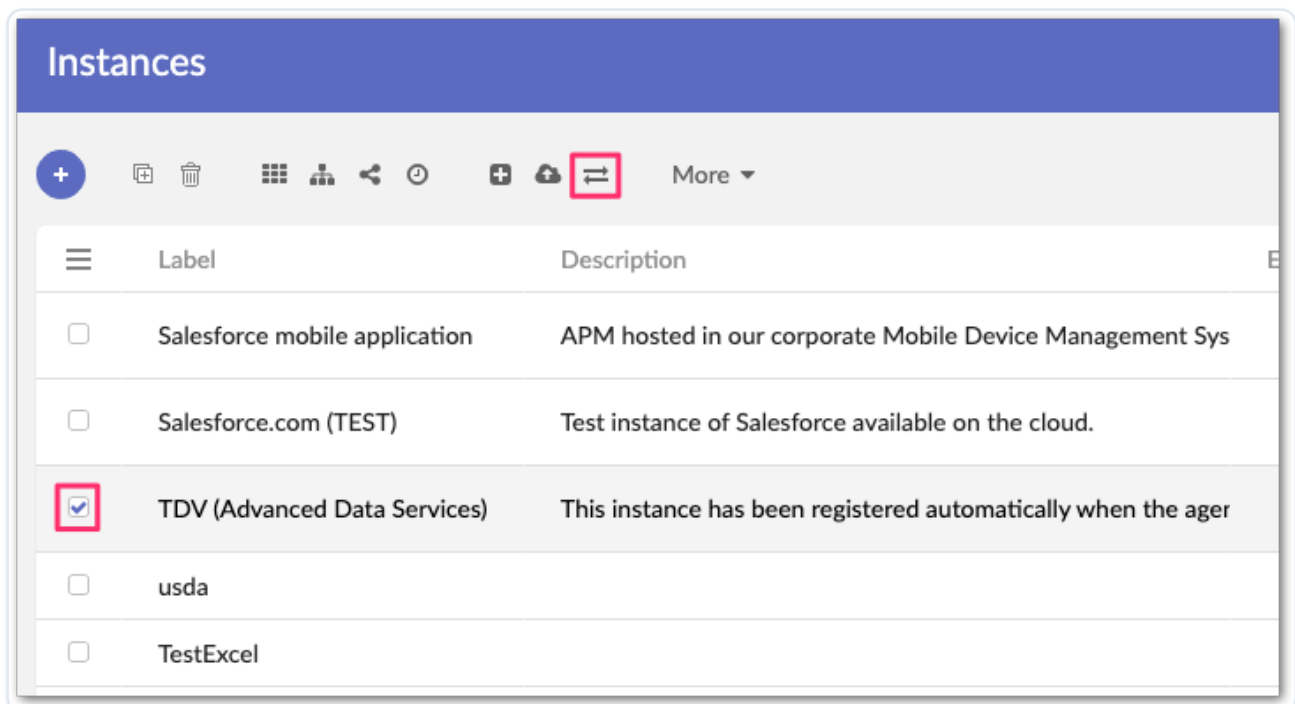
インスタンスビュー

階層ビューまたはテーブルビューでインスタンスを選択して、リネージを開始できます。ビューの上部にある アイコンをクリックするだけです。

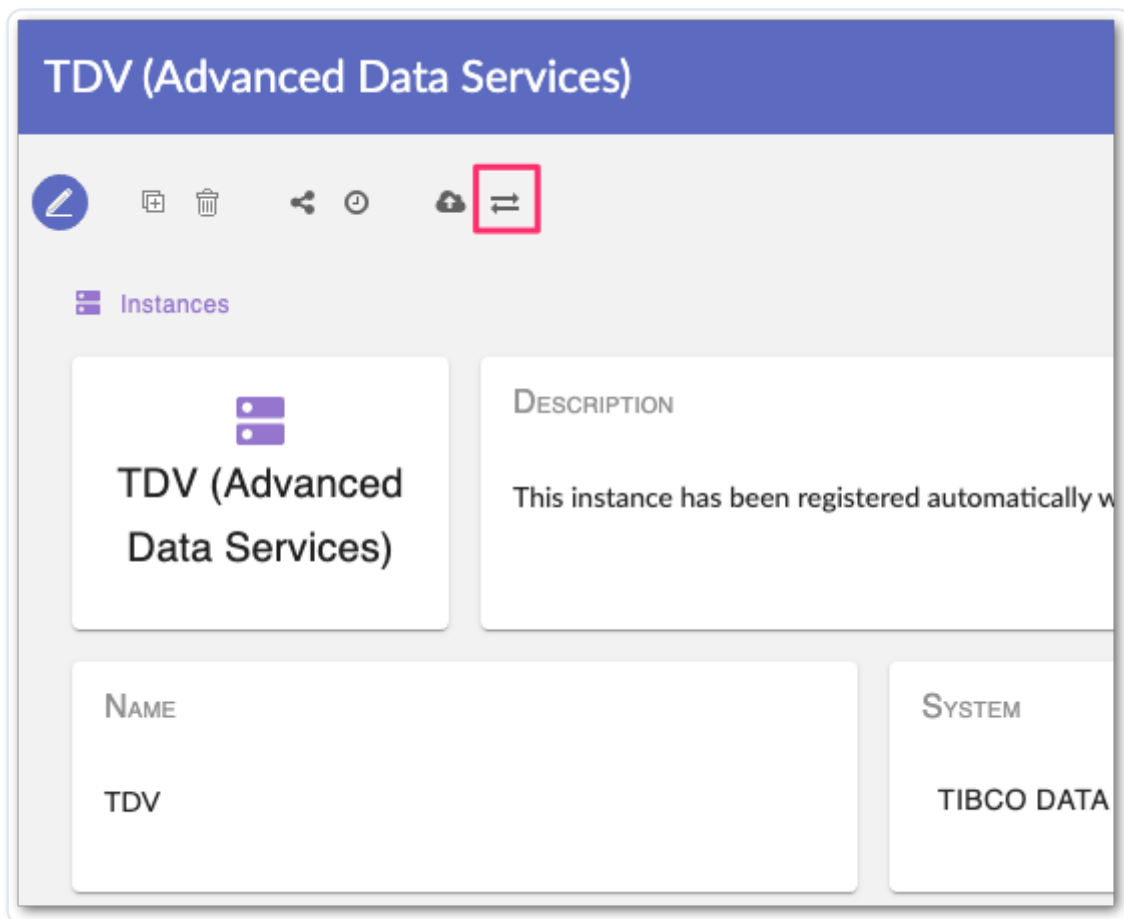
階層ビューから:



テーブルビューから:



インスタンスビューフォームを読み込んでいるときにリネージを開始することもできます。インスタンスビューフォームの上部にある アイコンをクリックします。



リネージの参照

サービスを開始すると、次の画面が表示されます。

- 以前に選択した資産またはインスタンス：選択したものが資産である場合、そのインスタンスとともに表示されます。
- インスタンスまたは資産の直接の子。
- アクションボタン。
- リネージグラフの参照に役立つ概要ツールボックス。

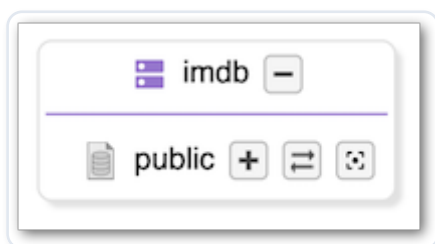


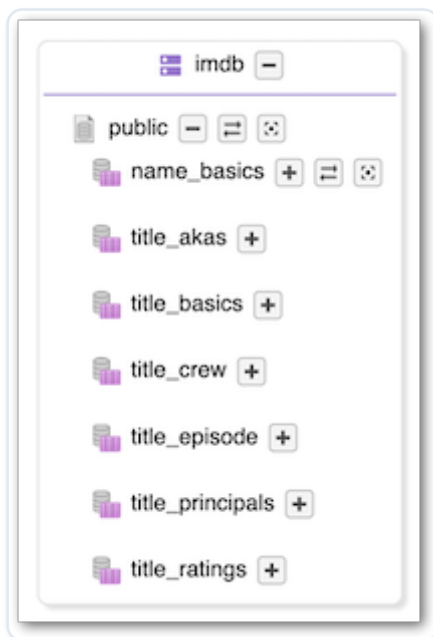
概要ツールボックス

概要ツールボックスは、複数のノードとリンクがあるためにリネージグラフが大きい場合に、リネージグラフをスクロールするのに役立ちます。概要ボックスは、リネージビュー内の任意の場所にドラッグできます。

ノードの展開/折りたたみ

ノード名の近くにある ボタンをクリックすると、ノードが展開され、その子が表示されます。ボタンは、子が存在する場合にのみ使用できます。





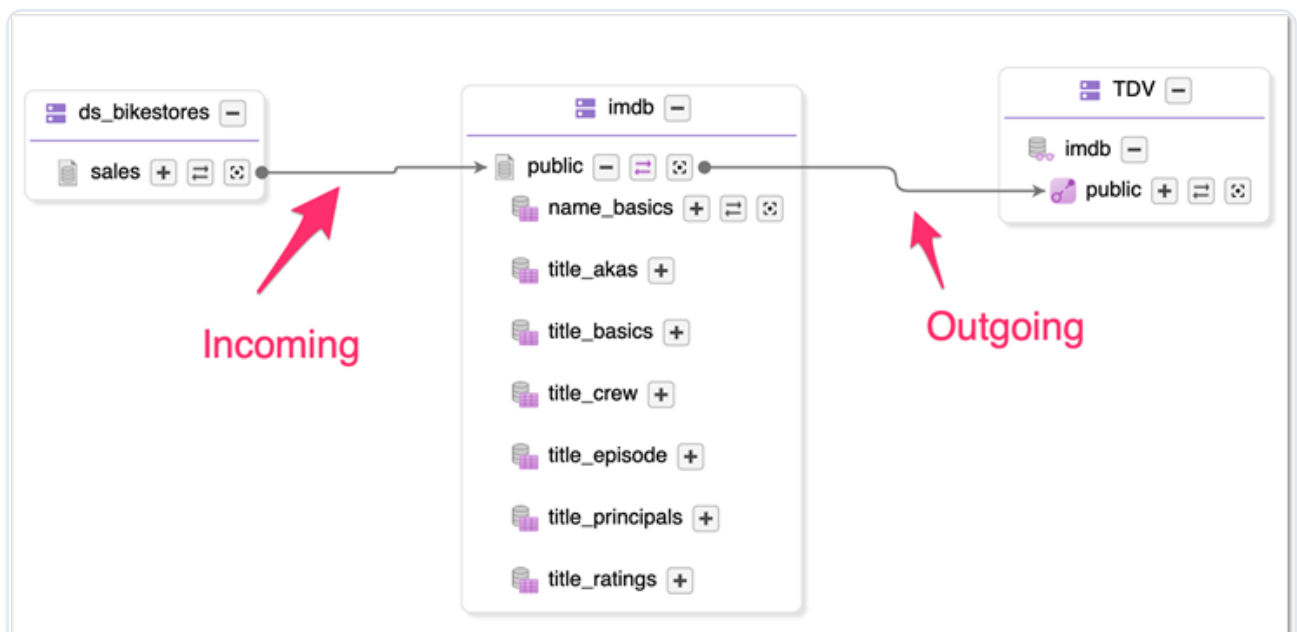
ノードが展開されているときは、ノード名の近くにある ボタンをクリックすると折りたたむことができます。

ノードリンクの表示

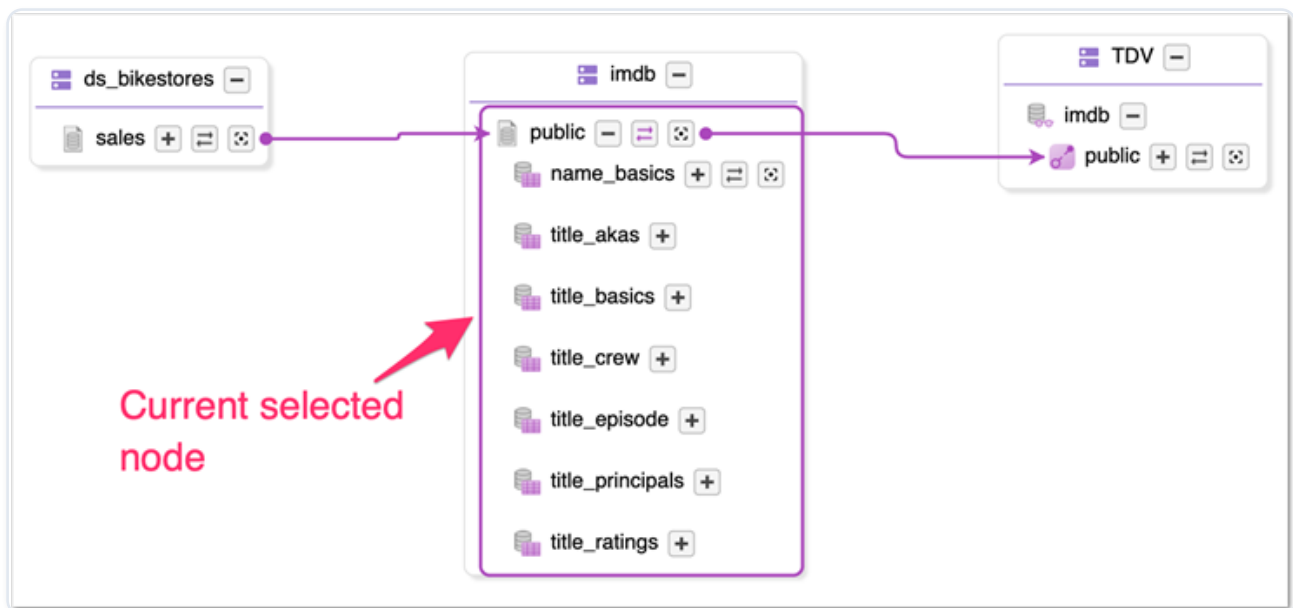
ノードに表示するリンクが含まれている場合は、ノード名の近くに アイコン (2 つの矢印が付いたアイコン) が表示されます。このボタンをクリックすると、次のものが表示されます。

- ノードの着信リンク
- ノードの発信リンク

ノードのすべてのリンクが表示されている場合は、 アイコンが強調表示されます。



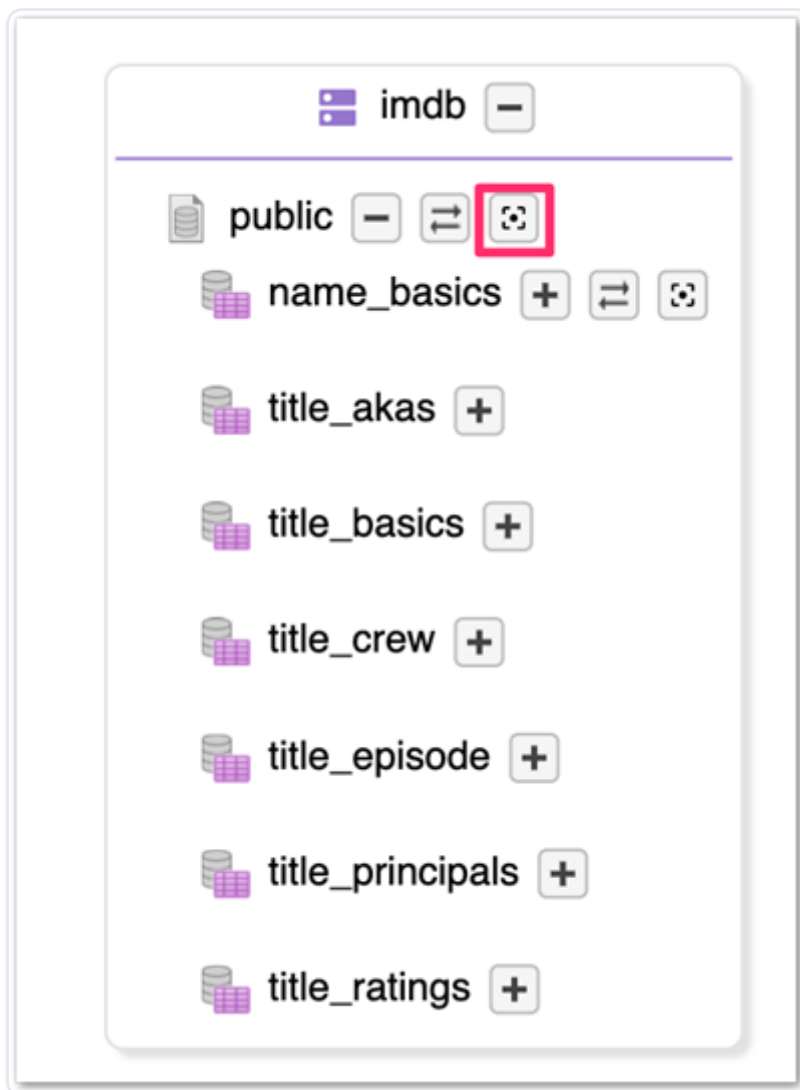
リネージが複雑すぎてリンクが多すぎる場合は、ノードをクリックしてそのリンクを強調表示することができます。



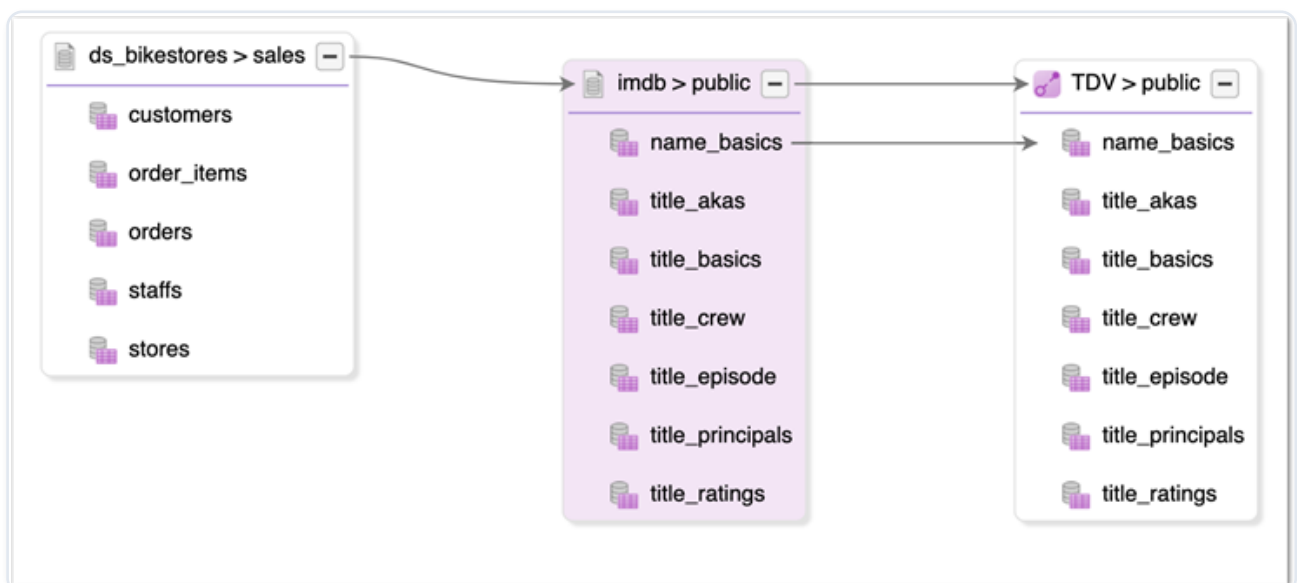
リンクの強調表示を削除するには、リネージグラフ上の他の場所をクリックします。

ノードの完全リネージの表示

最初のリネージビューでは、リネージを段階的に参照できます (各リンクボタンを手動でクリックします)。このグラフは、**参照可能なリネージ**と呼ばれます。選択したノードのリネージ全体を表示するには、ノードの近くにあるアイコンをクリックします。このボタンは、ノードにリンクが含まれている場合にのみ使用できます。



このアイコンをクリックすると、新しいリネージグラフビューが表示されます。このグラフの名前は 資産リネージです。このビューには、現在選択されているノードとその子ノードが含まれます。また、ノード、子ノード、およびその他のノードのすべてのリンクが再帰的に表示されます。



アイコンをクリックしたノードは、紫色の背景で表示されます。

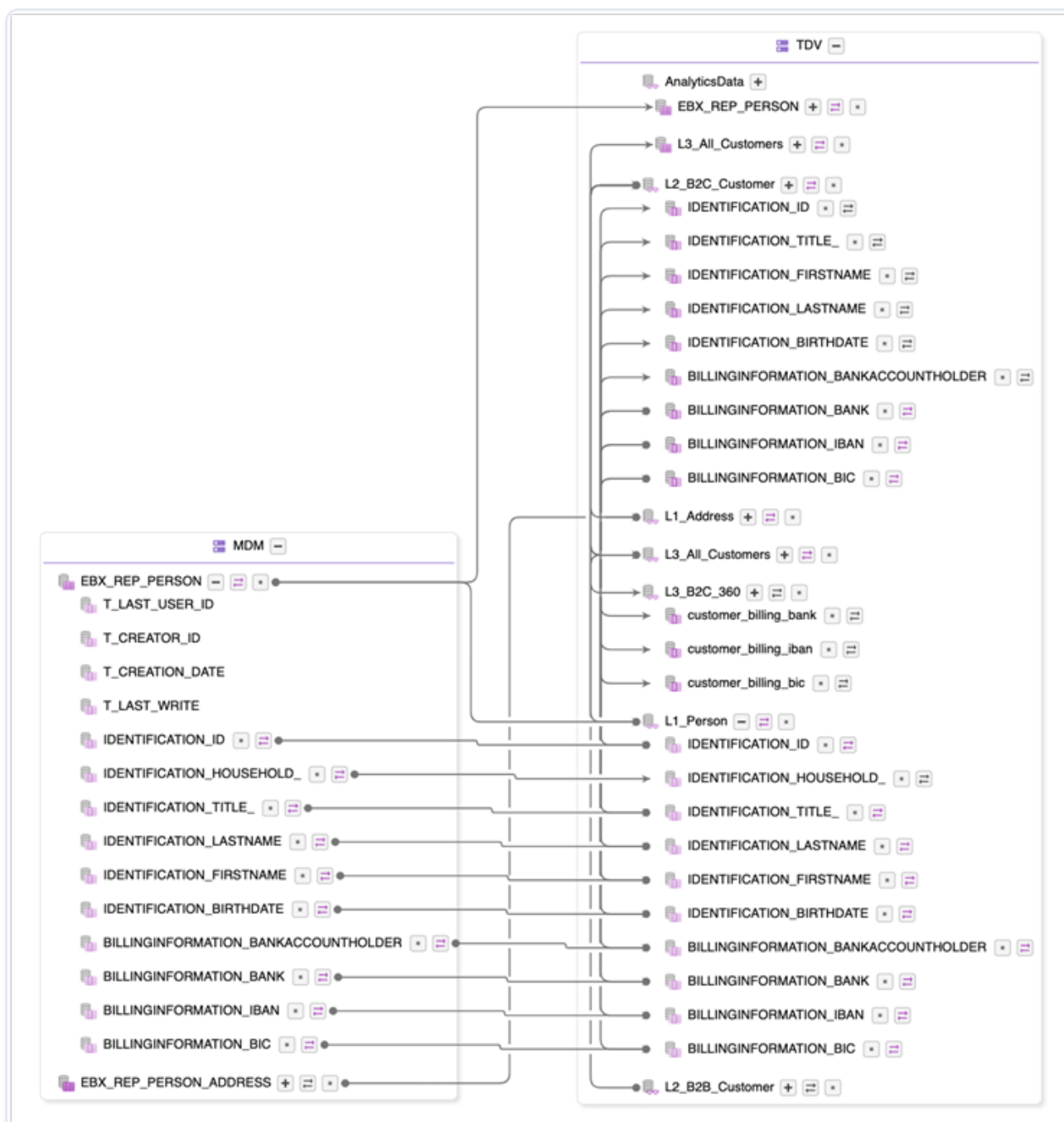
このグラフでは、各ノードのヘッドタイトルにインスタンス名が含まれています。テンプレートは > です。

リンクが存在する場合は、ノードをクリックするとリンクが強調表示されます。

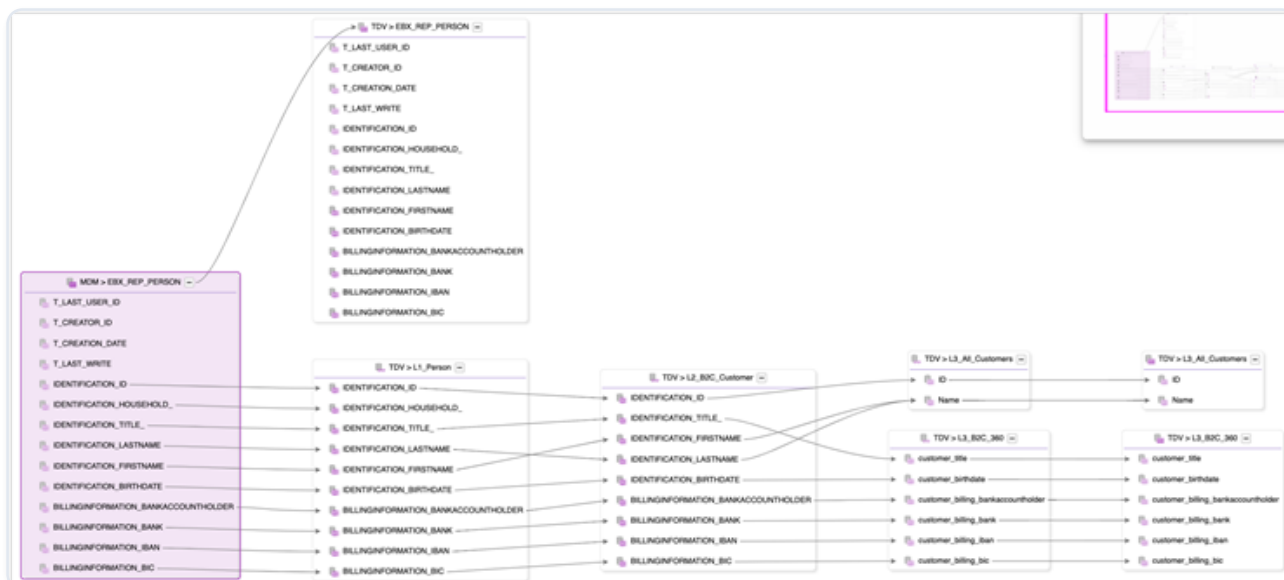


このグラフビューは、デフォルトの資産階層ビューではなく、展開バージョン (読みやすくするため) が表示されるため、前のものとは異なります。

以下は、参照可能なリネージグラフと資産リネージグラフを含む複雑なリネージビューの例です。この例では、EBX\REP\PERSON ノードからのすべてのリンクとその子リンクが再帰的に表示されます。

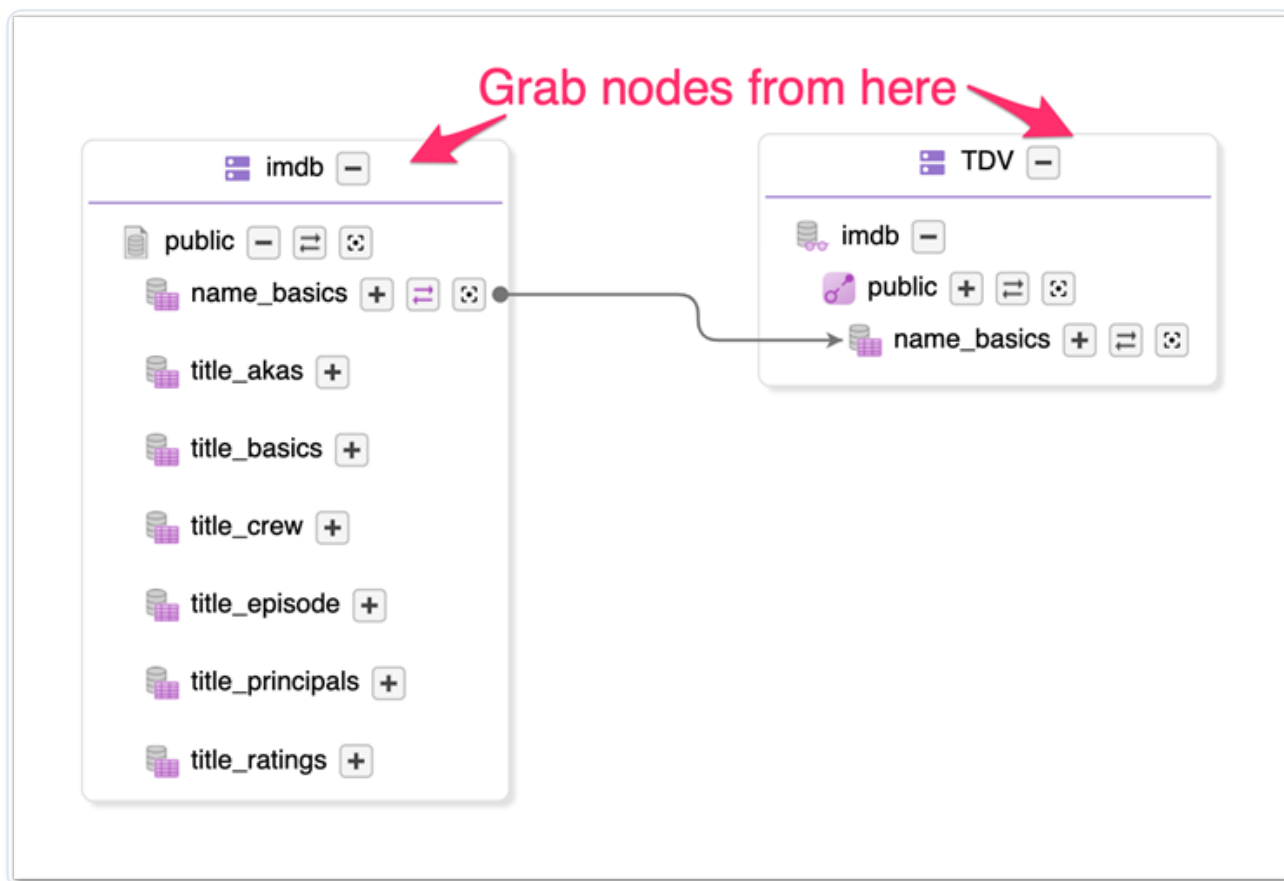


表示されているとおり、各資産ノードがインスタンスノードにグループ化されているため、完全なリネージを読み取るのは困難です。一方、以下のスクリーンショットでは、各資産ノードが分離されており、ノードグループのタイトルにインスタンス名が表示されています。



ノードの移動

マウスポインタでノードをドラッグして移動できます。移動できるのは ノードのみです。



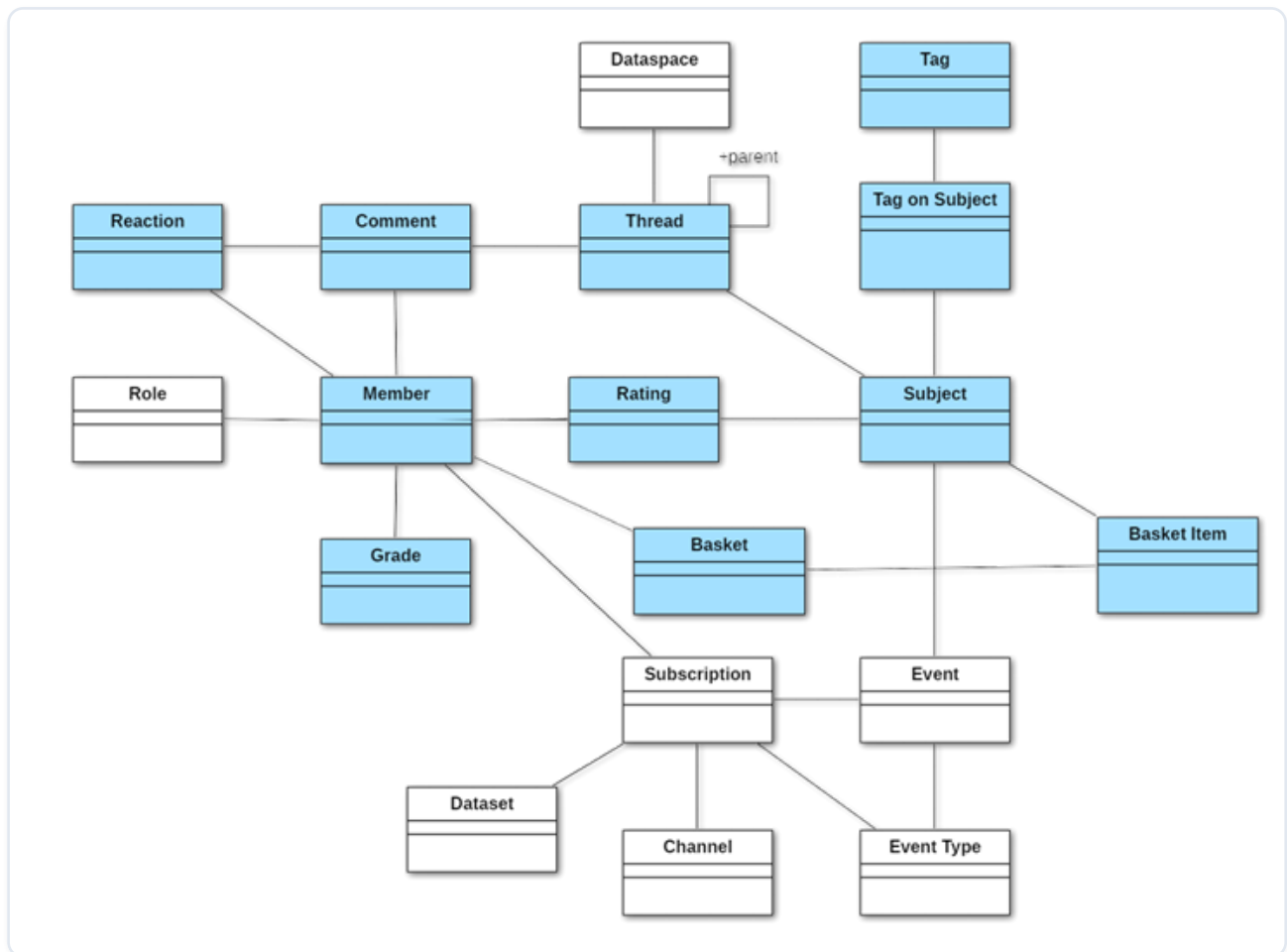
コラボレーション

- [コラボレーションの概要](#)
- [メタデータコメント](#)

- メタデータ評価
- メタデータのタグ付け
- コラボレーションワークフローによるメタデータの管理

コラボレーションの概要

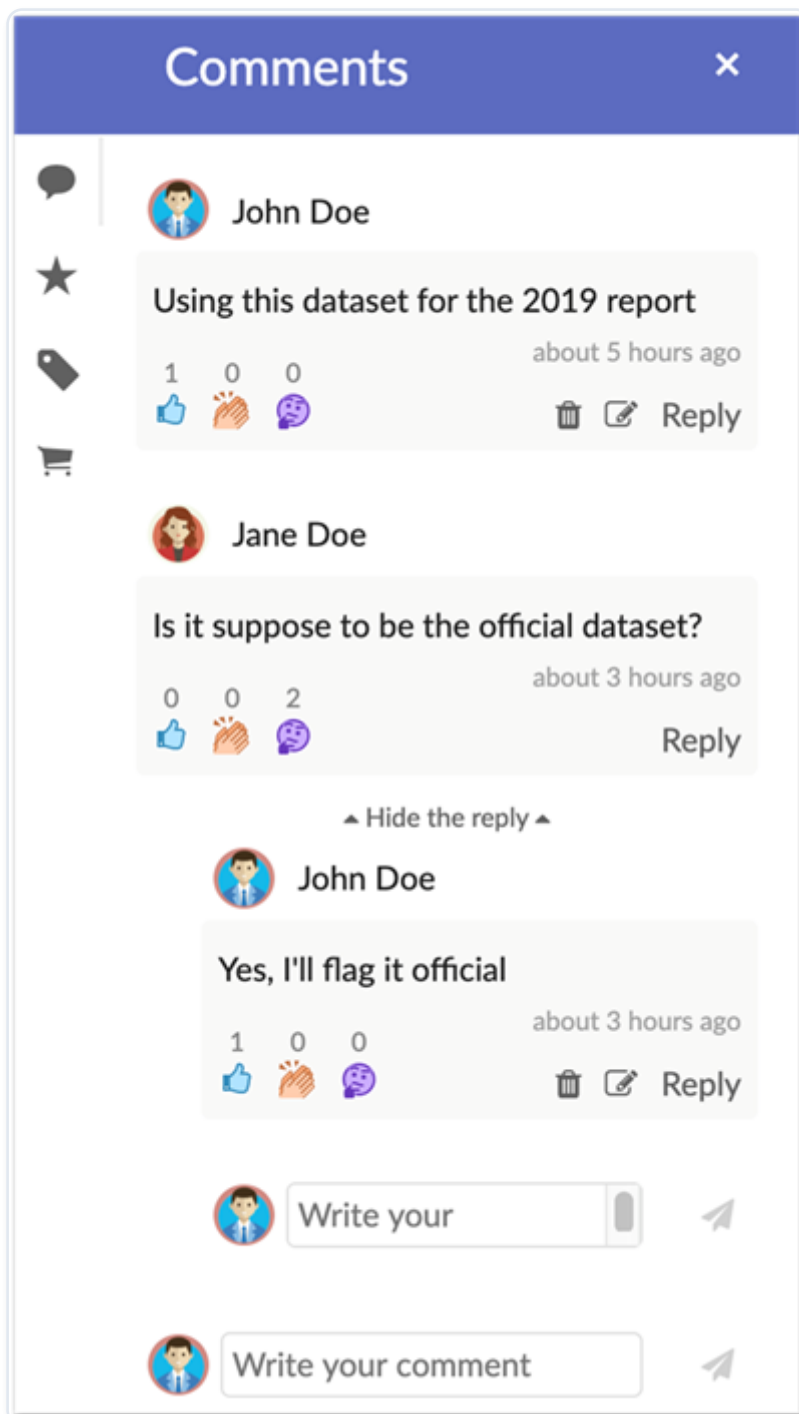
このセクションでは、タグ、コメント、評価、およびこれらの機能に関連するワークフローなどのメタデータコラボレーション機能について説明します。



メタデータコメント

ユーザーはレコードにコメントすることができ、これらのコメントは、ユーザーがメタデータをよりよく理解するのに役立つディスカッションに発展する可能性があります。また、コメントを使用して特定の情報をリクエストすることもできます。

コメント機能には、レコードを表示しているときに画面の右側にあるサイドバーからアクセスできます。



コメントは時系列で、最も古いコメントが最初に表示されます。コメントに対して次のアクションを実行できます。

- 特定のレコードに投稿されたコメントを読み取ります
- 特定のレコードにコメントを書き込みます。コメントの最大長は現在 255 文字です。自分のコメントを編集または削除することもできます
- 自分のコメントも含め、あらゆるコメントに返信できます。返信できるのは 1 レベルのみです。サブ返信スレッドを作成することはできません

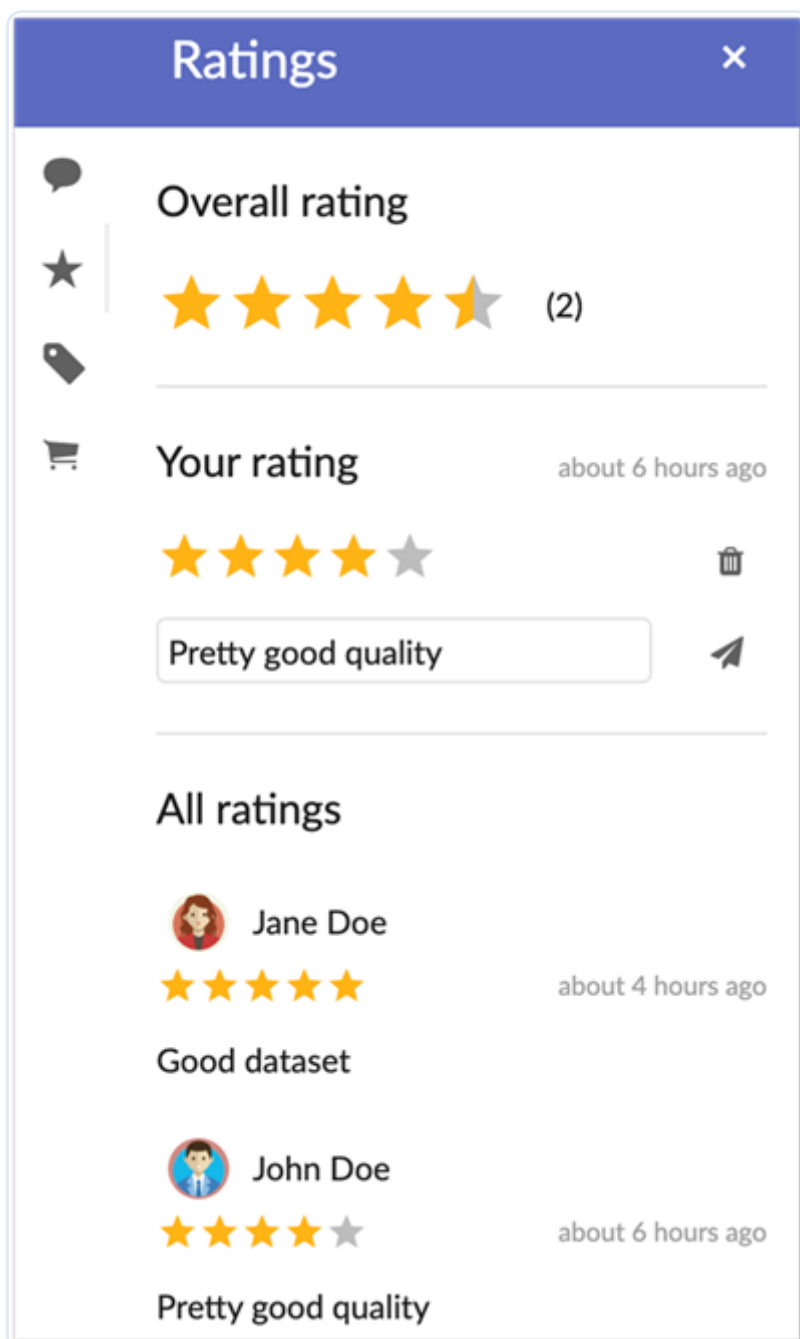
- コメントに反応します。反応 (同意、おめでとう、興味深い) は絵文字とラベルで表示されます。反応はコメントの下に表示されます。特定の反応を適用したユーザーの数が表示されます。現在、ユーザーごとに許可される反応は 1 つだけです。ユーザーが別の反応を適用すると、以前の反応は削除されます。ユーザーはいつでも反応を削除できます

メタデータ評価

ユーザーはレコードに評価を残して、データの評価を表明できます。総合評価は、特定のレコードに対するすべての評価の平均です。メタデータを検索するユーザーが、メタデータが信頼できるかどうかをよりよく理解するのに役立ちます。

たとえば、ユーザーはデータセットの品質が良いと言えます。また、特定の属性は通常空であるため役に立たないとも言えます。

評価機能には、どのレコードからでも、画面の右側にあるサイドバーからアクセスできます。



すべてのユーザーからの評価のリストは、最新の更新された評価を先頭にして時系列で表示されます。評価に対して次のアクションを実行できます。

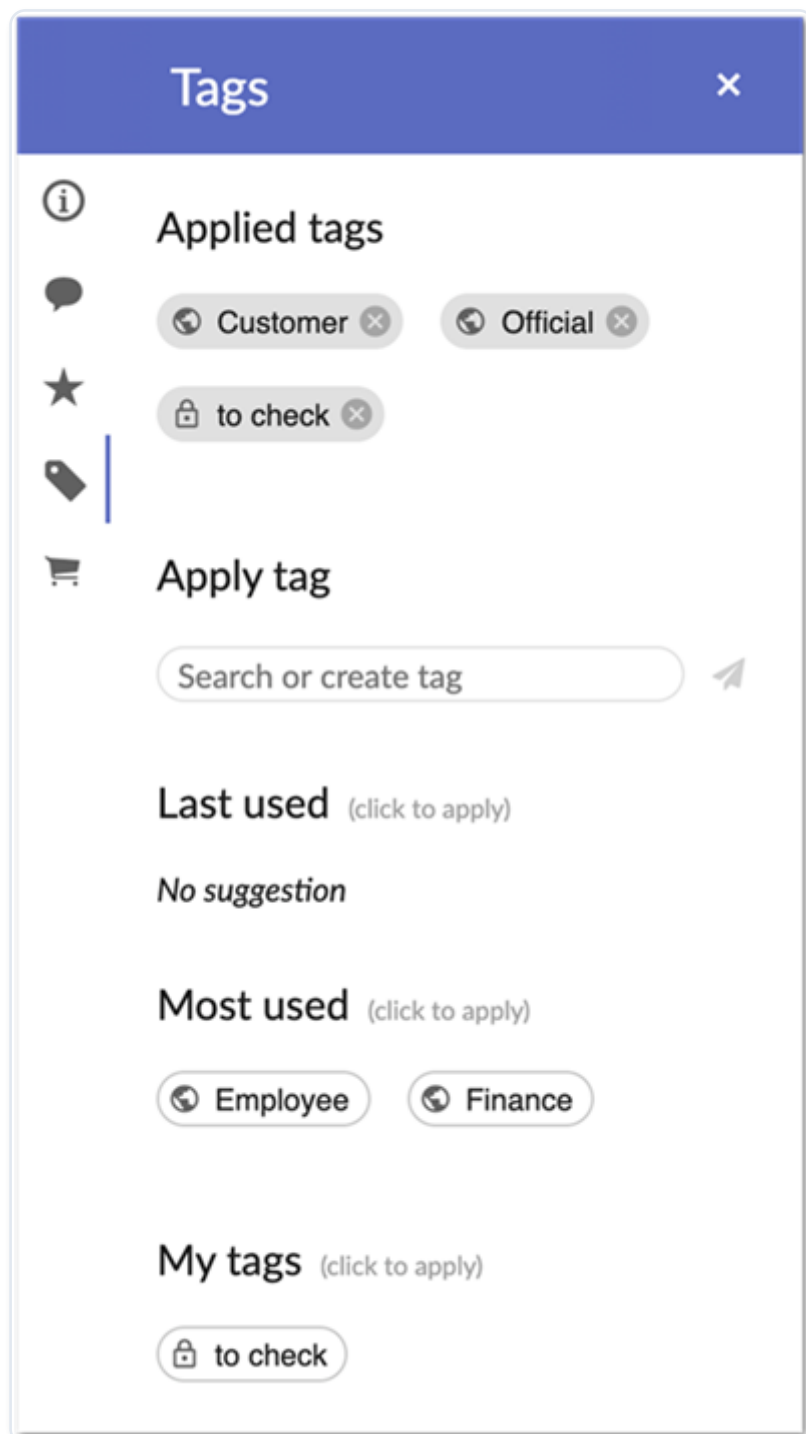
- 特定のレコードの全体的な評価を表示します。
- 特定のレコードに対するすべてのユーザーからの評価のリストを表示します。
- 自分のレコード評価を適用、変更、または削除します。ユーザーは 1 から 5 の値で評価できます。評価値の横に、理由のコメントを残すことができます。それに応じてレコードの全体的な評価が更新され、すべての評価のリストが更新されます。

メタデータのタグ付け

ユーザーはレコードにタグを付けることができ、1つのレコードに複数のタグを付けることができます。タグは、メタデータを定義する追加のコンテキスト情報をユーザーが提供するのに役立ちます。また、タグを使用してメタデータリストをフィルター処理することもできます。

注意：タグは単語またはフレーズ (単語のグループ) にすることができ、特殊文字を含めることもできます。

タグ機能には、どのレコードからでも、画面の右側にあるサイドバーからアクセスできます。



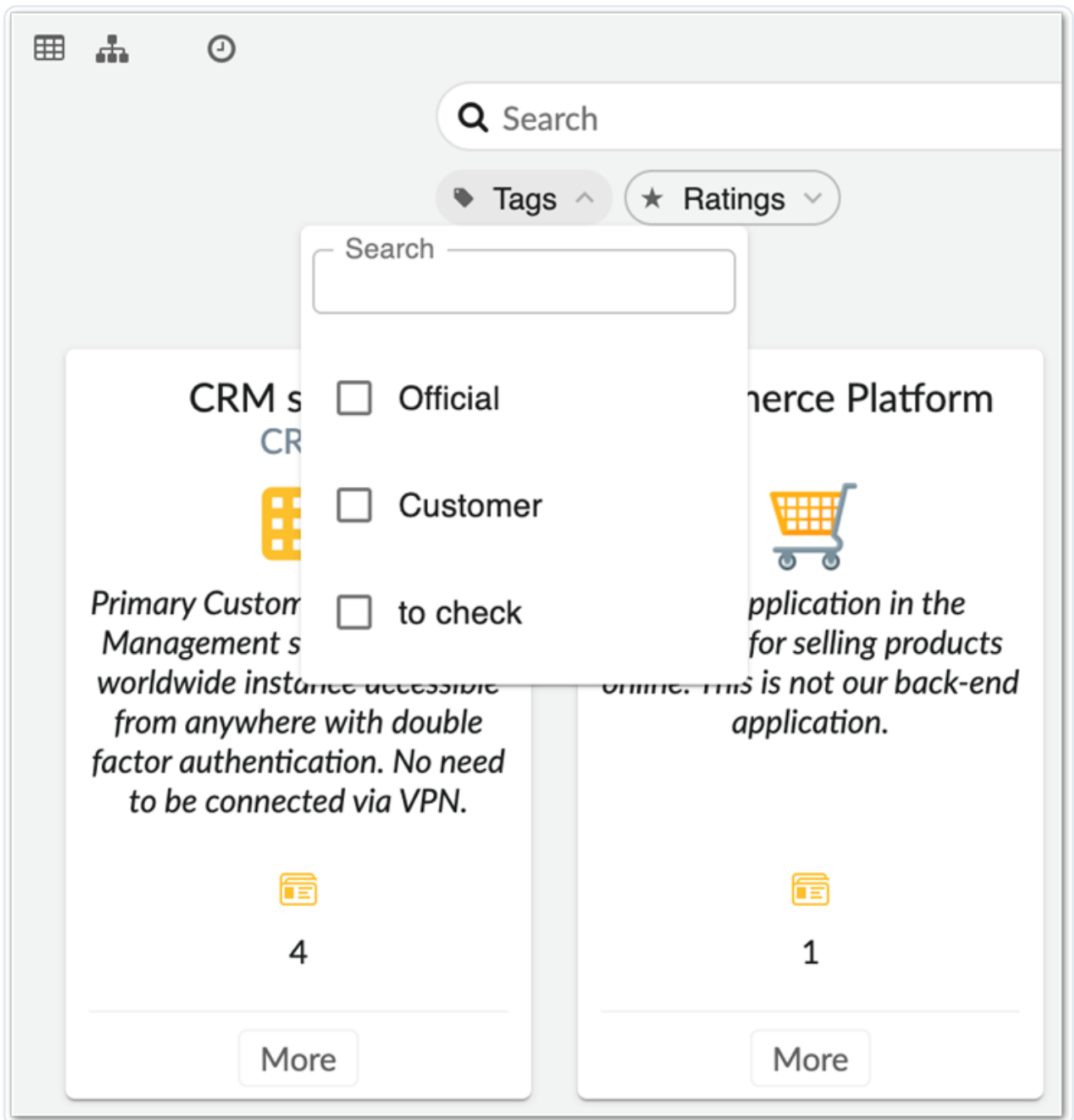
タグには、と の 2 種類があります。各種類は、ユーザーに対するタグの可視性に対応しています。タグはすべてのユーザーが表示および使用できますが、タグは、それを作成したユーザーのみに表示されます。左側のアイコンは可視性のタイプを示します。の場合はワールド、の場合は鍵です。

上部のセクションには、現在のレコードに適用されているタグが表示されます。タグの右側にある十字ボタンをクリックすると、適用されたタグを削除できます。

セクションでは、ユーザーは新しいタグを作成したり、既存のタグを検索してレコードに適用したりできます。タグの作成は、管理者が設定した権限によって制御されることに注意してください。すべてのユーザーがタグを作成できるわけではありません。タグの作成を許可されているユーザーは、送信ボタンをクリックした後に表示対象を選択できます。

セクションには、現在のユーザーが最後に使用した 10 個のタグが表示されます。セクションには、リポジトリ全体で最も頻繁に使用した 10 個のタグが表示されます。タグをクリックすると、現在のレコードに適用されます。このアクションはリアルタイムで実行されます。

最後に、セクションには、接続されたユーザーが作成したすべてのタグのリストが表示されます。



カードビューにはタグが統合されており、ユーザーは適用されているタグに基づいてレコードをフィルター処理できます。

コラボレーションワークフローによるメタデータの管理

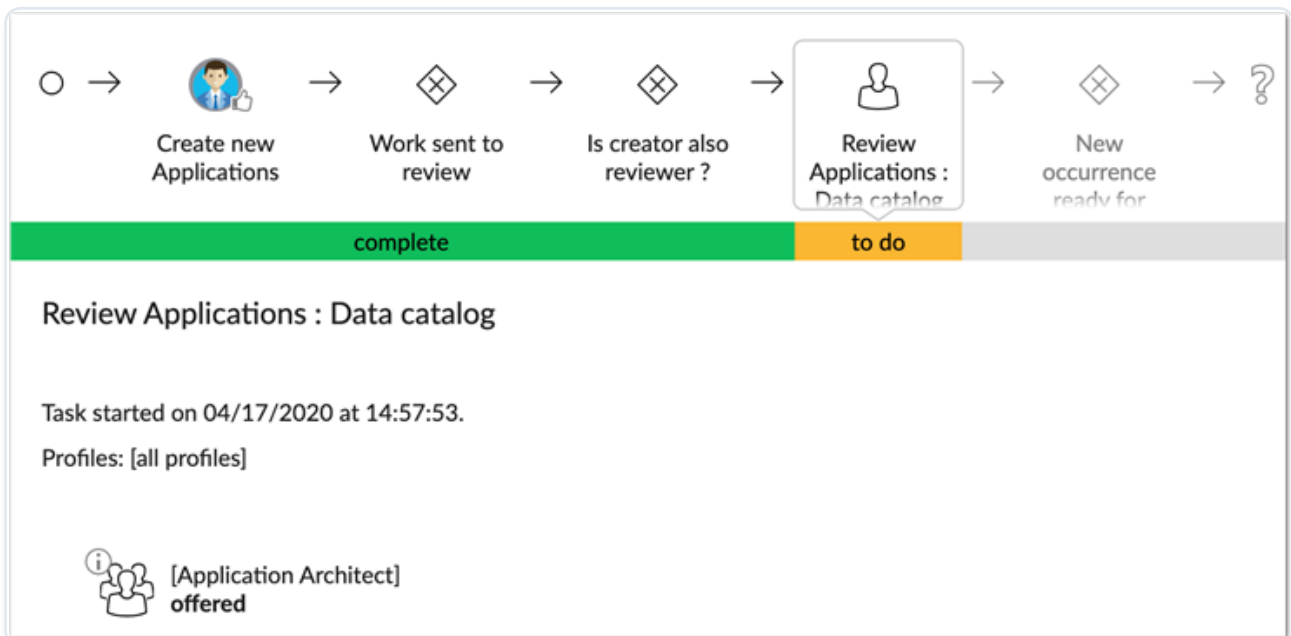
組織の管理者は、ユーザーがレコードを作成および更新するための権限を定義できます。管理者は、このような操作を実行するためにコラボレーションワークフローの使用を要求することができます。

作成ワークフローと更新ワークフローには、完了するまでにいくつかのステップがあります。

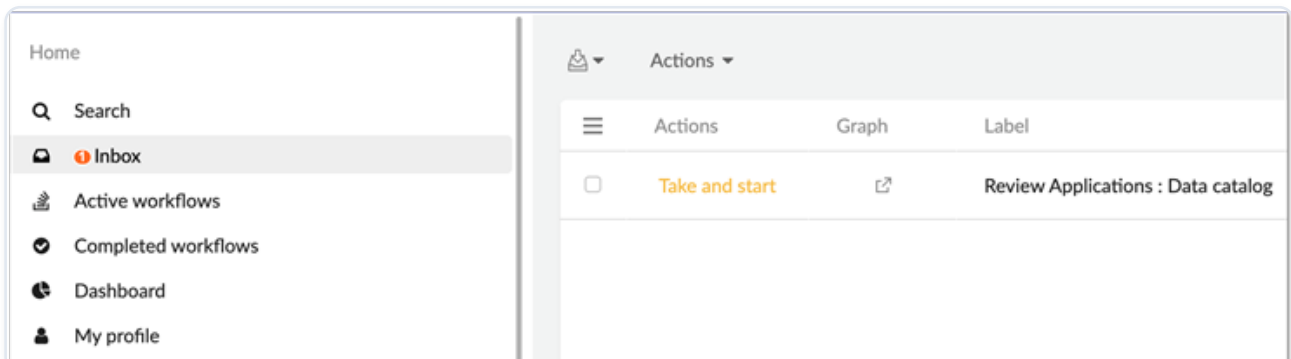
- 作成/更新
- レビュー

- 承認 (複数同時承認も可能)
- 確定

これらの手順は、組織および特定のエンティティに対して有効になっている場合と有効になっていない場合があります。詳細については、管理者にお問い合わせください。



ワークフローに関与している場合、タスクはこのユーザーと同じロールを持つ他のユーザーに提供され、に表示されます。を選択すると、タスクにアクセスできます。タスクを引き受けると、そのタスクはこのユーザーに非公開で割り当てられます。



タスク内で、上部に表示されるアクションはタスクの種類によって異なります。たとえば、などです。

Review Applications : Data catalog > Data catalog

Reject

Approve

Main

Datasets

Instances

Business processes

Consuming applications

Identifier

41

Label

*

Data catalog

Description

左側のナビゲーションメニューからアクティブなワークフローにアクセスできます。アクセスできるワークフローのみがリストされます。このアクティブなワークフローのリストから、ワークフローのグラフを表示して、実行されているステップを確認できます。

リファレンスマニュアル

- 統合
 - 統合と拡張の概要
 - TIBCO EBX® の Web コンポーネントとしての使用
 - ビルトインのユーザーサービス
 - サポートされている XPath 構文
- ローカリゼーション
 - ラベル付けとローカリゼーション
 - TIBCO EBX® ローカリゼーションの拡張
- 永続性
 - 永続性の概要
 - 履歴
 - レプリケーション
 - データモデルの進化
- その他
 - 継承と値の解決
 - 権限
 - 基準エディター
 - 検索

統合

- 統合と拡張の概要
- TIBCO EBX® の Web コンポーネントとしての使用
- ビルトインのユーザーサービス
- サポートされている XPath 構文

統合と拡張の概要

いくつかのサービスおよびコンポーネント API を使用すると、TIBCO EBX® のカスタム拡張機能を開発し、他のシステムと統合できます。

- ユーザーインターフェイスのカスタマイズと統合
- データサービス
- XML および CSV インポート/エクスポートサービス
- プログラムサービス

ユーザーインターフェイスのカスタマイズと統合

EBX® グラフィカルインターフェイスは、さまざまな EBX® API を通じてカスタマイズできます。

また、サポートされている Web ブラウザー からアクセスできるアプリケーションに統合することもできます。

詳細については、[インターフェイスのカスタマイズ](#) を参照してください。

データサービス

データサービスは、外部システムが次のいずれかを使用して EBX® と対話する手段を提供します。

- Web サービス記述言語 (WSDL/SOAP) 標準
- Representational state transfer (REST)

以下も参照してください。:

- [WSDL/SOAP データ サービス](#)
- [REST データサービス](#)

XML および CSV インポート/エクスポートサービス

EBX® には、XML および CSV 形式からデータをインポートおよびエクスポートするためのビルトインサービスが含まれています。XML および CSV のインポートとエクスポートは、次を使用して実行できます。

- ユーザーインターフェイス
- データサービス
- Java API

以下も参照してください。:

- XML のインポートとエクスポート
- CSV のインポートとエクスポート

プログラムサービス

プログラムサービスを使用すると、スケジュールされたタスクやバッチなど、明確に定義されたコンテキストで手順を実行できます。

プログラムサービスの例には次のようなものがあります。

- 外部ソースからのデータのインポート
- 複数のシステムへのデータのエクスポート
- 監視システムによって開始されたデータ履歴化
- データの最適化とリファクタリング

EBX® ビルトインの最適化サービス では不十分な場合

以下も参照してください。:

- `ProgrammaticService`

TIBCO EBX® の Web コンポーネントとしての使用

- 概要
- EBX® Web コンポーネントのアプリケーションへの統合
- リポジトリエレメントとスコープの選択
- 組み合わせた選択
 - 特定のケース

- 既知の制限事項
- リクエストの仕様
 - ベース URL
 - ユーザー認証とセッション情報のパラメーター
 - エンティティとサービスの選択パラメーター
 - レイアウトパラメーター
- EBX® Web コンポーネントへの呼び出しの例

概要

EBX® は、HTTP プロトコルを介して呼び出されるユーザーインターフェイス Web コンポーネントとして使用できます。EBX® Web コンポーネントは、サポートされている Web ブラウザーからアクセスできる任意のアプリケーションに統合できます。このアクセス方法は、ユーザー認証、データ検証、自動ユーザーインターフェイス生成など、EBX® の主な利点を提供すると同時に、リポジトリの特定の要素にユーザーナビゲーションを集中させる機能を提供します。

EBX® Web コンポーネントの一般的な使用法には、組織のイントラネットフレームワーク、またはユーザーへの特定のタスクの割り当てを管理するアプリケーションへの統合が含まれます。

以下も参照してください。:

- サポートされる Web ブラウザー

EBX® Web コンポーネントのアプリケーションへの統合

EBX® Web コンポーネントを呼び出す Web アプリケーションは次のとおりです。

1. Java 以外のアプリケーションで、最も基本的なのは静的 HTML ページです。

この場合、アプリケーションは、EBX® Web コンポーネントの [リクエストの仕様](#) に従う HTTP リクエストを送信する必要があります。

2. Java アプリケーションの例としては、次のものがあります。

- ターゲットとする EBX® リポジトリと同じアプリケーションサーバーインスタンスまたは別のアプリケーションサーバーインスタンスで実行されている Java Web アプリケーション。

- EBX® ユーザーサービスまたはカスタムウィジェット。この場合、新しいセッションは親 EBX® セッションから自動的に継承されます。

注意:

Java では、EBX® Webコンポーネントを呼び出す HTTP リクエストを作成するための推奨される方法は、API でクラス `UIHttpManagerComponent` を使用することです。

リポジトリエレメントとスコープの選択

EBX® Web コンポーネントが呼び出されると、ユーザーは最初に新しくインスタンス化された HTTP セッションで認証される必要があります。次に、Web コンポーネントはリポジトリエレメントを選択し、リクエストで定義された `scope` レイアウトパラメーターに従ってそれを表示します。

パラメーター `firstCallDisplay` は、その値に応じてこの自動表示を変更する場合があります。

選択できるリポジトリエレメントは次のとおりです。

- データスペースまたはスナップショット
- データセット
- ノード
- テーブルまたは公開されたビュー
- テーブルレコード

スコープは、ユーザーに表示されるユーザーインターフェイスの量を決定し、ユーザーがセッション内でナビゲートできる場所を定義します。Web コンポーネントが使用するデフォルトのスコープは、リクエストによって選択または呼び出されるエンティティまたはサービスに応じて、可能な限り最小になります。

以下も参照してください。:

- [Scope](#)

以下も参照してください。:

- [firstCallDisplay](#)

パースペクティブアクションだけでなく、特定のパースペクティブを選択することもできます。

デフォルトでは、スコープが「full」の場合、エレメントの選択はユーザーパースペクティブのコンテキストで行われます。

以下も参照してください。:

- [パースペクティブ](#)

組み合わせた選択

Web コンポーネントの URL は、パースペクティブとアクションまたはエンティティ (データスペース、データセットなど) を指定できます。したがって、URL でパースペクティブとエンティティを指定している (ただしアクションは指定していない) Web コンポーネントの場合、パースペクティブのアクションがこのエンティティと一致すると、このアクションが自動的に選択されます。

それ以外の場合、このエンティティに一致するアクションがない場合、アクションは選択されませんが、エンティティは関係なく開かれます。

エンティティと同時にアクションが指定されている場合、この最後は無視され、アクションが選択されます。

特定のケース

ターゲットエンティティがレコードであり、このレコードを含むテーブルにアクションがある場合、このアクションが選択され、アクション内でレコードが開かれます。

同様に、ワークフロー作業項目が Web コンポーネントの対象であり、パースペクティブに «受信トレイ» のアクションが存在する場合、このアクションが選択され、作業項目がその中で開かれます。

既知の制限事項

Web コンポーネントがテーブルをフィルタリングするための述語を指定する場合、パースペクティブアクションは、選択されるのとまったく同じ述語を指定する必要があります。

同様に、パースペクティブアクションがテーブルをフィルタリングするための述語を指定する場合、Web コンポーネントは、一致を確立するためにまったく同じ述語を指定する必要があります。

リクエストの仕様

ベース URL

デフォルトの展開では、ベース URL は次の形式にする必要があります。

```
http://[:]/ebx/
```

注意:

ベース URL は、Web アプリケーション `ebx.war` の展開記述子 `/WEB-INF/web.xml` に定義されているサーブレット `FrontServlet` を参照する必要があります。

ユーザー認証とセッション情報のパラメーター

パラメーター	説明	必須
<code>login</code> と <code>password</code> 、またはユーザーディレクトリ固有のトークン	ユーザー認証のプロパティを指定します。ログインとパスワードのペアもユーザーディレクトリ固有のトークンも提供されていない場合、ユーザーはリポジトリのログインページを介して認証する必要があります。詳細については、 <code>Directory</code> を参照してください。	いいえ
<code>trackingInfo</code>	新しいセッションの追跡情報を指定します。追跡情報は履歴テーブルに記録されます。	いいえ

パラメーター	説明	必須
	さらに、プログラムでアクセス許可を制限するために使用できます。詳細については、 <code>AccessRule</code> を参照してください。	
<code>redirect</code>	コンポーネントセッションの終了時にユーザーが ボタンをクリックしたときにリダイレクトされる URL。レコードを選択する場合は常に ボタンが表示されますが、それ以外の場合に表示するかどうかは、パラメーター <code>closeButton</code> を使用して指定する必要があります。詳細については、 終了ポリシー (非推奨) を参照してください。	いいえ
<code>locale</code>	使用するロケールを指定します。値は <code>en-US</code> または <code>fr-FR</code> のいずれかです。	いいえ。デフォルトはユーザーに登録されているロケールです。

エンティティとサービスの選択パラメーター

パラメーター	説明	必須
<code>branch</code>	指定されたデータスペースを選択します。	いいえ
<code>version</code>	指定したスナップショットを選択します。	いいえ
<code>instance</code>	指定されたデータセットを選択します。値は、選択したデータスペースまたはスナップショットに存在するデータセットの参照にする必要があります。	<code>xpath</code> または <code>viewPublication</code> が指定されている場合のみ。

パラメーター	説明	必須
<code>viewPublication</code>	選択したコンテンツに適用する表形式、階層、またはタイムビューのパブリケーション名を指定します。このパブリケーション名は、ビューの公開中に宣言された名前です。これは、> の下の 領域にあります。ビューのすべての設定、つまり、フィルター、並べ替え順序、および表示される列が結果に適用されます。このビューを適用するには、データスペースとデータセットを選択する必要があります。ビューの定義に基づいて自動的に決定できるため、ターゲットテーブルの選択は必要ありません。このパラメーターは、<code>xpath</code> パラメーターで論理「AND」演算として指定された述語と組み合わせることができます。	いいえ
<code>xpath</code>	データセット内のノード選択を指定します。値は、選択したデータセットにある有効な絶対パスにすることができます。表記は、構文が省略された簡略化された XPath に準拠している必要があります。また、他の Web コンポーネントパラメーター (たとえば、<code>viewPublication</code> や <code>workflowView</code>) を使ってテーブルを自動的に選択できる場合は、「」 (それぞれ %5B と %5D でエンコードされる) で囲まれた述語を使用することもできます。XPath 構文については、XPath でサポートされている構文を参照	いいえ

パラメーター	説明	必須
	してください。詳細については、 <code>UIHttpManagerComponent.setPredicate</code> を参照してください。	
<code>service</code>	アクセスするサービスを指定します。ビルトインユーザーサービスの詳細については、ビルトインサービスを参照してください。詳細については、Java API で <code>ServiceKey</code> を参照してください。	いいえ
<code>workflowView</code>	選択するワークフローセクションを指定します。詳細については、<code>WorkflowView</code> を参照してください。	いいえ
<code>perspectiveName</code>	選択するパースペクティブの名前を指定します。このパラメーターを指定した場合、<code>scope</code> パラメーターは <code>full</code> と <code>data</code> の 2 つの値のみを持つことができます。	<code>PerspectiveActionId</code> または <code>perspectiveActionName</code> が指定されている場合のみ。
<code>perspectiveActionId</code>	**非推奨。**代わりに <code>perspectiveActionName</code> の使用を検討してください。選択するパースペクティブアクションの識別子を指定します。	いいえ
<code>perspectiveActionName</code>	選択するパースペクティブアクションの一意の名前を指定します。	いいえ

レイアウトパラメーター

パラメーター	説明	必須
<code>scope</code>	Web コンポーネントが使用するスコープを指定します。値は、<code>full</code>、<code>data</code>、<code>dataspace</code>、<code>dataset</code>、または <code>node</code> です。 詳細については、 <code>UIHttpManagerComponent.Scope</code> を参照してください。	いいえ。デフォルトは、ターゲットの選択に応じて可能な限り最小になるように計算されます。
<code>firstCallDisplay</code>	選択と <code>scope</code> パラメーターの組み合わせによって決定される表示の代わりに使用する必要がある表示を指定します。 可能な値は次のとおりです。 <code>auto</code> : 選択に応じて表示が自動的に設定されます。 <code>view</code> : 表形式、階層、またはタイルビューの表示を強制します。 <code>record</code> : 述部に少なくとも 1 つのレコードがある場合、リストの最初のレコードを強制的に表示します。 例 <code>firstCallDisplay=view</code>	いいえ。デフォルトはターゲットの選択に従って計算されます。

パラメーター	説明	必須
	<code>firstCallDisplay=view:hierarchyExpanded</code> <code>firstCallDisplay=view:hierarchyExpandedToSearch</code> <code>firstCallDisplay=record</code> <code>firstCallDisplay=record:</code> <code>{predicate}</code> 詳細については、 <code>UIHttpManagerComponent.setFirstCallDisplay</code> を参照してください。 詳細については、 <code>UIHttpManagerComponent.setFirstCallDisplayHierarchyExpanded</code> を参照してください。 詳細については、 <code>UIHttpManagerComponent.setFirstCallDisplayHierarchyExpandedTo</code> を参照してください。 詳細については、 <code>UIHttpManagerComponent.setFirstCallDisplayRecord</code> を参照してください。	
<code>closeButton</code>	セッション終了ボタンの表示方法を指定します。値は、<code>logout</code> または <code>cross</code> にすることができます。 詳細については、 <code>UIHttpManagerComponent.CloseButtonSpec</code> を参照してください。	いいえ。スコープが <code>full</code> でない場合、デフォルトでは [閉じる] ボタンは表示されません。
<code>dataSetFeatures</code>	データセットレベルまたはテーブル外のフォームで UI サービスに表示する機能を指定します。	いいえ

パラメーター	説明	必須
	これらのオプションは、ワークスペースの機能にのみ関係します。このプロパティは、可能な限り最小の <code>スコープ</code>、つまり <code>データセット</code> または <code>ノード</code> で使用することをお勧めします。 構文 <pre><prefix> ":" <feature> ["," <feature>]*</pre> 説明 • <code><prefix></code> は <code>hide</code> または <code>show</code> • <code><feature></code> は <code>services</code>、<code>title</code>、<code>save</code>、または <code>revert</code> 例 <pre>hide:title show:save, revert</pre> 詳細については、<code>UIHttpManagerComponent.DataSetFeatures</code> を参照してください。	
<code>viewFeatures</code>	表形式、階層、またはタイトルビュー (テーブルレベル) で表示する機能を指定します。	いいえ

パラメーター	説明	必須
	これらのオプションは、ワークスペースの機能にのみ関係します。このプロパティは、可能な限り最小の <code>スコープ</code>、つまり <code>データセット</code> または <code>ノード</code> で使用することをお勧めします。 構文 <pre><prefix> ":" <feature> ["," <feature>]*</pre> 説明 • <code><prefix></code> は <code>hide</code> または <code>show</code> • <code><feature></code> は <code>create</code>、<code>views</code>、<code>selection</code>、<code>filters</code>、<code>services</code>、<code>refresh</code>、<code>title</code>、または <code>breadcrumb</code> 例 <pre>hide:title,selection show:service,title,breadcrumb</pre> 詳細については、<code>UIHttpManagerComponent.ViewFeatures</code> を参照してください。	

パラメーター	説明	必須
<code>recordFeatures</code>	レコードレベルでフォームに表示する必要がある機能を指定します。 これらのオプションは、ワークスペースの機能にのみ関係します。このプロパティは、可能な限り最小の <code>スコープ</code>、つまり <code>データセット</code> または <code>ノード</code> で使用することをお勧めします。 構文 <pre><prefix> ":" <feature> ["," <feature>]*</pre> 説明 • <code><prefix></code> は <code>hide</code> または <code>show</code> • <code><feature></code> は <code>services</code>、<code>title</code>、<code>breadcrumb</code>、<code>save</code>、<code>saveAndClose</code>、<code>close</code>、または <code>revert</code> 例 <pre>hide:title show:save, saveAndClose, revert</pre> 詳細については、<code>UIHttpManagerComponent.RecordFeatures</code> を参照してください。	いいえ

パラメーター	説明	必須
<code>pageSize</code>	テーブルビュー (表形式、階層形式、またはタイル形式) でページごとに表示されるレコードの数を指定します。	いいえ
<code>startWorkItem</code>	作業項目を自動的に取得して開始する必要があることを指定します。値は <code>true</code> または <code>false</code> にすることができます。 詳細については、 <code>ServiceKey.WORKFLOW</code> を参照してください。	いいえ。デフォルト値は <code>false</code> で、ターゲットの作業項目の状態は変更されません。

EBX® Web コンポーネントへの呼び出しの例

最小 URI

```
http://localhost:8080/ebx/
```

ユーザー「admin」としてログインし、「参照」データスペースを選択します。

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=Reference
```

「参照」データスペースを選択し、ビルトインの検証サービスにアクセスします。

```
http://localhost:8080/ebx/?  
login=admin&password=admin&branch=Reference&service=@validation
```

デフォルトディレクトリのロールテーブルを選択します。

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/roles
```

デフォルトディレクトリのレコード「admin」を選択します。

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/users[./login="admin"]
```

デフォルトディレクトリに新しいユーザーを作成するためのインターフェイスにアクセスします。

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/users&service=@creation
```

デフォルトディレクトリのレコード「admin」をレコード「jSmith」と比較します。

データスペース「Reference」のデータセット「instanceId」のレコード「R1」をレコード「R0」と比較します。

```
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"]
```

ビルトインのユーザーサービス

EBX®には、多数のビルトインユーザーサービスが含まれています。ビルトインのユーザーサービスを使用できます。

- ワークフローモデルタスクを定義する場合
- パースペクティブアクションメニュー項目を定義する場合
- サービス拡張機能とともに使用する場合の拡張ユーザーサービスとして
- EBX®をWebコンポーネントとして使用する場合

このリファレンスページでは、ビルトインのユーザーサービスとそのパラメーターについて説明します。

- データへのアクセス (デフォルトサービス)
 - 入力パラメーター
- 新しいレコードの作成
 - 入力パラメーター
 - 出力パラメーター
- レコードの複製
 - 入力パラメーター
 - 出力パラメーター
- データの XML ファイルへのエクスポート
 - 入力パラメーター
- データの CSV ファイルへのエクスポート
 - 入力パラメーター
- XML ファイルからのデータインポート
 - 入力パラメーター
- CSV ファイルからのデータインポート
 - 入力パラメーター
- データスペースへのアクセス
 - 入力パラメーター
- データスペース、スナップショット、またはデータセットの検証
 - 入力パラメーター
 - 出力パラメーター
- データスペースのマージ
 - 入力パラメーター

- 出力パラメーター
- データスペースマージビューへのアクセス
 - 入力パラメーター
- コンテンツの比較
 - 入力パラメーター
- データワークフロー
 - 入力パラメーター

データへのアクセス (デフォルトサービス)

デフォルトでは、ワークフローはこのサービスを自動的に完了したと見なします。つまり、ボタンは常に使用できます。

これは、サービスが指定されていない場合に使用されるデフォルトのサービスです。

入力パラメーター

パラメーター	ラベル	説明
<code>branch</code>	データスペース	指定されたデータスペースの識別子 - このサービスにはデータスペースが必要です。
<code>disableAutoComplete</code>	起動時に を無効化	デフォルトでは、このサービスに関連付けられているインタラクションは直接完了したと見なされます。したがって、作業項目の開始時に ボタンが自動的に表示されます。このパラメーターは、この動作を無効にするのに役立ちます。値が「true」の場合、開発者は、たとえば、ユーザーサービスまたはトリガーで <code>SessionInteraction</code> を使用して対話を完了する責任があります。デフォルト値は「false」です。パースペク

パラメーター	ラベル	説明
		タイプはこのパラメーターを使用しません。
<code>firstCallDisplay</code>	初期コール表示モード	最初の呼び出し時にフィルター処理されたテーブルまたはレコードを表示するときに使用する必要がある表示モードを定義します。デフォルト (value = 'auto') : 選択に応じて表示が自動的に設定されます。ビュー (value = 'view') : 表形式ビューまたは階層ビューの表示を強制します。レコード (value = 'record') : 述部に少なくとも 1 つのレコードがある場合、レコードフォームの表示を強制します。
<code>instance</code>	データセット	値は、選択したデータスペースに存在するデータセットの参照にする必要があります。
<code>scope</code>	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
<code>trackingInfo</code>	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や追加のエクスポート情報な

パラメーター	ラベル	説明
		ど、他の目的にも使用できます。
version	スナップショット	指定されたスナップショットの識別子-このサービスにはデータスペースが必要です。
viewPublication	ビュー	表示するビューのパブリケーション名。選択したテーブルに対してビューを構成する必要があります。
xpath	データセットノード (XPath)	値は、選択したデータセット内の有効な絶対位置パスにする必要があります。表記は、簡略化された構文で、簡略化された XPath に準拠している必要があります。

新しいレコードの作成

ワークフローの場合、最初の正常な送信が実行されたとき (レコードが作成されたとき) に、作成サービスは完了したと見なされます。このサービスがすでに完了しているときに呼び出された場合、作成されたレコードは更新モードまたは読み取り専用モードで表示されます (ユーザー権限によって異なります)。

サービス名パラメーター： `service=@creation`

入力パラメーター

パラメーター	ラベル	説明
branch	データスペース	指定されたデータスペースの識別子 - このフィールドは、このサービスに必要です。
instance	データセット	値は、選択したデータスペースに存在するデータセットの参照にする必要があります - このフィールドは、このサービスに必要です。

パラメーター	ラベル	説明
scope	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
trackingInfo	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や追加のエクスポート情報など、他の目的にも使用できます。
xpath	データセットテーブル (XPath)	値は、選択したデータセット内のテーブルの有効な絶対ロケーションパスにする必要があります。表記は、簡略化された構文の簡略化された XPath に準拠している必要があります - このフィールドは、このサービスに必要です。

出力パラメーター

パラメーター	ラベル	説明
created	作成されたレコード	作成されたレコードの XPath が含まれます。

レコードの複製

ワークフローの場合、最初の正常な送信が実行されたとき (レコードが作成されたとき) に、複製サービスは完了したと見なされます。このサービスがすでに完了しているときに呼び出された場合、作成されたレコードは更新モードまたは読み取り専用モードで表示されます (ユーザー権限によって異なります)。

サービス名パラメーター： `service=@duplicate`

入力パラメーター

パラメーター	ラベル	説明
<code>branch</code>	データスペース	指定されたデータスペースの識別子 - このフィールドは、このサービスに必要です。
<code>instance</code>	データセット	値は、選択したデータスペースに存在するデータセットの参照にする必要があります - このフィールドは、このサービスに必要です。
<code>scope</code>	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
<code>trackingInfo</code>	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や追加のエクスポート情報など、他の目的にも使用できます。

パラメーター	ラベル	説明
<code>xpath</code>	複製するレコード (XPath)	値は、既存のレコードの有効な絶対位置パスにする必要があります。表記は、簡略化された構文の簡略化された XPath に準拠している必要があります-このフィールドは、このサービスに必要です。

出力パラメーター

パラメーター	ラベル	説明
<code>created</code>	作成されたレコード	作成されたレコードの XPath が含まれます。

データを XML ファイルにエクスポート

exportToXML サービスは、エクスポートが完了してファイルがダウンロードされたときに完了したと見なされます。

サービス名パラメーター： `service=@exportToXML`

入力パラメーター

パラメーター	ラベル	説明
<code>branch</code>	データスペース	指定されたデータスペースの識別子 - このサービスにはデータスペースが必要です。
<code>instance</code>	データセット	値は、選択したデータスペースに存在するデータセットの参照にする必要があります - このフィールドは、このサービスに必要です。
<code>scope</code>	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選

パラメーター	ラベル	説明
		択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
trackingInfo	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や追加のエクスポート情報など、他の目的にも使用できます。
version	スナップショット	指定されたスナップショットの識別子-このサービスにはデータスペースが必要です。
xpath	エクスポートするデータセットテーブル (XPath)	値は、選択したデータセット内のテーブルの有効な絶対ロケーションパスにする必要があります。表記は、簡略化された構文の簡略化された XPath に準拠している必要があります - このフィールドは、このサービスに必要です。

データを CSV ファイルにエクスポート

ワークフローは、エクスポートが完了してファイルがダウンロードされると、exportToCSV サービスが完了したと見なします。

サービス名パラメーター： `service=@exportToCSV`

入力パラメーター

パラメーター	ラベル	説明
branch	データスペース	指定されたデータスペースの識別子 - このサービスにはデータスペースが必要です。
instance	データセット	値は、選択したデータスペースに存在するデータセットの参照にする必要があります - このフィールドは、このサービスに必要です。
scope	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
trackingInfo	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や追加のエクスポート情報など、他の目的にも使用できます。
version	スナップショット	指定されたスナップショットの識別子-このサービスにはデータスペースが必要です。
xpath	エクスポートするデータセットテーブル (XPath)	値は、選択したデータセット内のテーブルの有効な絶対ロケーションパスにする必要があります。表記は、簡略化された構文の簡略化された XPath に準拠している必要が

パラメーター	ラベル	説明
		あります - このフィールドは、このサービスに必要です。

データを XML ファイルからインポート

ワークフローは、インポートが実行されると、importFromXML サービスが完了したと見なされます。

サービス名パラメーター： `service=@importFromXML`

入力パラメーター

パラメーター	ラベル	説明
<code>branch</code>	データスペース	指定されたデータスペースの識別子 - このサービスにはデータスペースが必要です。
<code>instance</code>	データセット	値は、選択したデータスペースに存在するデータセットの参照にする必要があります - このフィールドは、このサービスに必要です。
<code>scope</code>	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
<code>trackingInfo</code>	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や

パラメーター	ラベル	説明
		追加のエクスポート情報など、他の目的にも使用できます。
xpath	インポートするデータセット テーブル (XPath)	値は、選択したデータセット内のテーブルの有効な絶対ロケーションパスにする必要があります。表記は、簡略化された構文の簡略化された XPath に準拠している必要があります - このフィールドは、このサービスに必要です。

データを CSV ファイルからインポート

ワークフローは、インポートが実行されると、importFromCSV サービスが完了したと見なされません。

サービス名パラメーター： `service=@importFromCSV`

入力パラメーター

パラメーター	ラベル	説明
branch	データスペース	指定されたデータスペースの識別子 - このサービスにはデータスペースが必要です。
instance	データセット	値は、選択したデータスペースに存在するデータセットの参照にする必要があります - このフィールドは、このサービスに必要です。
scope	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選択できるエンティティを定義します。空白のままにすると、デフォルト値が使用され

パラメーター	ラベル	説明
		ます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
trackingInfo	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や追加のエクスポート情報など、他の目的にも使用できます。
xpath	インポートするデータセットテーブル (XPath)	値は、選択したデータセット内のテーブルの有効な絶対ロケーションパスにする必要があります。表記は、簡略化された構文の簡略化された XPath に準拠している必要があります - このフィールドは、このサービスに必要です。

データスペースへのアクセス

ワークフローは、データスペース選択サービスが完了したと自動的に見なします。

サービス名パラメーター： `service=@selectDataSpace`

入力パラメーター

パラメーター	ラベル	説明
branch	データスペース	指定されたデータスペースの識別子 - このサービスにはデータスペースが必要です。
scope	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選

パラメーター	ラベル	説明
		択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
trackingInfo	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や追加のエクスポート情報など、他の目的にも使用できます。
version	スナップショット	指定されたスナップショットの識別子-このサービスにはデータスペースが必要です。

データスペース、スナップショット、またはデータセットの検証

ワークフローは、検証サービスが完了したと自動的に見なします。

サービス名パラメーター： `service=@validation`

入力パラメーター

パラメーター	ラベル	説明
branch	データスペース	指定されたデータスペースの識別子 - このサービスにはデータスペースまたはスナップショットが必要です。
instance	データセット	値は、選択したデータスペースに存在するデータセットの参照にする必要があります。

パラメーター	ラベル	説明
scope	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
trackingInfo	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や追加のエクスポート情報など、他の目的にも使用できます。
version	スナップショット	指定されたスナップショットの識別子-このサービスにはデータスペースまたはスナップショットが必要です。

出力パラメーター

パラメーター	ラベル	説明
hasError	エラーを検出しました	検証でエラーが発生した場合は、「true」が含まれます。
hasFatal	致命的なエラーを検出しました	検証によって致命的なエラーが発生した場合は、「true」が含まれます。
hasInfo	情報を検出しました	検証によって情報が生成された場合は、「true」が含まれます。

パラメーター	ラベル	説明
hasWarning	警告を検出しました	検証で警告が生成された場合は、「true」が含まれます。

データスペースのマージ

ワークフローは、マージが実行され、データスペースが閉じられると、マージサービスが完了したと見なします。

サービス名パラメーター： `service=@merge`

入力パラメーター

パラメーター	ラベル	説明
branch	データスペース	指定されたデータスペースの識別子 - このサービスにはデータスペースが必要です。
scope	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
trackingInfo	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や追加のエクスポート情報など、他の目的にも使用できます。

出力パラメーター

パラメーター	ラベル	説明
<code>mergeResult</code>	マージ成功	マージが成功した場合は「true」、それ以外の場合は「false」が含まれます。
<code>mergeState</code>	マージ状態	マージの戻りコードが含まれています。 InteractionMergeState UIHttpManagerComponentReturnCode を使用してこの値を解析することを強くお勧めします。

データスペースマージビューへのアクセス

merge.view サービスは、自動的に完了したと見なされます。

サービス名パラメーター：`service=@merge.view`

入力パラメーター

パラメーター	ラベル	説明
<code>branch</code>	データスペース	指定されたデータスペースの識別子 - このサービスにはデータスペースが必要です。
<code>scope</code>	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。

パラメーター	ラベル	説明
<code>trackingInfo</code>	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や追加のエクスポート情報など、他の目的にも使用できます。

コンテンツの比較

ワークフローは、比較サービスが完了したと自動的に見なします。

サービス名パラメーター：`service=@compare`

入力パラメーター

パラメーター	ラベル	説明
<code>branch</code>	データスペース	指定されたデータスペースの識別子-このサービスには、比較するデータスペースまたはスナップショットとデータスペースまたはスナップショットが必要です。
<code>compare.branch</code>	比較するデータスペース	比較するデータスペースの識別子 - このサービスには、比較するデータスペースまたはスナップショットとデータスペースまたはスナップショットが必要です。
<code>compare.filter</code>	比較フィルター	比較で継承フィールドと関数フィールドを無視するには (解決モードを無効にする)、フィルター 「persistedValuesOnly」を指定する必要があります。デフォルトでは、フィルターが定義されていない場合、比較では解決済みモードが使用されます。

パラメーター	ラベル	説明
<code>compare.instance</code>	比較するデータセット	値は、比較するために選択したデータスペースに存在するデータセットの参照に必要があります。
<code>compare.version</code>	比較するスナップショット	比較するスナップショットの識別子 - このサービスには、比較するデータスペースまたはスナップショットとデータスペースまたはスナップショットが必要です。
<code>compare.xpath</code>	比較するテーブルまたはレコード (XPath)	値は、比較するために、選択したデータセット内のテーブルまたはレコードの有効な絶対位置パスにする必要があります。表記は、簡略化された構文で、簡略化された XPath に準拠している必要があります。
<code>instance</code>	データセット	値は、選択したデータスペースに存在するデータセットの参照に必要があります。
<code>scope</code>	スコープ	このサービスのユーザーナビゲーションの範囲、つまり、ユーザーがセッション中に選択できるエンティティを定義します。空白のままにすると、デフォルト値が使用されます。パースペクティブの場合、デフォルト値は常に「ノード」です。ワークフローの場合、デフォルト値は選択したエンティティまたはサービスによって異なります。
<code>trackingInfo</code>	追跡情報	追跡情報は ログに記録されます。また、アクセス制御や

パラメーター	ラベル	説明
		追加のエクスポート情報など、他の目的にも使用できます。
version	スナップショット	指定されたスナップショットの識別子-このサービスには、比較するデータスペースまたはスナップショットとデータスペースまたはスナップショットが必要です。
xpath	テーブルまたはレコード (XPath)	値は、選択したデータセット内のテーブルまたはレコードの有効な絶対位置パスにする必要があります。表記は、簡略化された構文で、簡略化された XPath に準拠している必要があります。

データワークフロー

このサービスは、データワークフローのユーザーインターフェイスへのアクセスを提供します。

サービス名パラメーター： `service=@workflow`

注意:

このサービスは、パースペクティブのみを対象としています。

入力パラメーター

パラメーター	ラベル	説明
scope	スコープ	このサービスのユーザーナビゲーションの範囲を定義します。

パラメーター	ラベル	説明
<code>viewPublication</code>	ビューパブリケーション	このサービスに適用するビューのパブリケーション名を定義します。
<code>workflowView</code>	ワークフロービュー	ワークフロービュータイプを指定します。値は、「inbox」、「launcher」、「monitoringPublications」、「monitoringWorkflows」、「monitoringWorkItems」、または「completedWorkflows」のいずれかになります。
<code>xpath</code>	フィルター (XPath)	オプションのフィルター。構文は、「」で囲まれた XPath 述語に準拠している必要があります。

サポートされている XPath 構文

- 概要
- 式の例
 - 絶対パス
 - 相対パス
 - ルートパスと子孫パス
 - 述語を含むテーブルパス
 - 複雑な述語
 - パラメーター付きの述語
 - 検索述語
 - 検証検索の述語
- XPath 式の構文仕様
 - 概要
 - 述語仕様

- 外部キーの抽出

- [Java API](#)

概要

TIBCO EBX® で使用される XPath 表記は、[XML Path Language \(XPath\) バージョン 1.0](#) 標準の省略構文に準拠している必要がありますが、特定の制限があります。このドキュメントでは、サポートされている省略構文について詳しく説明します。

式の例

一般的な XPath 式は次のとおりです。

```
path
```

絶対パス

```
/library/books/
```

相対パス

```
./Author
```

```
../Title
```

ルートパスと子孫パス

```
//books
```

述語を含むテーブルパス

```
../../books/[author_id = 0101 and (publisher = 'harmattan')]
```

```
/library/books/[not(publisher = 'dumesnil')]
```

複雑な述語

```
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') )
```

パラメーター付きの述語

```
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'
```

注意:

パラメーター値は Java API のメソッド `Request.setXPathParameter` によってのみ設定できるため、この表記の使用は Java API に制限されています。

検索述語

- 構文: `osd:search(fields, queryString[, templateKey])`
- 例
 - `osd:search('col1', 'xxx')`

- `osd:search('col1,col2','xxx')`
- `osd:search('', 'xxx')`
- `osd:search(col1, 'xxx', myTemplate@myModule)`

`osd:search` 関数は、用語または用語のリストを現在のテーブルのフィールドのセットと照合しようとします。この関数は汎用であり、EBX® でサポートされるすべてのフィールドデータ型を処理します。フィールドが指定されていない場合は、すべてのフィールド (履歴テーブルの場合) またはすべての最適化されたフィールド (その他のすべての場合) に対して検索されます。最適化されたフィールドと最適化されていないフィールドの詳細については、[制限事項](#)を参照してください。

最適化されたテキスト検索フィルターは、最適化されていないフィルターまたは別のテーブルを含むフィルターと OR 結合できません。詳細については、[技術的制限事項](#)を参照してください。

関連するフィールドについて、ラベルが存在する場合、検索は値ではなくラベルを対象とします。ただし、場合によってはまだサポートされていません。詳細については、[値のラベル付け](#)を参照してください。

より高度な使用法として、クエリ文字列で特殊な演算子がサポートされています。詳細については、[特殊文字](#)を参照してください。

以下も参照してください。:

- [クイック検索](#)

述語 `osd:search` はローカライズされています。

注意:

ロケールは、Java API `Request.setLocale` または `Request.setSession` のメソッドで設定できます。

注意:

検索テンプレートの識別子を指定して、検索の動作をカスタマイズできます。

検証検索の述語

```
osd:has-validation-item()
```

```
osd:has-validation-item('error,info')
```

```
osd:contains-validation-message('xxx')
```

```
osd:contains-validation-message('xxx','info,warning')
```

- 検証検索用の XPath 関数は、関連付けおよび外部キーフィルターで定義された XPath 述語では使用できません。
- 述語 `osd:label`、`osd:contains-record-label`、および `osd:contains-validation-message` はローカライズされています。

注意:

ロケールは、Java API `Request.setLocale` または `Request.setSession` のメソッドで設定できます。

注意:

最新の検証レポートで検索が確実に実行されるようにするには、これらの述部を使用する直前に、テーブルの明示的な検証を実行する必要があります。

XPath 式の構文仕様

概要

式	形式	例
XPath 式	<code><container path></code>	<code>/books[title='xxx']</code>
<code><container path></code>	<code><absolute path></code> または <code><relative path></code>	
<code><absolute path></code>	<code>/a/b</code> または <code>//b</code>	<code>//books</code>
<code><relative path></code>	<code>../b</code> , <code>../b</code> または <code>b</code>	<code>../b/books</code>

述語仕様

式	形式	注意/例
<code><predicate></code>	例: A and (B or not(C)) A,B,C: <code><atomic expression></code>	構成: 論理演算子の括弧、 not() およびアトム式。
<code><atomic expression></code>	<code><path><comparator><criterion></code> または <code>method(<path>,<criterion>)</code>	<code>royalty = 24.5</code> <code>starts-with(title, 'Johnat')</code> <code>booleanValue = true</code>
<code><path></code>	<code><relative path></code> または <code>osd:label(<relative path>)</code>	それを含むテーブルの相対 パス <code>../authorstitle</code>
<code><comparator></code>	<code><boolean comparator></code> <code><numeric comparator></code> または <code><string comparator></code>	

式	形式	注意/例
<i><boolean comparator></i>	= または !=	
<i><numeric comparator></i>	= , != , < , > , <= , または >=	
<i><string comparator></i>	= または !=	
<i><method></i>	• <i><date method></i> • <i><string method></i> • <code>osd:is-null</code> メソッドまたは • <code>osd:is-not-null</code> メソッド	
<i><date, time & dateTime method></i>	• <code>date-less-than</code> • <code>date-equal</code> または • <code>date-greater-than</code>	
<i><string method></i>	• <code>contains</code> • <code>osd:contains-case-insensitive</code> • <code>osd:contains-record-label</code> • <code>osd:is-equal-case-insensitive</code> • <code>osd:is-empty</code> • <code>osd:is-empty-or-nil</code> • <code>osd:is-neither-empty-nor-nil</code>	

式	形式	注意/例
	<code>osd:is-not-empty</code> <code>ends-with</code> <code>osd:ends-with-case-insensitive</code> <code>matches</code> <code>osd:search</code> <code>starts-with</code> または <code>osd:starts-with-case-insensitive</code>	
<code><criteria></code>	<code><boolean criterion></code> <code><date criterion></code> <code><dateTime criterion></code> <code><numeric criterion></code> <code><string criterion></code> または <code><time criterion></code>	
<code><boolean criterion></code>	<code>true</code> , <code>false</code>	
<code><numeric criterion></code>	整数または小数	<code>-4.6</code>
<code><string criterion></code>	引用文字列	<code>'azerty'</code>
<code><date criterion></code>	'yyyy-MM-dd' として引用およびフォーマット設定	<code>'2007-12-31'</code>
<code><time criterion></code>	'HH:mm:ss' または 'HH:mm:ss.SSS' として引用およびフォーマット設定	<code>'11:55:00'</code>
<code><dateTime criterion></code>	'yyyy-MM-ddTHH:mm:ss' または 'yyyy-MM-ddTHH:mm:ss.SSS' として	<code>'2007-12-31T11:55:00'</code>

式	形式	注意/例
	引用およびフォーマット設定	

XPath 1.0 式

アトミック述語式の基準値部分 (右側) で XPath 1.0 式を使用することができます。

たとえば、`[./a=3]` の代わりに、式 `[./a=(floor(./d)+ 2.0)]` を使用できます。

述語はデータモデルノードと基準のノードタイプに強く依存するため、アトミック述語式のパス部分 (左側) はノードパスにする必要があります、XPath 式にすることはできません。たとえば、式 `/table[floor(./a)> ceiling(./d)]` は無効です。

ラベルの述語

`osd:label()` 関数は、値の代わりにラベル上の述語を解決するために、アトミック述語のパス部分に適用できます。この場合、使用できるのは文字列演算子と文字列基準のみです。つまり、`ends-with(osd:label(./price), '99')` です。

ラベル上の述語はローカライズされているため、条件は述語フィルター処理された要求と同じロケールで表現する必要があります。例：
`request.setLocale(Locale.FRENCH);`
`request.setXPathFilter("osd:label(./delivery_date)='30/12/2014'");`

注意:

述語の右側がコンテキスト値である場合、`osd:label` 関数を使用することは禁止されています。

注意:

`osd:label` 関数がデータモデルで使用されている場合、たとえば、テーブル参照ノードの選択またはフィルター述語で、データモデルのデフォルトロケール (モジュール宣言で定義されている) は、基準形式に使用する必要があります (これは一般的に推奨されていません)。

以下も参照してください。:

- `SchemaNode.displayOccurrence`

コンテキスト値

選択したノードに関連する述語の場合、基準値 (つまり、述語の右側) を構文 `${<relative-path>}` を使用するコンテキストパスで置き換えることができます。ここで、`<relative-path>` は、選択したノードを基準としたエレメントの相対パスです。基準値は単一の値にする必要があります。つまり、集約されたリストはコンテキストパスのターゲットとして許可されません。

注意:

メソッドを呼び出す場合、基準は 2 番目のパラメーターであり、最初のパラメーターを相対値にすることはできません。

集約リスト

集約リストの述語の場合、リストエレメントの 1 つが述語を検証すると、コンパレーターに関係なく、述語は `true` を返します。

さらに、述語 `osd:is-empty` および `osd:is-not-empty` は、リストが空かどうかに関係なく、タイプがリストである任意のノードに適用できます。

注意:

比較演算子 `!=` には特に注意が必要です。たとえば、集約されたリストの場合、`./list != 'a'` は `not(./list = 'a')` と同じではありません。リストにエレメント `(e1,e2,...)` が含まれている場合、最初の述語は、`e1 != 'a' or e2 != 'a' ...` と同等であり、2 番目の述語は、`e1 != 'a' and e2 != 'a' ...` と同等です。

「Null」値

null 値は、演算子 `osd:is-null` および `osd:is-not-null` を使用して、述語で明示的に処理する必要があります。

たとえば、`/root/products[./price<100]` または `/root/products[./price!=100]` は、価格が (`null`) に設定されていない商品を返しません。後者の場合も、未設定の値を返すには、述語を `/root/products[./price!=100 または osd:is-null(/price)]` にする必要があります。

リテラル式で一重引用符と二重引用符を管理する方法

デフォルトでは、リテラル式は一重引用符 (`'`) で区切られます。リテラル式に一重引用符が含まれ、二重引用符が含まれていない場合、式は二重引用符 (`"`) で区切る必要があります。リテラル式に一重引用符と二重引用符の両方が含まれている場合は、一重引用符を二重引用符で囲む必要があります。

Java API のメソッド `XPathExpressionHelper.encodeLiteralStringWithDelimiters` がこれを処理します。

`encodeLiteralStringWithDelimiters` の使用例

リテラル式の値	このメソッドの結果
<code>Coeur</code>	<code>'Coeur'</code>
<code>Coeur d'Alene</code>	<code>"Coeur d'Alene"</code>
<code>He said: "They live in Coeur d'Alene".</code>	<code>'He said: "They live in Coeur d'Alene".'</code>

外部キーの抽出

EBX® では、外部キーは `osd:tableRef` 宣言で 1 つのフィールドにグループ化されます。

標準の XPath 構文が拡張され、ターゲットの主キーフィールドの値が抽出されるようになりました。

例

テーブル `/root/tableA` に「fkB」という `osd:tableRef` フィールドがあり、そのターゲットが `/root/tableB` で、`tableB` の主キーが `xs:int` 型の `id` と `xs:date` 型の `date` という 2 つのフィールドがある場合、以下の式が有効となります。

- `/root/tableA[fkB = '123|2008-01-21']`。ここで、文字列 `"123|2008-01-21"` は、主キー値全体を表します。

詳細については、[Syntax of the internal String representation of primary keys](#) を参照してください。

- `/root/tableA[fkB/id = 123 and date-equal(fkB/date, '2008-01-21')]`。ここで、この述語は前の例の述語と同等ですが、より効率的です。
- `/root/tableA[fkB/id >= 123]`。対象となる主キーフィールドのタイプは `xs:int` であるため、任意の数値演算子を使用できます。
- `/root/tableA[date-greater-than( ./fkB/date, '2007-01-01' )]`。対象となる主キーフィールドがタイプ `xs:date` であるため、任意の日付演算子を使用できます。
- `/root/tableA[fkB = ""]` は、対象の主キーに 2 つの列があるため、無効です。
- `/root/tableA[osd:is-null(fkB)]` は、外部キーが `null` (未定義) であるかどうかを確認します。

Java API

Java API での XPath の使用

Java API では、`XPathFilter` クラスを使用して、XPath 述語を定義し、それらに対して要求を実行できます。

`XPathExpressionHelper` クラスは、XPath 述語とパスを処理するための実用的なメソッドを提供します。

ローカリゼーション

- [ラベル付けとローカリゼーション](#)
- [TIBCO EBX® ローカリゼーションの拡張](#)

ラベル付けとローカリゼーション

- [概要](#)
 - [ユーザーインタラクションのローカライズ](#)
 - [テキスト情報](#)
- [値のフォーマットポリシー](#)
- [ロケールの構文](#)

概要

TIBCO EBX®は、データモデルのラベリングと国際化を処理する機能を提供します。

ユーザーインタラクションのローカライズ

EBX®では、言語設定を2つのスコープに設定できます。

1. セッション：各ユーザーは、ユーザーペインからデフォルトのロケールを選択できます。
2. データモデル：データモデルがEBX®でネイティブにサポートされている言語以外の言語にローカライズされている場合、ユーザーはその特定のデータモデルに対してそれらの言語の1つを選択できます。詳細については、[TIBCO EBX® 国際化の拡張](#)を参照してください。

テキスト情報

EBX®では、ほとんどのマスターデータエンティティにラベルと説明を付けるか、ユーザーメッセージに対応させることができます。たとえば、次のとおりです。

- データスペース、スナップショット、およびデータセットには、独自のラベルと説明を付けることができます。ラベルは一意的な名前から独立しているため、ローカライズおよび変更可能です。
- データモデル内の任意のノードは、静的なラベルと説明を持つことができます。
- 値を列挙するときに、静的ラベルを付けることができます。
- 検証メッセージはカスタマイズでき、許可制限は理由を説明するテキストを提供できます。
- 各レコードは、その内容、および表示されているコンテキスト (階層内、外部キーなど) に従って動的に表示されます。

このすべてのテキスト情報は、モジュールによって宣言されたロケールにローカライズできます。

以下も参照してください。:

- [ラベルとメッセージ](#)
- [テーブル宣言](#)
- [外部キー宣言](#)

値のフォーマットポリシー

値がユーザーに表示されると、そのタイプと現在のロケールのフォーマットポリシーに従ってフォーマットされます。たとえば、日付は、一部のロケールでは「dd/MM/yyyy」、その他のロケールでは「MM/dd/yyyy」として表示されます。

フォーマットポリシーは、**単純型**の値を表示する方法を定義するために使用されます。

モジュールによって宣言されたロケールごとに、そのフォーマットポリシーは、`/WEB-INF/ebx/{locale}/frontEndFormattingPolicy.xml` ファイルで構成されます。たとえば、ギリシャ語 (`el`) のフォーマットポリシーを定義するために、エンジンはモジュール内で次のパスを探します。

```
/WEB-INF/ebx/el/frontEndFormattingPolicy.xml
```

対応するファイルがモジュールに存在しない場合、フォーマットポリシーはEBX®のクラスパスで検索されます。ロケール固有のフォーマットポリシーが見つからない場合は、`en_US` のフォーマットポリシーが適用されます。

ファイル `frontEndFormattingPolicy.xml` の内容は次のとおりです。

```
<?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>
```

エレメント `date`、`dateTime`、および `time` は必須です。

フォーマット設定された数値に表示されるグループおよび小数点記号は、エレメント `decimal` および `int` の属性 `groupingSeparator` および `decimalSeparator` を定義することで変更できます。

ロケールの構文

ロケールを表現する方法は 2 つあります。

1. XML の推奨事項は、区切り文字としてハイフン「-」を使用する [RFC 5646 - The Language Tag](#) の推奨事項に従います。
2. Java 仕様では、ハイフンの代わりにアンダースコア「_」を使用しています。

EBX® によって読み取られる XML ファイル (XSD、フォーマットポリシーファイルなど) では、どちらの構文も使用できます。

Web パス、つまり Web アプリケーション内のパスの場合、Java 構文のみが許可されます。したがって、フォーマットポリシーファイルは、ロケール名が Java 構文を使用可能なディレクトリに配置する必要があります。

以下も参照してください。:

- [TIBCO EBX® ローカリゼーションの拡張](#)

TIBCO EBX® ローカリゼーションの拡張

- [ネイティブ EBX® ローカリゼーションの概要](#)
- [言語パックソフトウェア](#)
- [EBX® ユーザーインターフェイスのローカリゼーションの拡張](#)
 - [新しいロケールの追加](#)
- [ローカライズされたリソースの解決](#)
- [既知の制限事項](#)

- 。拡張不可能なマテリアル

ネイティブ EBX® ローカリゼーションの概要

デフォルトでは、EBX® ビルトインユーザーインターフェイスは英語 (en-US) とフランス語 (fr-FR) で提供されます。

ローカリゼーションは、フォーマットポリシーと一連のメッセージファイル (リソースバンドル) で構成されます。

- 英語の場合、ローカリゼーションは、フォーマットポリシーと、ロケールが定義されていない一連のメッセージファイルによって提供されます。
- フランス語の場合、ローカリゼーションは、フォーマットポリシーと、ロケールが「fr」に設定された一連のメッセージファイルによって提供されます。

EBX® は、ユーザーインターフェイスのローカリゼーションを拡張し、データモデルと関連サービスのドキュメントを国際化するためにロケールを追加するオプションを提供します。

言語パックソフトウェア

TIBCO EBX® およびアドオンの言語パックは、EBX® リリースに応じて [TIBCO eDelivery](#) から入手できます。言語パックにはオンラインヘルプも含まれます。詳細は、`TIB_ebx-Ip_<version>_readme.txt` ファイルを参照してください。

注意:

各リリースでは、GA と対応するパックの可用性の間に若干の遅延があります。この場合、以前のパックを展開できます (ただし、すべてのメッセージに対応できない場合があります)。

EBX® ユーザーインターフェイスのローカリゼーションの拡張

EBX® は、互換性のある言語と地域へのユーザーインターフェイスのローカリゼーションをサポートしています。

注意:

現在、ラテン文字とキリル文字がサポートされています。他の文字セットを使用するロケールは使用できる場合がありますが、サポートされていません。

新しいロケールの追加

新しいロケールを追加するには、次の手順に従う必要があります。

- EBX® メイン構成ファイルで新しいロケールを宣言します。例：

```
ebx.locales.available=en-US, fr-FR, xx
```

- 最初のロケールは常にデフォルトと見なされます。
- ビルトインのロケール en-US および fr-FR は、必要に応じて削除できます。

EBX® ローカリゼーションの構成を参照してください。

- 次のファイルを jar ファイル (サンプル名：`ebx-lp_<version>_languagepack-xx.jar`) にパッケージ化します。
 - `com.orchestranetworks.i18n.frontEndFormattingPolicy_xx.xml` という名前のフォーマットポリシーファイル
 - リソースバンドル内のローカライズされたメッセージファイルのセット (`*_xx.mxml`)

注意:

ファイルは「.mxml」で終わる必要があります。

ファイルの完全なテンプレートセットは、[こちら](#)で入手できます。

- jar ファイルを EBX® クラスパスに展開します。このファイルには、`ebx.jar` クラスローダーからアクセスできます。

ローカライズされたリソースの解決

バージョン 5.7.0 以降、ローカライズされたリソースは、次のルックアップメカニズムを使用して、ロケール近接ベースで解決されます。

- `resourceName + "_" + language + "_" + country + "_" + variant + ".mxml"`
- `resourceName + "_" + language + "_" + country + ".mxml"`
- `resourceName + "_" + language + ".mxml"`
- `resourceName + ".mxml"`

注意:

解決は、ローカライズされたメッセージレベルで行われます。したがって、特定のローカリゼーションが必要なメッセージのみを含むロケールに対して 1 つ以上のファイルを定義することができます。

既知の制限事項

拡張不可能なマテリアル

以下のローカリゼーションは拡張できません。

- EBX® HTML エディターおよびビューアー

永続性

- [永続性の概要](#)
- [履歴](#)
- [レプリケーション](#)
- [データモデルの進化](#)

永続性の概要

この章は、履歴テーブルとレプリケートされたテーブルの概要です。

注意:

マップモードという用語は、そのまま保存されているため、データベース内のコンテンツに直接アクセスできるテーブルを指します。

- [管理対象マスターデータのプライマリ永続性](#)
- [履歴化](#)
- [レプリケーション](#)
- [マップモード](#)
 - [マップモードの概要](#)

- 構造上の制約
- マップモードによるデータモデルの制限

管理対象マスターデータのプライマリ永続性

EBX® リポジトリでモデル化および管理されるデータは、一般的なテーブル (すべてのデータセットとデータモデルに共通) を使用して、主にリレーショナルデータベースに保持されます。

履歴化

マスターデータテーブルは、複製されているかどうかに関係なく、データへの変更を追跡するために履歴をアクティブ化できます。

履歴自体はマップモードです。つまり、基になるデータベースで直接参照できます。

以下も参照してください。:

- [履歴](#)

レプリケーション

レプリケーションでは、リポジトリ内のデータのコピーをリレーショナルデータベース内のレプリカテーブルに作成することにより、マスターデータのテーブルへの直接 SQL アクセスが可能になります。履歴がアクティブ化されているかどうかに関係なく、任意のテーブルでレプリケーションを有効にできます。

レプリカテーブルは、主な目的が EBX® の外部の直接クエリにマスターデータにアクセスできるようにすることであるため、マップモードで永続化されます。

以下も参照してください。:

- [レプリケーション](#)

マップモード

マップモードの概要

マップモードとは、EBX® の外部でデータに直接アクセスできる形式で、テーブルが基になるリレーショナルデータベースに永続化される場合を指します。履歴テーブルとレプリカテーブルはすべて、マップモードのテーブルの例です。

マップモードのすべてのケースでは、必要な DDL ステートメントをバックグラウンドで自動的に実行することにより、必要に応じてデータベーススキーマ (データベーステーブル、インデックスなど) を自動的に変更します。このような手順は常にデータモデルのコンパイル時にトリガーされ、データモデルのコンパイルレポートは結果として生じるエラーを通知します。

マップモードに関するもう 1 つの一般的な考慮事項は、ほとんどの場合、データモデルエンティティが削除されても、対応するデータベースオブジェクトはすぐには削除されないということです。代わりに、無効としてマークされているため、後でオブジェクトを再度有効にする可能性があります。オブジェクトとそれに関連するデータおよびリソースをデータベースから確実に削除するには、パージのマークを付ける必要があります。削除は、次のグローバルパージ中に行われます。

以下も参照してください。:

- データベースマッピング管理
- データモデルの進化

構造上の制約

マップモードが設定されている場合、一部の EBX® データモデル制約は、基になる RDBMS スキーマに「構造制約」を生成します。これは、次の制約ファセットに関係します。

- `string` エレメントのファセット `xs:maxLength` および `xs:length`。
- `xs:decimal` エレメントのファセット `xs:totalDigits` および `xs:fractionDigits`。

データベースは、EBX® ほど寛容な検証モードをサポートしていません。したがって、上記の制約は **ブロッキング制約** になります。ブロッキング制約とは、更新が準拠していない場合に更新が拒否されることを意味します。トランザクションがブロッキング制約に準拠していない場合、トランザクションはキャンセルされ、`ConstraintViolationException` がスローされます。

以下も参照してください。:

- [ブロッキングと非ブロッキングの制約](#)

マップモードによるデータモデルの制限

基になるデータベースに直接保存するという性質上、マップモードで保存されるすべてのテーブルにはいくつかの制限が適用されます。

- [サポートされるデータベースの制限](#)
- 長さ無制限の文字列：外部キーを除く、`xs:string` 型、その派生型、および `xs:anyURI` のすべての文字列フィールドは、'maxLength' または 'length' ファセットを定義する必要があります。外部キーフィールドは、そのターゲットテーブルの最終主キーフィールドで構成されるため、このファセット要件は、外部キーフィールド自体ではなく、最終主キーフィールドのそれぞれに適用されます。さらに、VARCHAR や NVARCHAR2 などの文字型の最大長に関する、基になるデータベースの制限が適用されます。
- `type="osd:password"` 型のフィールドは無視されます。
- ターミナル複合型はサポートされていますが、レコードレベルでグローバルに `null` に設定することはできません。

より一般的には、マップモードのテーブルは、基になる RDBMS の制限に従います。たとえば、テーブル内の列の最大数が適用されます (Oracle の場合は 1000、PostgreSQL の場合は 1600)。履歴テーブルには、スキーマで宣言されたフィールドの 2 倍の数のフィールド (機能フィールド 1 つと、操作コード用に生成されたフィールド 1 つ) が含まれていることに注意してください。

既存のデータモデルに応じて、データモデルの進化は、基になる RDBMS によって制約される場合もあります。

以下も参照してください。:

- [データモデルの進化](#)

履歴

- [概要](#)
- [履歴の構成](#)

- リポジトリの履歴の構成
- データモデルでの履歴の構成
- 整合性
- 履歴ビューと権限
 - テーブル履歴ビュー
 - テーブル履歴の権限
 - トランザクション履歴ビュー
 - サイドバーの履歴サービス
- 履歴への SQL アクセス
 - アクセス制限
 - リレーショナルスキーマの概要
 - 生成された履歴テーブルの例
 - 操作の組み合わせ
 - 操作フィールドの値
- 履歴モードの影響と制限
 - 検証
 - 履歴テーブルのデータモデルの制限
 - 履歴モードのその他の制限

概要

履歴は、テーブル上のすべてのデータ変更 (レコードの作成、更新、削除) を追跡できる機能です。

これは、非推奨の [XML 監査証跡](#) を改良したものです。

以下も参照してください。:

- [履歴](#)
- [レプリケーション](#)

- データモデルの進化

履歴の構成

テーブルの履歴化を有効にするには、データモデル内のテーブルに履歴プロファイルを設定する必要があります。このセクションでは、履歴プロファイルと、それがテーブルに関連付けられる方法について説明します。

リポジトリの履歴の構成

履歴プロファイルは、履歴化がいつ作成されるかを指定します。履歴プロファイルを編集するには、>を選択します。

履歴プロファイルは名前で識別され、次の情報を定義します。

- 国際化されたラベル。
- 履歴が有効化されているデータスペース (ブランチ) のリスト。直接の子および/またはすべての子孫も対象とするかどうかを指定できます。

リポジトリのインストール時に、いくつかのプロファイルがすでに作成されています。これらのプロファイルは削除も変更もできません。

プロファイル ID	説明
<code>ebx-referenceBranch</code>	このプロファイルは参照データスペースでのみアクティブ化されます。
<code>ebx-allBranches</code>	このプロファイルはすべてのデータスペースでアクティブ化されます。
<code>ebx-instanceHeaders</code>	このプロファイルはデータセットヘッダーを履歴化します。ただし、内部データモデルではデータセットノードのみが定義されるため、このプロファイルは将来のバージョンでのみ設定されます。

データモデルでの履歴の構成

テーブル履歴の有効化

履歴は、データモデルアシスタントを使用するか、基になるデータモデルを編集することによって、テーブル上でアクティブ化できます。

データモデルを編集して履歴を有効にするには、`historyProfile` エlementを使用してテーブルに履歴プロファイルを宣言する必要があります。

```
<osd:table>

    <primaryKeys>/key</primaryKeys>

    <historyProfile>historyProfileForProducts</historyProfile>

</osd:table>
```

データモデルアシスタントを使用すると、リポジトリで定義されている履歴化プロファイルを表示できます。

履歴化は各テーブルごとに個別に有効にする必要があります。詳細については、[モデル設計](#) ドキュメントを参照してください。

特定のフィールドまたはグループの履歴の無効化

履歴化されたテーブルの場合、デフォルトの動作では、サポートされているすべてのElementが履歴化されます ([履歴モードの影響と制限](#) を参照)。

データモデルアシスタントを使用するか、基になるデータモデルを編集することにより、特定のフィールドまたはグループの履歴を無効にすることができます。

データモデルを編集してフィールドまたはグループの履歴を無効にするには、属性 `disable="true"` を持つElement `osd:history` を使用します。

```
<xs:element name="longDescription" type="xs:string">

    <xs:annotation>

        <xs:appinfo>

            <osd:history disable="true" />

        </xs:appinfo>

    </xs:annotation>

</xs:element>
```

```

        </xs:appinfo>

        </xs:annotation>

</xs:element>

```

データモデルアシスタントを使用してフィールドまたはグループの履歴を無効にするには、エレメントの **Advanced プロパティ** の **History** プロパティを使用します。

このプロパティがグループで定義されている場合、履歴はそのすべての子孫に対して再帰的に無効になります。グループが履歴を無効にすると、子孫の履歴を具体的に再度有効にすることはできません。

注意:

フィールドまたはグループを含むテーブルが履歴に記録されていない場合、このプロパティは効果がありません。

主キーフィールドの履歴を無効にすることはできません。

整合性

データモデルのコンパイル時に問題が検出された場合、警告メッセージまたはエラーメッセージがこのデータモデルに関連付けられた検証レポートに追加されます。さらに、エラーが検出されると、関連する各インスタンス (データセット) にアクセスできなくなります。最も一般的なエラーケースは次のとおりです。

- テーブルは、リポジトリで定義されていないプロファイルを参照しています。
- データモデルで参照されている履歴プロファイルは、現在のリポジトリ内の未定義または閉じたデータスペースに言及しています。

注意:

予期されたプロファイルがないリポジトリにデータモデルを展開するには、管理者がそれらを追加する必要があります。

履歴ビューと権限

テーブル履歴ビュー

データモデルのテーブルで履歴がアクティブ化されると、ユーザーインターフェイスのさまざまな場所 (レコード、レコードの選択、テーブル、データセット) から履歴ビューにアクセスできます。

次のセクションでは、アクセス許可がどのように解決されるかについて説明します。

詳細については、[テーブル履歴ビュー](#) セクションを参照してください。Java からテーブル履歴ビューにアクセスするには、メソッド `AdaptationTable.getHistory` を呼び出す必要があります。

テーブル履歴の権限

データ権限はデータ履歴にも適用されます。履歴権限は、データ権限と アクセス権の間で最も制限された権限として自動的に解決されます。

これは、ユーザー定義のアクセス許可ルールとプログラムによるアクセス許可ルールにも当てはまります。

プログラムルールを定義する際、機能的なデータセットコンテキストと履歴ビューコンテキストを区別する必要がある場合があります。これは、期待される権限が同じでない、または履歴構造に存在しないフィールドがあるためです。これには、データセットフィールド、計算値、および履歴が無効になっているフィールドがあります。次に、メソッド `Adaptation.isHistory` および `AdaptationTable.getHistory` をプログラムルールで使用して、履歴動作に関する特定の実装を行うことができます。

注意:

現在、テーブルにスクリプト化されたレコードのアクセス許可ルールが指定されている場合、制限があります。セキュリティ上の理由から、テーブル履歴へのアクセスはすべてのユーザーに対して完全に無効になっています。履歴へのアクセスは、将来のバージョンで許可される予定です。

トランザクション履歴ビュー

トランザクション履歴ビューを使用すると、テーブル、データセット、またはデータモデルに関係なく、実行されたトランザクションにユーザーインターフェイスから直接アクセスできます。

テーブルを表示するには、領域に移動し、ナビゲーションペインの下矢印メニューを使用してを選択します。トランザクション履歴には、履歴化されたデータスペースを選択し、ワークスペースのメニューを使用して、データスペース領域からアクセスすることもできます。

詳細については、[トランザクション履歴ビュー](#) を参照してください。

サイドバーの履歴サービス

サイドバーからレコードの履歴ビューにアクセスします。このビューには、次のリストが表示されます。

- トランザクション
- 変更されたフィールド値
- レコードを変更したユーザー
- タイムスタンプ

トランザクションを日付と時刻で並べ替えたり、フィールドでフィルタリングしたりできます。履歴表示は、多値リストタイプを含む、単純型と複合型の両方のデータタイプをサポートしています。

履歴への SQL アクセス

このセクションでは、SQL を使用して履歴データに直接アクセスする方法について説明します。

アクセス制限

データベーステーブルには、読み取り専用モードでのみアクセスする必要があります。[データベースアクセスとユーザー権限のルール](#) のセクションで指定されているように、TIBCO EBX® で使用されるデータベースユーザーを除いて、書き込みアクセスを禁止するのはデータベース管理者の責任です。

リレーショナルスキーマの概要

データベース内の履歴テーブルの説明は次のとおりです。

データベーススキーマには次のものが含まれます (次のセクションのダイアグラムも参照してください)。

共通および汎用テーブル

メインテーブルは `HV_TX` です。このテーブルの各レコードはトランザクションを表し

	ます。少なくとも 1 つの履歴テーブルを含むトランザクションのみが記録されます。 これらの共通テーブルにはすべて「HV」というプレフィックスが付いています。
特定の生成されたテーブル	履歴テーブルごとに、特定の履歴テーブルが生成されます。このテーブルには、テーブルのデータ変更の履歴が含まれています。 EBX® ユーザーインターフェイスでは、データベース内のこのテーブルの名前は、テーブルのドキュメントペイン (詳細モード) をクリックして取得できます。すべての特定の履歴テーブルには、「HG」というプレフィックスが付いています。

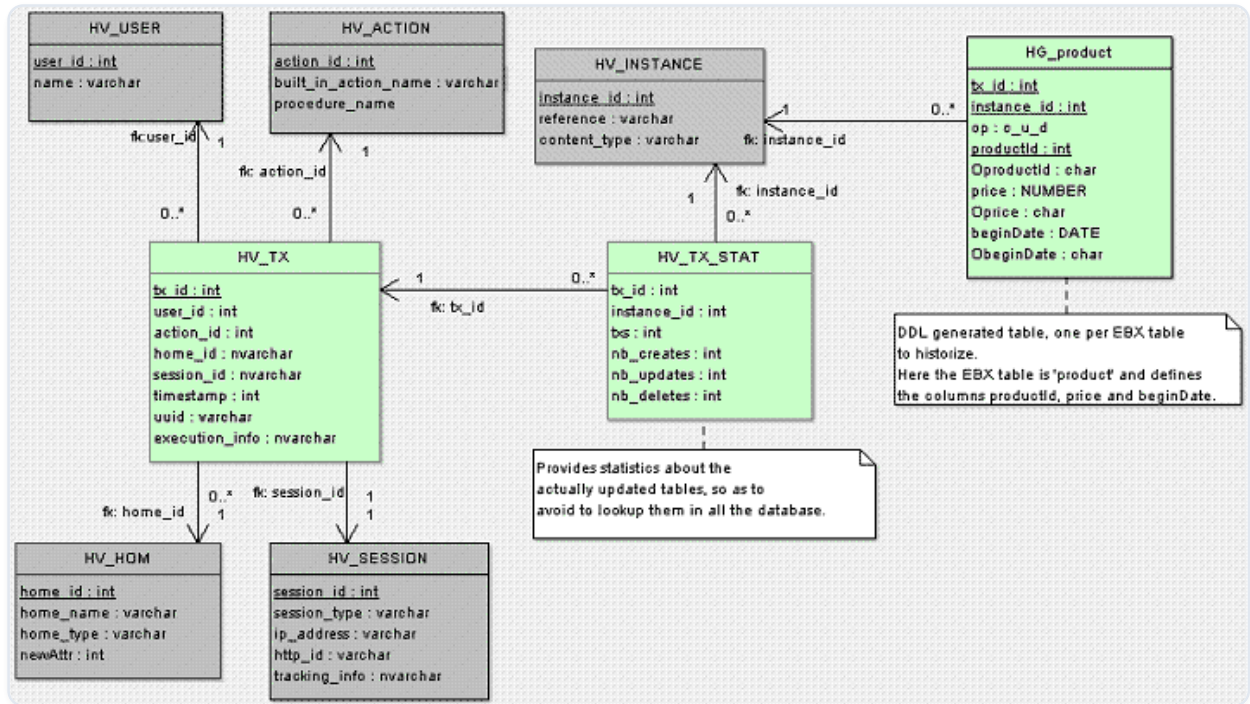
生成された履歴テーブルの例

次の例では、`product` というテーブルを履歴にしています。このテーブルが EBX® データモデルの 3 つのフィールドを宣言していると仮定します。

Product

- productId: int
- price: int
- beginDate: Date

次のダイアグラムは、結果のリレーショナルスキーマを示しています。



このテーブルで履歴をアクティブ化すると、上記の履歴スキーマ構造に示されている HG_product テーブルが生成されます。さまざまなフィールドの説明は次のとおりです。

- **tx_id** : トランザクション ID。
- **instance** : インスタンス ID。
- **op** : 操作タイプ - C (作成)、U (更新)、または D (削除)。
- **productId** : **productId** フィールド値。
- **OproductId** : **productId** の操作フィールド。次のセクションを参照してください。
- **price** : **price** フィールド値。
- **Oprice** : **price** の操作フィールド。次のセクションを参照してください。
- **beginDate** : **date** フィールド値。
- **ObeginDate** : **beginDate** の操作フィールド。次のセクションを参照してください。

操作の組み合わせ

同じトランザクションで複数の操作が組み合わせられた場合、操作フィールドは次のように解決されます。

- C + U -> C
- D + U -> D
- D + C -> U

C + D - > {} (履歴にエントリなし)

操作フィールドの値

機能フィールドごとに、文字 **0** が前に付いたフィールド名で構成される追加の操作フィールドが定義されます。このフィールドは、機能フィールドが変更されているかどうかを指定します。次のいずれかの値に設定されます。

- **null** : 機能フィールドの値が変更されていない場合 (およびその値が INHERIT ではない場合)。
- **M** : 機能フィールドの値が変更されている場合 (継承ではない)。
- **D** : レコードが削除された場合。

継承 が有効になっている場合、操作フィールドには次の 3 つの追加値を含めることができます。

- **T** : 機能フィールドの値が変更されておらず、その値が INHERIT の場合。
- **I** : 機能フィールドの値が INHERIT に設定されている場合。
- **0** : レコードが OCCULTING モードに設定されている場合。

履歴モードの影響と制限

履歴機能には、いくつかの影響と既知の制限があります。これらは、このセクションに一覧表示されます。履歴化モードを使用する場合は、これらの制限を注意深く読み、質問があれば Cloud Software Group, Inc. のサポートに連絡することを強くお勧めします。

検証

一部の EBX® データモデル制約は、テーブル履歴がアクティブ化されるとブロッキング制約になります。詳細については、[構造上の制約](#) のセクションを参照してください。

履歴テーブルのデータモデルの制限

履歴テーブルを含むデータモデルには、いくつかの制限が適用されます。

- [マップモードによるデータモデルの制限](#)
- 別の集約リストの下集約リストと、ターミナルグループの下集約リストの 2 種類の集約リストには制限があります。このような集約リストを含むデータモデルを使用できますが、これらのリストは無視されます (履歴は作成されません)。
- 計算された値は無視されます。

- リンクされたフィールドは無視されます。
- 履歴テーブルのユーザー定義属性により、データモデルのコンパイルエラーが発生します。

データモデルの進化は、関係するテーブルにすでに含まれているデータに応じて、基盤となる RDBMS によって制約される場合もあります。

以下も参照してください。:

- データモデルの進化

履歴モードのその他の制限

- 既存のデータを含むテーブルが履歴に対してアクティブ化されている場合、データのコピーは実行されません。
- データセットに対するグローバル操作は、履歴テーブルを宣言している場合でも、履歴化されません (インスタンスの作成と削除)。
- 非履歴フィールドを参照するデフォルトのラベルは、履歴テーブルではサポートされていません。

結果として、計算フィールドを参照するデフォルトのラベルは、履歴テーブルではサポートされていません。

回避策は、`UILabelRenderer` インターフェイスを実装し、ラベル計算を履歴に適合させることです。

- D3: 履歴はプライマリノードの配信データスペースで有効にできますが、レプリカノードの配信データスペースでは履歴機能は常に無効になります。
- 履歴に記録されたユーザー：特定の操作によっては、最後の操作を実行したユーザーと、対応する履歴レコードに記録されたユーザーが異なる場合があります。

これは、これらの操作が実際には前の状態でのデータステータスのレポートであるという事実によるものです。

- アーカイブのインポート：データスペースにアーカイブをインポートする場合、子データスペースで最後に実行された操作の時間とユーザーが保持されますが、履歴に記録されるユーザーはインポートを実行するユーザーです。

- プログラムによるマージ：データスペースでプログラムによるマージを実行すると、子データスペースで最後に実行された操作の時間とユーザーが保持されますが、履歴に記録されるユーザーはマージを実行するユーザーです。
- D3: 分散データ配信機能の場合、ブロードキャストが実行されると、プライマリノードからのデータがレプリカノードに報告され、子データスペースで実行された最後の操作の時間とユーザーが保持されますが、履歴に記録されるユーザーは「ebx-systemUser」となり、ブロードキャスト時のレプリカノードでのレポートの実行は、このユーザーによって行われます。

レプリケーション

- 概要
- レプリケーションの構成
 - レプリケーションの有効化
 - 特定のフィールドまたはグループでのレプリケーションの無効化
- SQL を使用したレプリカテーブルへのアクセス
 - データベース内のレプリカテーブルの検索
 - アクセス制限
 - SQL 読み取り
- 「オンデマンド」レプリケーション更新のリクエスト
- 複製の影響と制限
 - 検証
 - レプリケートされたテーブルのデータモデルの制限
 - データベース構成
 - 分散データ配信 (D3)
 - レプリケーションのその他の制限

概要

TIBCO EBX® リポジトリに保存されたデータは専用のリレーショナルテーブルにミラーリングできるため、SQL 要求やビューによるデータへの直接アクセスが可能になります。

履歴と同様に、このデータレプリケーションはエンドユーザーとクライアントアプリケーションに対して透過的です。特定のアクションにより、データベース内のレプリカへの自動変更がトリガーされます。

- モデルレベルでレプリケーションをアクティブ化すると、必要な DDL ステートメントが自動的に実行され、データベーススキーマが更新されます。
- 新しい列の作成など、複製されたテーブルに影響するデータモデルの進化も、DDL ステートメントを使用してデータベーススキーマを自動的に更新します。
- 「onCommit」更新モードを使用する場合：EBX® リポジトリ内のデータを更新すると、レプリカデータベーステーブルで関連する挿入、更新、および削除がトリガーされます。

以下も参照してください。:

- [履歴](#)
- [データモデルの進化](#)
- [リポジトリ管理](#)

注意:

複製テーブル：複製されたプライマリデータテーブルを指します。

レプリカテーブル (または レプリカ)：レプリケーションの対象となるデータベーステーブルを指します。

レプリケーションの構成

レプリケーションの有効化

データモデルで を定義するには、エレメント `annotation/appinfo` の下のエレメント `osd:replication` を使用します。各レプリケーションユニットは、特定のデータスペース内の単一のデータセット内のテーブルを指定します。

ネストされたエレメントは次のとおりです。

エレメント	説明	必須
<code>name</code>	レプリケーションユニットの名前。この名前は、現在のデータモデルのレプリケーションユニットを識別します。一意にする必要があります。	はい
<code>dataSpace</code>	このレプリケーションユニットに関連するデータスペースを指定します。スナップショットにすることはできません。	はい
<code>dataSet</code>	このレプリケーションユニットに関連するデータセットを指定します。	はい
<code>refresh</code>	データ同期ポリシーを指定します。可能なポリシーは次のとおりです。 • <code>onCommit</code>: データベース内のレプリカテーブルの内容は、ソーステーブルに対して常に最新です。EBX® ソーステーブルを更新するすべてのトランザクションは、レプリカテーブルで対応する挿入、更新、および削除ステートメントをトリガーします。 • <code>onDemand</code>: 指定されたテーブルのレプリケーションは、明示的な更新	はい

エレメント	説明	必須
	操作が実行された場合にのみ実行されます。 「onDemand」レプリケーション更新のリクエストを参照してください。	
<code>table/path</code>	データベースに複製される現在のデータモデル内のテーブルのパスを指定します。	はい
<code>table/nameInDatabase</code>	データがレプリケートされるデータベース内のテーブルの名前を指定します。この名前は、すべてのレプリケーションユニット間で一意にする必要があります。	はい
<code>table/element/path</code>	データベースに複製されるテーブル内の集計リストのパスを指定します。	はい
<code>table/element/ nameInDatabase</code>	集計リストのデータが複製されるデータベース内のテーブルの名前を指定します。この名前は、すべてのレプリケーションユニット間で一意にする必要があります。	はい

例

```
<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>
```

注意

- 複製されたテーブルのデータモデルの制限 を参照してください。
- データモデルのコンパイル時に、指定されたデータセットやデータスペースが現在のリポジトリに存在しない場合は警告が報告されますが、レプリカ テーブルはデータベースに作成されます。指定されたデータスペースとデータセットが作成されると、レプリケーションがアクティブになります。
- データモデルのコンパイル時に、テーブルレプリケーションが削除された場合、または上記のプロパティの一部が変更された場合、レプリカ テーブルはデータベースから削除され、必要に応じて新しい定義で再作成されます。

特定のフィールドまたはグループでのレプリケーションの無効化

複製されたテーブルの場合、デフォルトの動作では、サポートされているすべてのエレメントが複製されます (複製されたテーブルのデータモデルの制限 を参照)。

データモデルアシスタントを使用するか、基になるデータモデルを編集することで、特定のフィールドまたはグループのレプリケーションを無効にすることができます。

データモデルを編集してフィールドまたはグループのレプリケーションを無効にするには、属性 `disable="true"` を持つエレメント `osd:replication` を使用します。

```
<xs:element name="longDescription" type="xs:string">

  <xs:annotation>

    <xs:appinfo>

      <osd:replication disable="true" />

    </xs:appinfo>

  </xs:annotation>

</xs:element>
```

データモデルアシスタントを使用してフィールドまたはグループのレプリケーションを無効にするには、エレメントの **[詳細プロパティ]** にある **[レプリケーション]** プロパティを使用します。

このプロパティがグループに定義されている場合、その子孫すべてに対してレプリケーションが再帰的に無効になります。グループがレプリケーションを無効にすると、子孫に対してレプリケーションを明示的に再度有効にすることはできません。

注意:

フィールドまたはグループを含むテーブルが複製されていない場合、このプロパティは効果がありません。

主キーフィールドのレプリケーションを無効にすることはできません。

SQL を使用したレプリカテーブルへのアクセス

このセクションでは、SQL を使用してレプリカテーブルに直接アクセスする方法について説明します。

以下も参照してください。:

- [履歴への SQL アクセス](#)

データベース内のレプリカテーブルの検索

複製された EBX® テーブルごとに、対応するテーブルが RDBMS に生成されます。EBX® ユーザーインターフェイスを使用して、テーブルのドキュメントペインをクリックすると、このデータベーステーブルの名前を見つけることができます。RDBMS のテーブルには、次の追加の技術列が含まれています。

- **T_CREATOR_ID** : レコードを作成したユーザーの参照
- **T_LAST_USER_ID** : レコードを最後に更新したユーザーの参照
- **T_CREATION_DATE** : レコードの作成日
- **T_LAST_WRITE** : レコードの最終変更時刻

`T_CREATOR_ID` 列と `T_LAST_USER_ID` 列の最大長は 320 文字です。したがって、EBX® ディレクトリで定義されているユーザーの識別子は、この制限よりも短い最大長である必要があります。

アクセス制限

レプリカデータベーステーブルには、読み取り専用モードでのみ直接アクセスする必要があります。EBX® が使用するユーザーを除くすべてのデータベースユーザーへの書き込みアクセスをブロックするのは、データベース管理者の責任です。

以下も参照してください。:

- [データベースアクセスとユーザー権限のルール](#)

SQL 読み取り

適切に管理された、できれば短期間のトランザクションでは、直接 SQL 読み取りが可能です。ただし、このようなアクセスでは、EBX® 権限は考慮されません。そのため、読み取りを実行する権限が付与されたアプリケーションは、他の認証プロセスと権限を通じて信頼される必要があります。

「オンデマンド」レプリケーション更新のリクエスト

「onDemand」更新ポリシーでは、複製されたテーブルデータを更新するための明示的な要求が必要です。

レプリケーションの更新を要求する方法はいくつかあります。

- **ユーザーインターフェイス**: データセットアクションメニューで、グループ「レプリケーション」の下にあるアクション「レプリカの更新」を使用して、レプリケーション更新ウィザードを起動します。
- **データ サービス**: レプリケーション更新データサービス操作を使用します。詳細については、データサービスの [レプリケーションの更新](#) を参照してください。
- **Java API**: レプリケーションユニットの更新を開始するには、`ReplicationUnit` API の `ReplicationUnit.performRefresh` メソッドを呼び出します。

複製の影響と限界

レプリケーション機能には、以下に示すように、いくつかの既知の制限と副作用があります。レプリケーションを使用する場合は、このセクションを注意深く読み、質問があれば Cloud Software Group, Inc. のサポートに問い合わせることを強くお勧めします。

レプリケーションがサポートされているデータベースについては、[サポートされるデータベース](#)を参照してください。

検証

レプリケーションを有効にすると、一部の EBX® データモデル制約がブロック制約になります。詳細については、[構造上の制約](#)を参照してください。

複製されたテーブルのデータモデルの制限

複製されたテーブルを含むデータモデルには、いくつかの制限が適用されます。

- [マップモードによるデータモデルの制限](#)
- 指定されたデータセットがルートデータセットではない場合、またはまだ作成されていない場合、データセットの継承は「onCommit」更新ポリシーではサポートされません。詳細については、[データセットの継承](#)を参照してください。
- フィールド継承も、「onDemand」更新ポリシーでのみサポートされます。つまり、データモデルのコンパイル時に、更新モードが「onCommit」で、レプリケートされるテーブルに継承されたフィールドがある場合、エラーが報告されます。詳細については、[継承されたフィールド](#)を参照してください。
- 計算された値は無視されます。
- リンクされたフィールドは無視されます。
- 集約リストには、別の集約リストの下にある集約リストと、ターミナルグループの下にある集約リストの 2 種類があります。このような集約リストを含むデータモデルは使用できますが、これらのリストは無視されます (複製されません)。
- ユーザー定義属性はサポートされていません。レプリケーションユニットにユーザー定義属性が含まれている場合、コンパイルエラーが発生します。

データモデルの進化は、関係するテーブルにすでに含まれているデータに応じて、基盤となる RDBMS によって制約される場合もあります。

以下も参照してください。:

- データモデルの進化

データベース構成

更新操作は、前回の更新以降に変更された (作成と削除に関して) ソーステーブルの行のみを送信するように最適化されています。ただし、交換されるデータの量によっては、集中的な操作となり、大規模なトランザクションが必要になる場合があります。特に、最初の更新操作では、多数の行が関係する可能性があります。このようなトランザクションが最適な状態で実行されるように、データベースを適切に構成する必要があります。

例：Oracle の場合

- レプリケーションユニット内のすべてのレプリカ テーブルの大部分が 'UNDO' テーブルスペースに収まるようにする必要があります。
- 最小限のディスク アクセスでトランザクションを実行できるように、バッファークッシュに十分なスペースを確保することをお勧めします。
- これらのトランザクションが 'db_writer' プロセスを待機することを回避するために、十分な大きさの 'REDO' ロググループをプロビジョニングすることをお勧めします。

分散データ配信 (D3)

レプリケーションは、D3 プライマリおよびレプリカ配信データスペースの両方で利用できます。プライマリデータスペースでは、レプリケーションの動作は標準のセマンティックデータスペースと同じですが、レプリカデータスペースでは、レプリケートされたコンテンツは最後のブロードキャストスナップショットのコンテンツになります。

レプリカ配信データスペースでは、いくつかの制限が発生します。

- データモデルで定義された更新ポリシーは、上記の動作には影響しません。レプリケーションは常にスナップショットで実行されます。
- アクション項目 `レプリカの更新` は使用できません。
- `ReplicationUnit.performRefresh` メソッドを呼び出すことはできません。

以下も参照してください。:

- [D3 の概要](#)

レプリケーションのその他の制限

- サポートされるデータベースの制限
- 継承の場合、レプリカレコードフィールドは「継承値」フラグ (`AdaptationValue.INHERIT_VALUE`) を保持できません。このような場合は、継承された値のみが保持されます。より一般的には、継承状態と上書き状態を区別することはできません。

データモデルの進化

この章では、データモデルで可能な変更と、潜在的な制限について説明します。

注意:

データモデラーがテーブルの主キーに対して展開を実行するときは、いつでも結果の定義は新しいテーブルと見なされます。このような場合、既存のデータを何らかの方法で保存する必要がある場合は、新しいデータモデルを公開または展開する前に、データ移行計画を設定して運用する必要があります。また、データモデルの進化の直後にデータが破棄されないことにも注意してください。データモデルが前の状態にロールバックされると、前のデータが取得されます。

注意:

特定の型のデータモデルの展開は、ユーザーインターフェイスで直接実行できないため、データモデルをエクスポートし、XSD 形式で変更してから、再インポートする必要があります。構造だけでなく構成に影響を与えるデータモデルへの変更については、XSD をモジュールから TIBCO EBX® にインポートする必要があります。それ以外の場合、構成の変更は考慮されません。

以下も参照してください。:

- マップモード
- 許可される進化の種類
 - モデルレベルの進化

- テーブルレベルの進化
- フィールドレベルの進化
- 制限/制約
 - 主キーの進化に関連する制限
 - 外部キーの進化に関連する制限
 - フィールドレベルの進化に関連する制限

許可される進化の種類

このセクションでは、データモデルの作成後に可能な変更について説明します。

モデルレベルの進化

既存のデータモデルに次の変更を加えることができます。

- レプリケーションユニットをデータモデルに追加できます。更新ポリシーが「onCommit」の場合、対応するレプリカテーブルが作成され、次のスキーマコンパイル時に更新されます。
- レプリケーションユニットはデータモデルから削除できます。対応するレプリカテーブルはすぐに削除されます。
- データモデルは削除できます。レプリケーションユニットを宣言すると、対応するレプリカテーブルはすぐに削除されます。履歴テーブルが含まれている場合、この変更により、関連するマップされたテーブルが無効としてマークされます。関連するデータベースオブジェクトの実際の削除については、[データベースマッピング](#) を参照してください。

テーブルレベルの進化

テーブルレベルでデータモデルに次の変更を加えることができます。

- 新しいテーブルを追加できます。作成時に、テーブルは1つ以上のマップモードを宣言することもできます。
- 既存のテーブルを削除できます。レプリケーションユニットを宣言すると、対応するレプリカテーブルはすぐに削除されます。履歴がある場合、この変更により、マップされたテーブルが無効としてマークされます。関連するデータベースオブジェクトの実際の削除については、[データベースマッピング](#) を参照してください。
- 履歴は、テーブルで有効または無効にできます。履歴では、無効になっている間に実行された操作は考慮されません。

- テーブルの名前を変更できます。この変更は削除と作成の組み合わせと見なされるため、XML またはアーカイブファイルをエクスポートしてから再インポートすることにより、データを手動で移行する必要があります。

フィールドレベルの進化

フィールドレベルでデータモデルに次の変更を加えることができます。

- 新しいフィールドを追加できます。
- 既存のフィールドは削除できます。削除されたフィールドのデータは、次の更新時に各レコードから削除されます。レプリカテーブルの場合、対応する列は自動的に削除されます。履歴モードでは、フィールドは無効としてマークされます。
- 属性 `disable="true"` を使用して、フィールドを含むテーブルに適用される履歴またはレプリケーションからフィールドを明確に無効にすることができます。レプリカテーブルの場合、対応する列は自動的に削除されます。履歴テーブルの場合、列は残りますが、無効としてマークされます。[特定のフィールドまたはグループの履歴の無効化](#) および [特定のフィールドまたはグループでのレプリケーションの無効化](#) を参照してください。
- [制限/制約](#) にリストされているファセットを除いて、フィールドのファセットを変更できます。

次の変更は受け入れられますが、データが失われる可能性があります。これらの変更は削除と作成の組み合わせと見なされるため、XML またはアーカイブファイルをエクスポートしてから再インポートすることにより、データを手動で移行する必要があります。

- フィールドの名前を変更できます。
- フィールドの型は変更できます。

制限/制約

主キーの進化に関する制限

主キー定義が変更された場合

- すべてのデータセットとデータスペースで、テーブルの内容が空の内容にリセットされます。
- 新しい主キーが過去に使用されている場合、テーブルの内容は、すべてのデータセットとデータスペースで、この主キーが使用された時点で存在していた以前のデータにリセットされます。

- 主キーフィールドの型変換はサポートされていません。したがって、以前の主キー定義ですでに使用済みの型を再利用する場合でも、主キーの型を変更するとテーブルのコンテンツは常に空にリセットされます。
- 既存のデータスペースでテーブルの履歴がアクティブになっている場合、またはアクティブになっていた場合、変更は拒否されます。回避策として、まず専用テーブルに関連付けられている履歴テーブルを削除してから、主キーの変更に進みます。マップされたテーブルデータベースリソースを削除する手順については、[データベースマッピング](#)を参照してください。

注意:

変更された主キーが別のテーブルの主キーで参照されている場合、上記のすべての制限がターゲットテーブルに適用されます。

外部キーの進化に関する制限

- `osd:tableRef` ファセットの宣言が追加または変更された場合、またはそのターゲットテーブルの主キーが変更された場合、既存の値は空から再開されます (この変更が以前の定義に戻る場合を除きます。この場合は、以前のコンテンツが取得されます)。
- レプリケーションモードでは、外部キーフィールドの構造は、ターゲットのプライマリ キーの構造と一致するように設定されます。 `osd:tableRef` 制約を宣言する単一のフィールドは、ターゲットのプライマリキーの数とタイプに対応する数とタイプの列に分割される可能性があります。したがって、次の進化のケースは、マップされたテーブルの構造に影響を与えます。
 - テーブルフィールドに新しい `osd:tableRef` 制約を宣言します。
 - テーブルフィールドの既存の `osd:tableRef` 制約を削除します。
 - 既存の `osd:tableRef` 制約によって参照される主キーに列を追加 (または削除) します。
 - 既存の `osd:tableRef` 制約によって参照される主キーの任意の列の型またはパスを変更します。

これらの進化のケースでは、フィールドの削除と作成の組み合わせが発生します。したがって、既存のデータは手動で移行する必要があります。

フィールドレベルの進化に関する制限

フィールドの型を互換性のない型またはカーディナリティに変更すると、フィールドは新しいフィールドと見なされ、空のコンテンツで開始されます。モデルが以前の定義にロールバックされると、以前のコンテンツが取得されます。

- 次の型は完全に相互変換可能です (つまり、これらの型はまったく同じ永続表現を持ち、次のチャートで互いに置き換えることができます)。

- `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`

- 次の変換は完全にサポートされています (つまり、カーディナリティに関係なく)。

- `xs:decimal` から `xs:string`
- `xs:datetime` から `xs:string`
- `xs:date` から `xs:string`
- `xs:integer` から `xs:string`
- `xs:int` から `xs:decimal`
- `xs:integer` から `xs:decimal`
- `xs:decimal` から `xs:integer` (小数部は失われる)
- `xs:int` から `xs:integer`
- `xs:datetime` から `xs:date` (時間部分は失われる)
- `xs:date` から `xs:datetime` (時間部分はデフォルトで 0 に設定)

- 次の変換は、元の型が単一値の場合にのみ可能です。

- `xs:boolean` から `xs:string`

- `xs:time` から `xs:string`
- `xs:int` から `xs:string`
- `xs:long` から `xs:string`

型のカーディナリティは変更できます。変換がサポートされている場合、次の動作になります。

- 単一の要素を集約リストに変更すると、以前の単一の値が保持され、新しい集約リストに追加されます。
- 集約リストを単一の要素に変更すると、集約リストの最後の値のみが単一の要素に保持されます。その他の値は失われます。

注意:

グループと複合型は、他の型への変換 (および他の型からの変換) をサポートしていません。さらに、グループまたは複合型が単一オカレンスと複数オカレンスの間で変更される場合、変換はグループまたは複合型がターミナルである場合にのみサポートされます。

その他

- [継承と値の解決](#)
- [権限](#)
- [基準エディター](#)
- [検索](#)

継承と値の解決

- [概要](#)
 - [データセットの継承](#)
 - [継承されたフィールド](#)
 - [計算値 \(関数\)](#)
- [データセットの継承](#)
 - [データセット継承宣言](#)
 - [値ルックアップメカニズム](#)

- レコードルックアップメカニズム
- テーブルレベルでの継承動作の定義
- 継承されたフィールド
 - フィールド継承宣言
 - 値ルックアップメカニズム
 - 検証の無効化
- 最適化およびリファクタリングサービス
 - プロシージャの詳細
 - サービスの可用性

概要

継承の原則は、複数のコンテキストまたはエンティティによって共有されるリソースを相互化することです。TIBCO EBX® は、データ値を定義、因数分解、および解決するためのメカニズムを提供します。データセットの継承および 継承されたフィールド です。

さらに、値を計算するために 関数 を定義できます。

注意:

この章で説明する継承メカニズムを「構造的継承」と混同しないでください。これは通常、モデルに適用され、たとえば UML クラスダイアグラムで提案されます。

以下も参照してください。:

- 継承 (用語集)

データセットの継承

データセットの継承は、データが子会社やビジネスパートナーなどのグローバルなエンタープライズコンテキストに適用される場合に特に役立ちます。

データセットの階層が与えられると、共通データをルートまたは中間データセットに分解し、特定のコンテキストで特殊なデータを定義することができます。

データセットの継承メカニズムについては、以下の [データセットの継承](#) で詳しく説明しています。

継承されたフィールド

データセット間のグローバルなビルトイン関係を利用するデータセットの継承とは異なり、継承されたフィールドは、データ構造に固有のよりきめ細かい依存関係を利用します。これにより、ビジネスエンティティレベルでデータを因数分解および特殊化できます。

たとえば、モデルで「Product」が「FamilyOfProducts」に関連付けられていると指定されている場合、「Product」の一部の属性は、関連付けられている「FamilyOfProducts」で定義されている属性から値を継承する可能性があります。

注意:

同じデータセットで両方の継承メカニズムを使用する場合、フィールド継承はデータセット継承よりも優先されます。

以下も参照してください。:

- [継承フィールドのパフォーマンスに関する考慮事項](#)

計算値 (関数)

データモデルでは、ノードが *計算された値* を保持するように指定することもできます。この場合、値が要求されるたびに、指定された Java Bean 関数が実行されます。

この関数は、現在のレコードの値や別のテーブルに基づく計算などの現在のコンテキストを考慮し、サードパーティのシステムにリクエストを送信できます。

以下も参照してください。:

- [計算値](#)

データセットの継承

データセット継承宣言

データセット継承メカニズムは、DMA からデータモデル (>) で宣言するか、XML スキーマ定義で直接宣言します (>)。

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ebxnd="urn:ebx-schemas:binding_1.0">

  <xs:annotation>

    <xs:appinfo>

      <osd:inheritance>

        <dataSetInheritance>all</dataSetInheritance>

      </osd:inheritance>

    </xs:appinfo>

  </xs:annotation>

  ...

</xs:schema>
```

エレメント `osd:inheritance` は、このデータモデルに基づくデータセットでの継承の使用を指定するためのプロパティ `dataSetInheritance` を定義します。次の値を指定できます。

- `all` は、データモデルに基づいてすべてのデータセットに対して継承が有効になっていることを示します。
- `none` は、データモデルに基づくすべてのデータセットの継承が無効になっていることを示します。

指定しない場合、継承メカニズムは無効になります。

値ルックアップメカニズム

値のデータセット継承ルックアップメカニズムは次のように進行します。

1. 値がローカルで定義されている場合は、値が返されます。

明示的に `null` にすることもできます。

2. それ以外の場合は、データセットの階層内のデータセットのビルトインの子と親の関係に従って、最初のローカルで定義された値を検索します。
3. ローカルで定義された値が見つからない場合は、デフォルト値が返されます。

デフォルト値が定義されていない場合、`null` が返されます。

****注意：****デフォルト値は次の場所では定義できません。

- 単一の主キーノード
- 自動インクリメントノード
- 計算値を定義するノード

レコードロックアップメカニズム

値と同様に、テーブルレコードも複数のコンテキストによってユニットとして継承できますが、特定のコンテキストに対して部分的に再定義 (上書き)、特定のコンテキストに対して定義 (ルートモード)、または 非表示にすることができます。

正式には、テーブルレコードには 4 つの異なる定義モードのいずれかがあります。

ルートレコード	テーブルでローカルに定義されており、親はありません。これは、同じ主キーを持つレコードが親テーブルに存在しないか、この親が非表示レコードであることを意味します。
レコードの上書き	テーブルでローカルに定義され、親レコードがあります。これは、同じ主キーを持つレコードが親テーブルに存在し、この親が非表示レコードではないことを意味します。上書きレコードは、明示的に再定義する値を除いて、その値を親から継承します。
継承されたレコード	現在のテーブルでローカルに定義されておらず、親レコードがあります。すべての値が継承されます。

	関数は常に現在のレコードのコンテキストで解決され、継承されません。
非表示レコード	同じ主キーを持つ親が定義されている場合、この親がテーブルの子孫に表示されないことを指定します。

以下も参照してください。:

- データセットの継承

テーブルレベルでの継承動作の定義

データモデルのテーブルの宣言で管理ルールを指定することもできます。

以下も参照してください。:

- データセットの継承に関連するプロパティ

継承されたフィールド

特定の継承メカニズムにより、他のテーブルとの関係に従ってフィールドの値をフェッチできます。

フィールド継承宣言

次のように、特定の継承を指定する必要があります。

- 基礎となるデータモデルのノードの高度なプロパティ。
- 基礎となるデータモデルのターミナルノード。次のように宣言されます。

```
<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>

```

エレメント `sourceRecord` は、値の継承元のレコードを検索する方法を説明する式です。これは、現在のエレメントからソーステーブルへの外部キーまたは外部キーのシーケンスです。

`sourceRecord` がデータモデルで定義されていない場合、継承されたフィールドは現在のレコードからフェッチされます。

エレメント `sourceNode` は、ソースレコードで継承するノードのパスです。

特定の継承については、次の条件を満たす必要があります。

- エレメント `sourceNode` は必須です。
- ソースレコードへのパスの式は、現在のエレメントからソースレコードへの外部キーの一貫したパスにする必要があります。この式には、1 対 1 および 0 対 1 の関係のみが含まれている必要があります。
- `sourceRecord` には、集約されたリストエレメントを含めることはできません。
- `sourceRecord` の各エレメントは外部キーにする必要があります。
- 継承されたフィールドが外部キーでもある場合、`sourceRecord` はそれ自体を参照して、継承された値のソースレコードへのパスを取得できません。
- `sourceRecord` のすべてのエレメントが存在する必要があります。
- ソースノードは、ソースレコードを含むテーブルに属している必要があります。
- ソースノードはターミナルにする必要があります。

- 。ソースノードは書き込み可能にする必要があります。
- 。ソースノードタイプは、現在のノードタイプと互換性がある必要があります。
- 。ソースノードのカーディナリティは、現在のノードのカーディナリティと互換性がある必要があります。
- 。継承元のフィールドが同じレコードにフェッチされている場合、ソースノードを継承フィールドと同じにすることはできません。

値ルックアップメカニズム

継承されたフィールド値のルックアップメカニズムは、次のように進行します。

1. 値がローカルで定義されている場合は、値が返されます。

明示的に `null` にすることができます。

2. それ以外の場合は、データモデルで定義されているプロパティに従って、継承元のソースレコードと値を検索します。
3. このプロセスは再帰的です。ソースノードがローカルで値を定義していない場合は、ソースノードの継承動作に従って値が検索されます。

検証の無効化

場合によっては、パフォーマンス上の理由から継承されたフィールドの検証を無効にすると便利ことがあります。データモデルで指定されたプロパティ `osd:disableValidation= "true"` を使用すると、継承されたフィールドの検証を無効にできます (以下の例を参照)。

例:

```
<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>

```

最適化およびリファクタリングサービス

EBX® は、データセットの階層におけるデータセットの継承を最適化するためのビルトインのユーザーサービスを提供します。このサービスは、次の機能を実行します。

- ****重複値の処理**：**継承された値の重複であるすべてのパラメーター値を検出して削除します。
- ****共通値の相互化**：**共通の祖先の子孫の間で共通の値を検出し、相互化します。

プロシージャの詳細

データセットはボトムアップで処理されます。つまり、レベル N のデータセットでサービスを実行し、 $N+1$ がその子のレベル、 $N+2$ がその子の子のレベルである場合、サービスはまずレベル $N+2$ のデータセットを処理して、レベル $N+1$ のデータセットに対して最適化できるかどうかを判断します。次に、レベル $N+1$ の最適化をレベル N に対して行います。

注意:

- これらの最適化およびリファクタリング関数は、データモデルで宣言されているデフォルト値を処理しません。
- 最適化手順で考慮される最高レベルは、常にサービスが実行されるデータセットです。これは、ターゲットデータセットとそれ自体の祖先の間で最適化とリファクタリングが実行されないことを意味します。
- テーブルの最適化は、同じ主キーを持つレコードに対して実行されます。
- 継承されたフィールドは最適化されていません。

- 最適化およびリファクタリング機能は、データセットがアクティブ化されている場合、データセットの解決されたビューを変更しません。

サービスの可用性

「最適化とリファクタリング」サービスは、子データセットがあり、データセット情報で「アクティブ化」プロパティが「いいえ」に設定されているデータセットで使用できます。

このサービスは、現在のデータセット値への書き込みアクセス権を持つすべてのプロファイルで利用できます。プロファイルに制限付きアクセス権を設定することで無効にできます。

注意:

パフォーマンス上の理由から、アクセス権はすべてのノードおよびテーブルレコードで検証されるわけではありません。

権限

権限は、各ユーザーがデータとアクションにアクセスできるかどうかを決定します。

- 概要
 - ユーザー、ロール、プロファイル
 - 権限ルール
 - 権限の解決
 - 所有者と管理者の特別な権限
 - 権限に対するマージの影響
- 権限に関する重要な考慮事項
 - 高い権限を付与するアクションとユーザーサービス
 - 権限チェックなしの API アクセス
 - UI で情報を非表示にするための権限の使用
 - クエリ API での権限チェックの制限事項
 - レコードとテーブル履歴に関するスクリプト化された権限ルール
 - カスタム表示ラベルでの非表示フィールドの使用

- リンクフィールド権限チェック
- テーブルアクション権限に関連する制限事項
- 権限キャッシュのライフサイクル
- 致命的なエラーが発生したデータセットに対する権限
- 機密性の定義
- ユーザー定義ルールの定義
 - データスペースのユーザー定義ルールの定義
 - データセットのユーザー定義ルールの定義
 - データセット権限構成サービスの使用
- 動的ルールの定義
 - データに対するスクリプト化された権限ルールの定義
 - データに対するアクセスルールの定義
 - サービスのアクティベーションルールの定義
 - サービスの権限ルールの定義
- データ権限の解決
 - ユーザー定義ルールの解決
 - 動的ルールの解決
- サービスの権限の解決
 - ユーザー定義ルールの解決
- アクションの権限の解決
 - ユーザー定義ルールの解決

概要

権限は、アクションが許可されているかどうかに関連しています。これらは、アクセス権限、つまり、エンティティが非表示、読み取り、または読み取り/書き込みのいずれであるかにも関連しています。権限によって制御される主なエンティティは次のとおりです。

- データスペース
- データセット

- テーブル
- グループ
- フィールド

ユーザー、ロール、プロファイル

権限の定義と解決には、ユーザーまたはロールに適用される一般的な用語である *プロファイル* の概念が広く使用されています。

各ユーザーは複数のロールに参加でき、ロールは複数のユーザーで共有できます。

これらの関係は、ユーザーおよびロールディレクトリで定義されます。[ユーザーとロールのディレクトリ](#)を参照してください。

特別な定義:

- *ビルトインの管理者*は、ビルトインのロール「ADMINISTRATOR (管理者)」のメンバーです。
- *データセットの所有者*は、ルートデータセットの情報で指定された所有者属性のメンバーです。この場合、データセットのコンテキストで権限が解決されると、ビルトインのロール「OWNER (所有者)」がアクティブになります。
- *データスペースの所有者*は、データスペースに指定された所有者属性のメンバーです。この場合、データスペースのコンテキストで権限が解決されると、ビルトインのロール「OWNER (所有者)」がアクティブになります。

権限ルール

権限ルールは、特定のエンティティのプロファイルに付与される認証を定義します。

ユーザー定義の権限ルールは、ユーザーインターフェイスを介して作成されます。[ユーザー定義ルールの定義](#)セクションを参照してください。

動的権限ルールは、開発者が作成したプログラムルール、または管理者が作成したスクリプトルールのいずれかです。[動的ルールの定義](#)セクションを参照してください。

権限の解決

権限は常に認証されたユーザーセッションのコンテキストで解決されるため、権限は主にユーザープロファイルに基づいています。

一般に、権限の解決は、特定のレベルとその親レベルの間で制限的に実行されます。したがって、どのレベルでも、ユーザーは親レベルで解決された権限よりも高い権限を持つことはできません。

動的な権限は常に制限的であると見なされます。

注意:

Java API では、クラス `SessionPermissions` が解決された権限へのアクセスを提供します。

以下も参照してください。:

- データ権限の解決
- サービスの権限の解決
- アクションの権限の解決

所有者と管理者の特別な権限

データセット上

ビルトインの管理者またはデータセットの所有者は、次のアクションを実行できます。

- 権限を管理
- データセットがルートデータセットの場合は、所有者を変更
- 一般情報 (ローカライズされたラベルと説明) を変更

注意:

権限の定義により、ビルトインの管理者またはデータセット所有者がデータを表示したり特定のアクションを実行したりする権限を制限できますが、権限管理へのアクセス権限は常にあるため、自分のアクセス権限を変更することは可能です。

データスペース上

データスペースのスーパーオーナーになるには、ユーザーは次のいずれかを行う必要があります。

- データスペースを所有し、その権限の管理を許可する。

- 現在のデータスペースの祖先であるデータスペースを所有し、その祖先データスペースの権限を管理できるようにする。

ビルトインの管理者またはデータスペースのスーパーオーナーは、次のアクションを実行できます。

- データスペースの権限を管理
- 所有者を変更
- ロックまたはロック解除
- 一般情報 (ローカライズされたラベルと説明) を変更

さらに、ワークフローでは、「データスペースの作成」または「スナップショットの作成」ビルトインスクリプトタスクを使用する場合、解決された権限は、現在のセッションではなく、スクリプトタスクの構成で定義された所有者を使用して計算されます。これは、これらの場合、現在のセッションがシステムユーザーに関連付けられているためです。

注意:

権限の定義により、ビルトインの管理者またはデータスペース所有者がデータを表示したり特定のアクションを実行したりする権限を制限できますが、権限管理へのアクセス権限は常にあるため、自分のアクセス権限を変更することは可能です。

権限に対するマージの影響

データスペースがマージされると、ユーザーがマージプロセス中に指定した場合にのみ、子データセットの権限が親データセットの権限とマージされます。親データスペースの権限が影響を受けることはありません。

マージを実行しようとしているプロファイルに対して一部のエレメントが非表示になっている場合、データに対するマージの影響が完全に表示されないため、続行できません。

権限に関する重要な考慮事項

このセクションには、権限を操作する際に留意する必要があるいくつかの非常に重要な情報がリストされています。

高い権限を付与するアクションとユーザーサービス

次のアクションとそれに関連するユーザーサービスは、信頼できる管理者にのみ許可する必要があります。

- 「データスペースの作成」 (データスペースの権限を定義する権限を付与する「所有者」ロールを付与します)
- 「データセットの作成」 (データセットの権限を定義する権限を付与する「所有者」ロールを付与します)
- 「アーカイブのインポート」 (権限に関係なくアーカイブコンテンツの書き込みを許可します)

注意:

これらのプロファイルに付与される権限の詳細については、[所有者と管理者の特別な権限セクション](#)を参照してください。

権限チェックなしの API アクセス

開発者と管理者は、API の一部が権限チェックなしで実行できることに注意する必要があります。一般に、コードが `セッション` が指定されたコンテキストで実行される場合、権限がチェックされることを意味します。権限がチェックされない特定のケースを次に示します。

- Java プロシージャが、`ProcedureContext.setAllPrivileges` を使用してすべての権限チェックを無効にする場合。
- データベースに直接クエリを実行して EBX® データにアクセスする場合、テーブルで [レプリケーションモード](#) または [履歴](#) が有効になっています。これは、EBX® 権限が基礎となるデータベースで「変換」されないためです。結果として、データベースへのアクセスをグローバルに制限するか、データベースに適切な権限を定義する必要があります。

UI で情報を非表示にするための権限の使用

一部の非機密情報を UI で非表示にするためだけに権限を使用することはお勧めできません。特に、この情報が一部のクエリでのフィルタリング、結合、並べ替えに使用される可能性がある場合には注意が必要です。このような場合は、代わりに UI のみの非表示メソッドを使用する必要があります。たとえば、データモデルのプロパティでフィールドを [デフォルトビュー](#) では非表示として設定するか、関係するユーザー用の [ビュー](#) を作成します。

上記の提案を適用できず、権限を使用して非機密情報を非表示にする必要がある場合は、モデルでノードを非機密として設定する必要があります。詳細については、[機密性の定義](#)を参照してください。

クエリ API での権限チェックの制限事項

クエリ または リクエスト でセッションを指定するときに実行される権限チェックは、クエリで使用されている機密フィールドが現在のユーザーに対して非表示になっている場合、`QueryPermissionException` をスローします。以下を除くすべてのフィールドは機密情報と見なされます。

- 主キーに属するフィールド
- 非機密として明示的に定義されたフィールド。詳細については、[機密性の定義](#)を参照してください。

注意:

レコード依存の `AccessRule` を持つ機密ノードの機密性は、部分的にのみ適用されます。実際に、`AccessRule.getPermission` メソッドは、権限チェックの実行中に `aDataSetOrRecord` パラメーターとしてデータセットのみで呼び出され、レコードで呼び出されることはありません。結果として、ルールのレコード依存ロジックは、このチェック中には適用されません。

レコードとテーブル履歴に関するスクリプト化された権限ルール

スクリプト化された権限ルールを使用して、テーブルレコードとテーブル履歴をフィルタリングできます。デフォルトでは、テーブル履歴へのアクセスはすべてのユーザーに対して無効になっています。この動作はセキュリティの強化に役立ちます。ただし、権限ルールを編集し、[履歴へのアクセスを許可するプロファイルリスト](#)に目的のプロファイルを追加することで、テーブル履歴へのアクセスを許可されたプロファイルのリストを作成できます。

カスタム表示ラベルでの非表示フィールドの使用

テーブル (「defaultLabel」プロパティ) および関係 (「display」プロパティ) のカスタム表示ラベルの解決では、権限が考慮されます。ラベルで非表示フィールドが検出されるとすぐに、代わりに主キーが表示されます。

注意:

これは、`TableRefDisplay` や `Adaptation.getLabelOrName` などの API を使用する場合には当てはまりません。提供されたコンテキストには現在のセッションが含まれていない

ため、権限チェックを実行できません。結果として、開発者はこれらの API を使用するとき機密データが公開されていないことを確認する必要があります。

注意:

また、クイック検索では、履歴ビューのコンテキストや子データセットのカスタム表示ラベルに非表示のフィールドがあるノードが無視されることに注意してください。

この動作のため、クエリでのフィルタリングにラベルを使用することは推奨されません。非表示フィールドのあるラベルを使用すると、pk 値に置き換えられ、フィルターの一貫性が失われます。

リンクフィールド権限チェック

リンクされたフィールドアクセス権限が計算されると、結果は、メインテーブルのノードに適用される権限とターゲットテーブルのノードの間の最小値になります。実際には、フィールドがテーブルで非表示になっている場合、他のテーブルでそのフィールドを指しているすべてのリンクされたフィールドも非表示になることを意味します。

テーブルアクション権限に関連する制限事項

プロシージャ内のテーブルに対してアクション (作成、削除、上書き、または非表示) を実行する場合、権限の解決中に、テーブルノードに対する現在のユーザーセッションアクセス権限は無視されます。このチェックを実行する場合、クライアントコードは、プロシージャの前に

`SessionPermissions.getNodeAccessPermission` を明示的に呼び出す必要があります。

権限キャッシュのライフサイクル

データサービスとユーザーサービスの両方の権限の解決を最適化するために、セッションレベルで専用キャッシュが実装されています。動的ルールを含むすべての権限がキャッシュされます。これは、以下で説明するキャッシュの期間中、ルールの結果が変更されないことを意味します。

セッションキャッシュのライフサイクルは、以下で説明するように、コンテキストによって異なります。

- UI では、ajax 以外のイベント (ページ表示、ポップアップの開始など) ごとにキャッシュがクリアされます。

- プログラムプロシージャでは、明示的にクリアされない限り、キャッシュはプロシージャの最後まで存続します (以下を参照)。

注意:

EBX® アーカイブをインポートしたり、プログラムでデータスペースをマージしたりして、プロシージャのコンテキストで権限を変更する場合は、`Session.clearCache` を呼び出してセッションキャッシュをクリアする必要があります。そうしないと、プロシージャが終了するまで変更が反映されません。

致命的なエラーが発生したデータセットに対する権限

データセットに致命的なエラーがある場合、所有者と管理者のみがアクセス権を持ち、次のアクションのみが使用可能になります。

- 検証レポートの表示
- 情報ページの表示
- データセットの削除

機密性の定義

ノードの機密性は、特定のユーザーに対してノードが非表示になっているときに `クエリ` または `リクエスト` で使用できるかどうかを定義します。デフォルトでは、すべてのノードは機密として扱われます。

ノードで機密性を定義すると、それを使用する関係およびビューに次の影響があります。

- 関係 (関連付けまたはテーブル参照) は、その定義ノードの少なくとも 1 つが機密で非表示の場合、非表示になります。
- 追加の XPath フィルターで使用されるノードの少なくとも 1 つが機密で非表示の場合、関係は非表示になります。
- フィルター条件または並べ替え条件で使用されているノードの少なくとも 1 つが機密で非表示の場合、ビューは非表示になります。

ノードと関係を定義する追加の XPath フィルターの詳細については、[外部キー](#)および[関連付け](#)を参照してください。

注意:

フィールドを非機密として設定する前に、十分に注意することをお勧めします。これは、非公開であっても、悪意のあるユーザーがこのノードを使用してフィルター処理または並べ替えを行うことで、値を「推測」する可能性があるためです。常にできるわけではありませんが、UI で情報を非表示にするための権限の使用で推奨されているように、一般に、ビューを使用してこの種のユースケースを処理する方が優れたオプションです。

ノードの機密性は DMA で構成できます。詳細については、[ノードの機密性](#)を参照してください。

ユーザー定義ルール of 定義

各レベルには同様のスキーマがあり、プロファイルの権限ルールを定義できます。

データスペースのユーザー定義ルール of 定義

特定のデータスペースについて、各プロファイルに許可される権限は次のとおりです。

データスペースへのアクセス	AUTHORIZATION
書き込み	- データスペースを表示できます。 - データセットの権限に従ってデータセットにアクセスできます。
読み取り専用	- データスペースとそのスナップショットを表示できます。 - 権限で許可されている場合は、子データスペースを表示できます。 - データスペースの内容を表示できますが、変更することはできません。
非表示	データスペースもそのスナップショットも表示できません。

データスペースへのアクセス	AUTHORIZATION
	- 子データスペースの表示が許可されている場合、現在のデータスペースを表示できますが、選択することはできません。 - データセットを含むデータスペースの内容にアクセスできません。 - データスペースに対してアクションを実行できません。
制限ポリシー	このデータスペースプロファイルと権限の関連付けを他の権限ルールよりも優先する必要があるかどうかを示します。
子データスペースの作成	プロファイルが現在のデータスペースから子データスペースを作成できるかどうかを示します。
子スナップショットの作成	プロファイルが現在のデータスペースのスナップショットを作成できるかどうかを示します。
マージの開始	プロファイルが現在のデータスペースをその親データスペースとマージできるかどうかを示します。
アーカイブのエクスポート	プロファイルが現在のデータスペースをアーカイブとしてエクスポートできるかどうかを示します。
アーカイブのインポート	プロファイルがアーカイブを現在のデータスペースにインポートできるかどうかを示します。
データスペースを閉じる	プロファイルが現在のデータスペースを閉じることができるかどうかを示します。

スナップショットを閉じる	プロファイルが現在のデータスペースのスナップショットを閉じることができるかどうかを示します。
サービスの権限	プロファイルにデータスペースでサービスを実行する権限があるかどうかを示します。デフォルトでは、すべてのデータスペースサービスが許可されています。現在のデータスペースのビルトイン管理者またはスーパーオーナー、または現在のデータスペースの権限の変更を許可されている特定のユーザーは、これらの権限を変更して、特定のプロファイルのデータスペースサービスを制限できます。
作成時の子データスペースの権限	ユーザーが子データスペースを作成すると、この新しいデータスペースの権限は、親データスペースの「作成時の子データスペースの許可」で定義された権限に基づいて、プロファイルの所有者に自動的に割り当てられます。異なるロールを通じて所有者に複数の権限が定義されている場合、所有者のプロファイルは他のプロファイルと同じように動作し、通常どおりに権限が解決されます See "権限の解決" on page 464.

データセットのユーザー定義ルールの定義

特定のデータセットについて、各プロファイルに許可される権限は次のとおりです。

データセットに対するアクション

制限ポリシー	このデータセットプロファイルと権限の関連付けを他の権限ルールよりも優先する必要があるかどうかを示します。
--------	--

子データセットの作成	プロファイルに現在のデータセットの子データセットを作成する権限があるかどうかを示します。
データセットの複製	プロファイルに現在のデータセットを複製する権限があるかどうかを示します。
データセットの親の変更	プロファイルに、特定の子データセットの親データセットを変更する権限があるかどうかを示します。

テーブルに対するアクション

デフォルトテーブルのアクション権限は、データセットレベルで定義されます。その後、1 つ以上のテーブルに対するこれらのデフォルトの権限をオーバーライドすることができます。各プロファイルに許可される権限は次のとおりです。

新しいレコードの作成	プロファイルにテーブルにレコードを作成する権限があるかどうかを示します。
継承されたレコードを上書きする	プロファイルに、テーブル内の継承されたレコードを上書きする権限があるかどうかを示します。
継承されたレコードを非表示にする	プロファイルに、テーブル内の継承されたレコードを非表示にする権限があるかどうかを示します。
レコードの削除	プロファイルにテーブル内のレコードを削除する権限があるかどうかを示します。

ノード値のアクセス権限

特定のターミナルノードで定義された権限は、デフォルトのアクセス権限をオーバーライドします。

読み書き	ノード値を表示および変更できます。
------	-------------------

読み取り	ノードを表示できますが、それらの値を変更することはできません。
非表示	ノードを表示できません。

サービスの権限

ビルトインの管理者または現在のデータスペースの所有者は、サービスのデフォルトの権限を変更して、特定のプロファイルへのアクセスを制限または許可できます。

有効	現在のプロファイルへのサービスアクセスを許可します。
無効	現在のプロファイルへのサービスアクセスを禁止します。メニューには表示されず、Web コンポーネントを介して起動することもできません。
デフォルト	サービス宣言時に定義されたデフォルトの権限に従って、サービス権限を有効または無効に設定します。 詳細については、 <code>ActivationContext.setDefaultPermission</code> を参照してください。

データセット権限構成サービスの使用

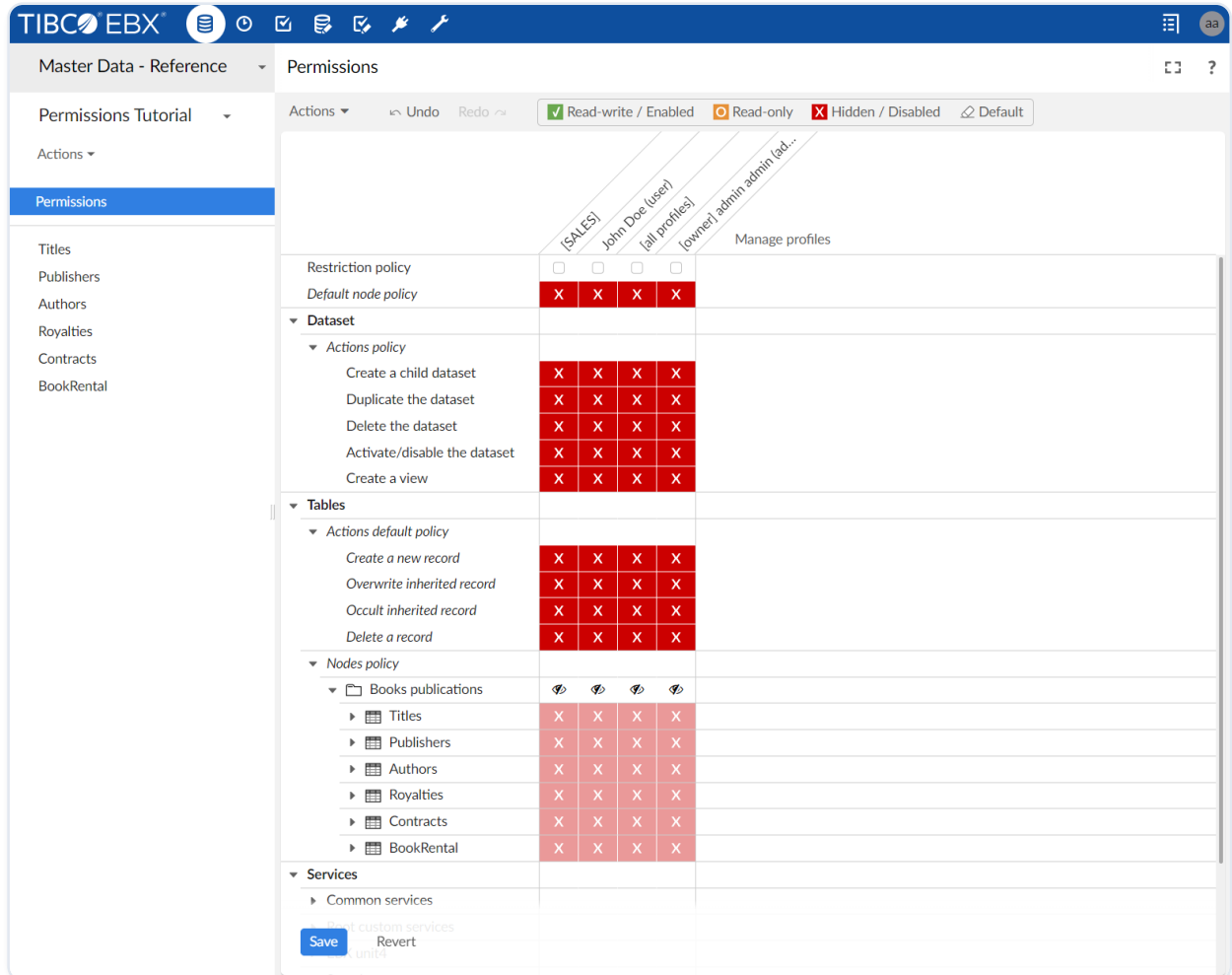
データセットレベルでのユーザー定義のアクセスルールは、データセットのメニューから利用できるサービスを使用して定義されます。このサービスから、管理者はこのデータセット内のデータ、アクション、サービスへのユーザーアクセスを確認および構成できます。

権限サービス UI の概要

サービスは次のようなグリッドとして表示されます。

- 各列は、データセット内の 1 つのプロファイルのルールを表します。以下のスクリーンショットの例に示すように、プロファイルは、SALES、John Doe (ユーザー)、および admin admin です。[プロファイルの管理](#)を参照してください。

- 各行は、ルールが適用されるノード、アクション、またはサービスを表します。[権限の管理](#)を参照してください。
- 変更可能な各セルは、特定のプロファイルのノード、アクション、またはサービスにアクセスするための権限レベルを表します。



プロファイルの管理

プロファイル管理タスクには次のものが含まれます。

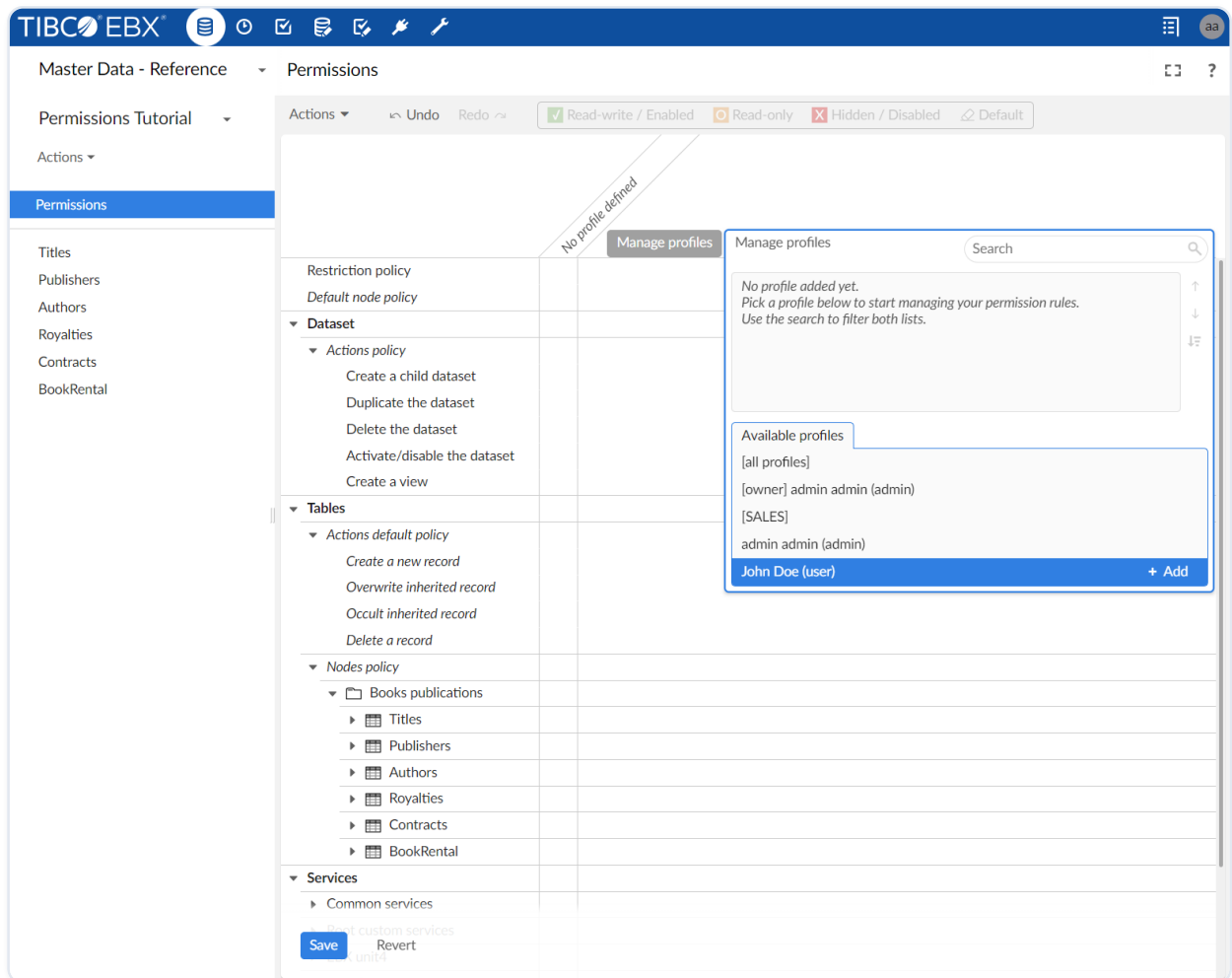
- [ユーザー定義のアクセスルールの追加](#)
- [プロファイル構成の別プロファイルへの複製](#)
- [プロファイル構成の削除](#)

ユーザー定義のアクセスルールの追加

プロファイルのルールを追加するには、次の手順を実行します。

- グリッドの上部にある ボタンをクリックします。

- ポップアップメニューで、タブからプロファイルを選択するか、プロファイルの上にマウスポインターを置きます。
- ボタンをクリックします。



プロファイル構成の別プロファイルへの複製

既存のプロファイルの構成を別のプロファイルに複製するには、次の手順を実行します。

- グリッドの上部にある をクリックします。
- ポップアップメニューの ウィンドウで、ソースプロファイルを選択するか、ソースプロファイルの上にマウスポインターを置きます。
- ソースプロファイル名の右側に表示される ボタンをクリックします。
- で複製先のプロファイルを選択します。

プロファイル構成の削除

プロファイル構成を削除するには、次の手順を実行します。

- グリッドの上部にある をクリックします。

- ポップアップメニューで、リスト内のプロファイルを選択するか、プロファイルの上にマウスポインターを置きます。
- ソースプロファイル名の右側に表示される ボタンをクリックします。

権限の管理

ノード、アクション、サービスに権限を割り当てる方法には、次の 2 つがあります。

- 1 つ以上のセルを選択し、ツールバーまたはショートカットを使用して権限を適用します。
- セルが選択されていない状態で、ツールバーまたはショートカットを使用して権限を選択し、この権限を 1 つ以上のセルに適用します。この方法を使用すると、マウスポインターの横にアイコンが表示され、コンテキストに応じてアイコンが変化します。

注意:

権限ツールバーの および の値は、選択がノードであるかアクション/サービスであるかによって変わります。実際の値は、ノードの場合は と 、アクション/サービスの場合は と です。

以下のスクリーンショットでは、 サービスとユーザー **John Doe** に対応する権限セルが に設定されています。

動的ルール の定義

動的ルールは、コンテキストに応じてデータまたはユーザーサービスにアクセスするための条件をより正確に定義する可能性を提供します。

プログラムルールにはさまざまな種類があります。

- **AccessRule**。データに対するアクセスルールの定義のセクションで説明されています。
- スクリプト化されたレコードの権限ルール。データに対するスクリプト化された権限ルールの定義のセクションで説明されています。
- **ServiceActivationRule**。サービスのアクティベーションルールの定義のセクションで説明されています。
- **ServicePermissionRule**。サービスの権限ルールの定義で説明されています。

データに対するスクリプト化された権限ルールの定義

スクリプト化された権限ルールは、コンテキストに応じて、テーブルのレコードに対する読み取り/書き込み権限を動的に定義するルールです。

このようなルールを定義するには、DMA に **レコード権限スクリプト** を作成する必要があります。スクリプトエディターは、テーブルノード定義の タブ で利用できます。

データに対するアクセスルールの定義

`AccessRules` は、コンテキストに応じて、データモデルノードまたはテーブルのレコードに対する読み取り/書き込み権限をプログラムで定義するルールです。

`AccessRule` の定義は次のように実行されます。

1. `AccessRule` または `AccessRuleForCreate` インターフェイスを実装する Java クラスの形式でのルールの作成。
2. スキーマ拡張内の関連ノードへのこのルールの割り当て: `SchemaExtensions`。

ルールの対象 (モデルノードまたはレコード) とタイプ (`AccessRule` または

`AccessRuleForCreate`) に応じて、

`SchemaExtensionsContext.setAccessRuleOnOccurrence` や

`SchemaExtensionsContext.setAccessRuleForCreateOnNode` などいくつかのメソッドを

使用することができます。

このように割り当てられたルールは「ローカル」と呼ばれ、ターゲットエンティティが要求された場合にのみ実行されます。詳細については、[データ権限の解決](#)を参照してください。

注意:

ノード、データスペース、またはレコードごとに定義できる `AccessRule` は 1 つだけです。テーブルの子ノードごとに定義できる `AccessRuleForCreate` は 1 つだけです。あるタイプの新しいプログラムルールを定義すると、既存のルールが置き換えられます。

サービスのアクティベーションルールの定義

`ServiceActivationRules` を使用すると、特定のデータスペースまたはデータセットに対してサービスをアクティブ化するかどうかを指定できます。このルールによって非アクティブ化されたサービスは、現在のプロファイルに関係なく、権限画面であっても、非アクティブ化されたエンティティで実行または表示することはできません。

`ServiceActivationRule` の定義は、次のように実行されます。

1. サービスの種類に応じて、`ServiceActivationRuleForDataspace` インターフェイス または `ServiceActivationRuleForDataset` を実装する Java クラスの形式でルールを作成します。
2. `ActivationContextOnDataspace.setActivationRule` または `ActivationContextWithDatasetSet.setActivationRule` メソッドにより、サービスの種類に応じて、影響を受けるサービスの宣言レベルでこのルールを割り当てます。

結果として割り当てられたルールは、サービスアクティベーションの評価中に評価されます。詳細については、[サービスの権限の解決](#)を参照してください。

サービスの権限ルールの定義

`ServicePermissionRules` は、コンテキスト (現在のセッション、選択したエンティティなど) に応じて、サービスの表示条件と実行条件を動的に定義できる高度なルールです。このタイプのルールをトリガーするには、事前に現在のコンテキストに対してサービスをアクティブ化する必要があります。

`ServicePermissionRule` の定義は、次のように実行されます。

1. `ServicePermissionRule` インターフェイスを実装する Java クラスの形式でルールを作成します。
2. 次のように、このルールを、影響を受けるサービスに割り当てます。

- いずれの場合も、新しいサービスの場合は、`ActivationContext.setPermissionRule` メソッドを介した宣言レベルで割り当てます。

このように割り当てられたルールは「グローバル」と呼ばれ、現在のコンテキストでサービスがアクティブ化された場合にのみ実行されます。詳細については、[サービスの権限の解決](#)を参照してください。

- または、既存のサービスの場合 [スキーマ拡張](#)で、`SchemaExtensionsContext.setServicePermissionRuleOnNode` および `SchemaExtensionsContext.setServicePermissionRuleOnNodeAndAllDescendants` メソッドを介して割り当てます。この場合、1 つ以上のデータモデルノード (テーブルノード、関連付けノードなど) で、EBX® から提供される標準サービスを含む任意のサービスにルールを割り当てることができます。

このように割り当てられたルールは「ローカル」と呼ばれ、拡張スキーマコンテキストで、ノードが指定されたものに対応する場合にのみ実行されます。詳細については、[サービスの権限の解決](#)を参照してください。

注意:

モデルノードごとに定義できる `ServicePermissionRule` は 1 つだけです。したがって、新しいプログラムルールの定義は既存のルールを置き換えます。

データ権限の解決

ユーザー定義ルールの解決

ユーザーインターフェイスを使用して定義されたアクセス権限は、データスペース、データセット、レコード (該当する場合)、およびノードの 4 つのレベルで解決されます。

プロファイルが特定のレベルで制限付きアクセス権限に関連付けられている場合、そのレベルで定義されているすべての制限付き権限の最小値が解決されます。そのレベルで制限が定義されていない場合、そのレベルで定義されているすべてのアクセス権限の最大値が解決されます。

プロファイルに制限付き権限が定義されている場合、ユーザーの他のロールによって付与される可能性のある他の権限よりも優先されます。通常、現在のユーザーセッションに一致するすべてのユーザー定義の権限ルールについて、次のことが適用されます。

- 制限付きのルールが定義されている場合、これらの制限付きルールの最小権限が適用されます。
- 制限付きのルールが定義されていない場合、一致するすべてのルールの最大権限が適用されます。

例:

同じユーザーに関する 2 つのプロファイル *P1* と *P2* がある場合、次の表に、そのユーザーのサービスへの権限を解決する際の可能性を示します。

P1 認証	P2 認証	権限の解決
有効	有効	有効。制限に何の違いもありません。
無効	無効	無効。制限に何の違いもありません。
有効	無効	P2 の許可が制限されていない限り、有効になります。

P1 認証	P2 認証	権限の解決
無効	有効	P1 の許可が制限されていない限り、有効になります。

同じ制限ポリシーがデータアクセス権限の解決に適用されます。

別の例では、ビルトインプロファイル「Profile.EVERYONE」とアクセス権限「hidden」の間に制限付きの関連付けを定義することにより、データスペースをすべてのユーザーから隠すことができます。

どのレベルでも、このレベルとそれ以上のレベルで解決されたものの間で最も制限の厳しいアクセス権限が適用されます。たとえば、ユーザーのデータセットアクセス権限が読み取り/書き込みアクセスに解決されたが、コンテナデータスペースが読み取りアクセスのみを許可している場合、ユーザーはこのデータセットへの読み取り専用アクセスのみを持ちます。

注意:

データセットの継承メカニズムは、値とアクセス権限の両方に適用されます。つまり、データセットで定義されたアクセス権限は、その子データセットに適用されます。子データセットでこれらの権限をオーバーライドすることができます。

アクセス権限解決の例

この例では、次の定義済みのロールとプロファイルに属する 3 人のユーザーが存在します。

ユーザー	プロファイル
ユーザー 1	• user1 • ロールA • ロールB
ユーザー 2	• user2 • ロールA • ロールB

	• ロールC
ユーザー 3	• user3 • ロールA • ロールC

特定のエレメントのプロファイルのアクセス権限は次のとおりです。

プロファイル	アクセス権限	制限ポリシー
user1	非表示	はい
user3	読み取り	いいえ
ロールA	読み取り/書き込み	いいえ
ロールB	読み取り	はい
ロールC	非表示	いいえ

上記のロールとプロファイルのアクセス権限に基づいて解決した後、各ユーザーに適用される権限は次のとおりです。

ユーザー	解決されたアクセス権限
ユーザー 1	非表示
ユーザー 2	読み取り
ユーザー 3	読み取り/書き込み

データスペースとスナップショットのアクセス権限の解決

データスペースレベルでは、アクセス権限は次のように解決されます。

- ユーザーが複数のプロファイルを通じて定義された複数の権限を持っている場合
 - 権限に制限が含まれている場合は、最小限の制限的なプロファイルと権限の関連付けが適用されます。

- それ以外の場合は、プロファイルと権限の関連付けの最大値が適用されます。
- ユーザーに権限が定義されていない場合
 - ユーザーがビルトインの管理者またはデータスペースの所有者である場合、このデータスペースには読み取り/書き込みアクセス権限が付与されます。
 - それ以外の場合、データスペースは非表示になります。

データセットのアクセス権限の解決

データセットレベルでは、データスペースレベルと同じ原則が適用されます。データセットレベルのみでアクセス権限を解決した後、最終的なアクセス権限は、解決されたデータスペースの権限と解決されたデータセットの権限の間の最小の権限を取得することによって決定されます。たとえば、データスペースがユーザーに対して読み取り専用であると解決され、そのデータセットの1つが読み取り/書き込みであると解決された場合、ユーザーはそのデータセットへの読み取り専用アクセスのみを持ちます。

ノードアクセス権限の解決

ノードレベルでは、データスペースおよびデータセットレベルと同じ原則が適用されます。ノードレベルのみでアクセス権限を解決した後、最終的なアクセス権限は、解決されたデータセットの権限と解決されたノードの権限の間の最小の権限を取得することによって決定されます。

特定のアクセス権限は、ノードレベルで定義できます。特定のアクセス権限が定義されていない場合、デフォルトのアクセス権限が使用されます。

注意:

解決手順は、テーブルノードとテーブル子ノードでわずかに異なります。

テーブルおよびテーブルの子ノードの特殊なケース

これは、特定のテーブルノードまたはテーブルレコード N に使用される解決プロセスについて説明しています。

ユーザーのプロファイルの1つに一致するユーザー定義のアクセス権限ルールごとに、 N のアクセス権限は次のいずれかです。

1. N のローカルで定義されたアクセス権限。
2. テーブルノードで定義されたアクセス権限から継承。

3. データセット値のデフォルトのアクセス権限から継承。

一致するすべてのユーザー定義のアクセス権限ルールは、 N のアクセス権限を解決するために使用されます。解決は、[制限ポリシー](#)に従って行われます。

最終的に解決されたアクセス権限は、 N のデータスペース、データセット、および解決されたアクセス権限の間の最小値になります。

動的ルールの解決

動的アクセス権限ルールの解決には、データセット、レコード、ノードの3つのレベルがあります。特定のレベルに設定できるプログラムによるアクセスルールは1つだけなので、最後のルールセットは解決手順で使用されるルールセットです。ただし、スクリプト化されたルールは、テーブルレベルでプログラムルールの上に指定できます。

データセットのルール解決

データセットの場合、最後のルールセットが解決されたルールと見なされます。

レコードのルール解決

レコードの場合、解決されたルールは、データセットで解決されたルールセットとこのレコードで設定されたルールの間の最小値です。

詳細については、`SchemaExtensionsContext.setAccessRuleOnOccurrence` を参照してください。

ノードでのルール解決

レコードの子ノードであるノードの場合、解決されたルールは、レコード上の解決されたルールとこのノードに設定されたルールの間の最小値です。

データセットの子ノードの場合、解決されたルールは、データセットで解決されたルールセットとこのノードで設定されたルールの間の最小値です。

See `SchemaExtensionsContext.setAccessRuleOnNode` for more details.

外部キードロップダウンメニューの表示ポリシー

アクセスルールのためにレコードが非表示になっている場合、そのレコードは外部キーのドロップダウンメニューに表示されません。

注意:

データセットまたはデータセットノードで解決されたアクセス権限は、ユーザーインターフェイスで定義された解決されたアクセス権限と、解決された動的ルール (存在する場合) の間の最小値です。

サービスの権限の解決

ユーザーサービスは、ユーザーインターフェイスから特定の高度な機能を実行する可能性を提供します。定義に応じて、これらのサービスは、メニューから、ワークフローのアクションとして、パースペクティブアイテムとして呼び出すことも、URL から [Web コンポーネント](#) として直接実行することもできます。

以下も参照してください。:

- [概要](#)

サービスの権限は、サービスがユーザーインターフェイスから呼び出されるときに解決されます。つまり、次のようになります。

- 実行中、サービスが表示される直前。

ユーザーコンテキストで解決された権限が **有効** でない場合、サービスの代わりに制限メッセージが表示されます。

- サービスがメニューで表示可能として定義されている場合、メニューの表示中。

ユーザーのコンテキストで解決された権限が **有効** でない場合、サービスはメニューに表示されません。

したがって、すべてのリクエストに応じて、サービスの権限の解決は、次の順序で、条件に従う限り、次のように実行されます。

1. サービスのアクティブ化は、現在のコンテキストに対応している必要があります。このアクティベーションでは、次のことを考慮します。

- 選択したエンティティタイプ (データセット、テーブル、レコードなど)。
- `UserServiceDeclaration.defineActivation` メソッド内で定義された静的アクティベーションルール。

- 潜在的な動的アクティベーションルール ([ServiceActivationRule](#)) も `UserServiceDeclaration.defineActivation` メソッド内で定義されています。

2. 現在のコンテキストでサービスがアクティブ化されると、ユーザーセッションの権限が評価されます。

- 現在のユーザー (またはそのロール) のユーザーインターフェイスを介して権限が定義されている場合、その解決策は `有効` を返す必要があります。

詳細については、[ユーザー定義ルールの解決](#)セクションを参照してください。

- サービスに[グローバル権限ルール](#)が定義されている場合、提供されたコンテキストに対して `有効` を返す必要があります (`ServicePermissionRuleContext` を参照)。
- 選択したノードに対して[ローカル権限ルール](#)定義されている場合、提供されたコンテキストに対して `有効` を返す必要があります (`ServicePermissionRuleContext` を参照)。

ユーザー定義ルールの解決

例

この例では、異なるロールとプロファイルに属する 2 人のユーザーが存在します。

ユーザー	プロファイル
ユーザー 1	• user1 • ロールA • ロールB
ユーザー 2	• ロールC • ロールD

データセットレベルで定義されたロールとプロファイルに関連付けられている権限は次のとおりです。

プロファイル	ビルトインサービスの作成 (@CREATION)	ビルトインサービスの複製 (@DUPLICATE)	ビルトインサービスの比較 (@COMPARE)	カスタムサービス 1 (CUSTOM1)	カスタムサービス 2 (CUSTOM2)	制限ポリシー
user1	有効	無効	有効	無効	有効	いいえ
ロールA	有効	有効	無効	有効	無効	はい
ロールB	有効	無効	有効	有効	無効	はい
ロールC	有効	有効	無効	無効	無効	いいえ
ロールD	有効	無効	無効	有効	無効	いいえ

権限解決後に各ユーザーが利用できるサービスは次のとおりです。

ユーザー	利用可能なサービス
ユーザー 1	ビルトインサービスの作成 (@creation)
カスタムサービス 1 (custom1)	
ユーザー 2	ビルトインサービスの作成 (@creation)
ビルトインサービスの複製 (@duplicate)	
カスタムサービス 1 (custom1)	

以下も参照してください。:

- ユーザー定義ルールの解決

アクションの権限の解決

アクションは、プロファイルの実行権限を定義できる EBX® オブジェクト操作の低レベルの操作です。ユーザーインターフェイスにのみ影響するユーザーサービスの権限とは異なり、これらの権限は、操作がプログラムで (つまり、 **プロシージャ** を介して) または間接的に (たとえば、データ

のインポート中に、テーブルに対するアクション (作成、オーバーライド、非表示、および削除) が評価されます) 実行される場合にも適用されます。

権限を定義できるアクションのリストは次のとおりです。

アクションオブジェクト	利用可能なアクション
データスペース	子データスペースの作成
スナップショットの作成	
マージの開始	
アーカイブのエクスポート	
アーカイブのインポート	
データスペースを閉じる	
スナップショットを閉じる	
データセットの作成	
データセット	データセットの複製
データセットの削除	
データセットのアクティブ化/非アクティブ化	
ビューの作成	
テーブル	新しいレコードの作成
レコードの上書き	
レコードの非表示	
レコードの削除	

アクションの権限の解決では、現在のユーザー (またはそのロール) のユーザーインターフェイスを介して定義された権限のみが考慮され、ユーザーインターフェイスを介して定義された他の権限と同様に制限ポリシーが適用されます。

詳細については、[ユーザー定義ルールの解決セクション](#)を参照してください。

ユーザー定義ルール of 解決

例

この例では、異なるロールとプロフィールに属する 2 人のユーザーが存在します。

ユーザー	プロフィール
ユーザー 1	• user1 • ロールA • ロールB
ユーザー 2	• ロールC • ロールD

特定のテーブルのアクションのロールとプロフィールに関連付けられている権限は次のとおりです。

プロフィール	レコードの作成	レコードの上書き	レコードの非表示	レコードの削除	制限ポリシー
user1	いいえ	はい	いいえ	はい	いいえ
ロールA	はい	いいえ	はい	いいえ	はい
ロールB	いいえ	はい	はい	いいえ	はい
ロールC	はい	いいえ	いいえ	いいえ	いいえ
ロールD	いいえ	いいえ	はい	いいえ	いいえ

権限を解決した後に各ユーザーが使用できるアクションは次のとおりです。

ユーザー	利用可能なアクション
ユーザー 1	レコードの非表示
ユーザー 2	レコードの作成

ユーザー	利用可能なアクション
レコードの非表示	

以下も参照してください。:

- [ユーザー定義ルールの解決](#)

基準エディター

- [概要](#)
- [条件付きブロック](#)
- [アトミック基準](#)
 - [式](#)

概要

基準エディターは、ユーザーインターフェイスのいくつかの異なる領域に含まれています。これにより、テーブルフィルターの定義、およびデータの検証と計算のルールが可能になります。このエディターは、XPath 1.0W3C 勧告に基づいています。

基準には、アトミック基準と条件付きブロックの 2 種類があります。

以下も参照してください。:

- [サポートされている XPath 構文](#)

条件付きブロック

条件付きブロックは、アトミック基準とその他の条件付きブロックで構成されています。基準に基づいて条件を表現します。次のタイプのブロックが存在します。

- **基準が一致しない**：ブロック内のどの基準も一致しません。
- **すべての条件が一致するわけではない**：ブロック内の少なくとも 1 つの基準が一致しません。
- **すべての条件が一致する**：ブロック内のすべての基準が一致します。

- 少なくとも 1 つの基準が一致する：1 つ以上の基準が一致します。

アトミック基準

アトミック述語は、フィールド、演算子、および式 (値または XPath 式のいずれか) によって定義されます。

フィールド	基準が適用されるテーブルのフィールドを指定します。
演算子	使用する演算子を指定します。使用可能な演算子は、フィールドのデータ型によって異なります。
値	値または式を指定します。以下の 式 を参照してください。
コードのみ	オンにすると、デフォルトで検索されるラベルではなく、フィールドの基になる値を検索することを指定します。

式

式は、固定値または式のいずれかです。フィルタを作成するときは、固定値のみが許可されます。検証または計算ルールの作成中に、ウィザードを使用して数式を作成できます。

****既知の制限事項：****数式フィールドは入力値を検証せず、構文とパスのみがチェックされます。

検索

- [概要](#)
 - [値のラベル付け](#)
 - [制限事項](#)
- [文字列フィールドの検索戦略](#)
 - [文字列の基本的なビルトイン戦略](#)
 - [文字列の高度な戦略](#)
 - [文字列フィールドのデフォルト戦略](#)
 - [高度なカスタム戦略](#)

- 同義語とストップワード
- 主キーフィールドの検索戦略
- 外部キーフィールドの検索戦略
- 関連付けの検索戦略
- 検索からのフィールドの除外 (「Void」 インデックス)
- 検索からのフィールドの除外 (プロパティ)
- 検索戦略へのフィールドの割り当て

概要

検索戦略は、フィールドのインデックス作成とクエリの方法を定義します。すべてのフィールドは、主にそのデータ型に基づいて、デフォルトの検索戦略に関連付けられています。

検索戦略は、データモデルアシスタントで指定されます。

- フィールドを編集するとき、その検索戦略は タブで設定できます。
- データモデルレベルでは、左側のペインの > エLEMENTでカスタム検索戦略を指定できます。

以下も参照してください。:

- クイック検索

値のラベル付け

値のラベル付けは、未加工の値の代わりにユーザーフレンドリーなラベルを表示する EBX® のグローバル機能です。たとえば、ユーザーインターフェイスでは、外部キーフィールドにはリンクされたレコードのラベルが表示され、静的列挙に基づくフィールドにはデータモデルで指定された未加工の値に関連するローカライズされたラベルが表示されます。

フィールドで値のラベル付けがサポートされる場合、直感的なユーザーインターフェイスを維持するために、ユーザーインターフェイスでのクイック検索と並べ替えは通常、表示されたラベルに適用されます。

一部例外的に、未加工の値がクイック検索と並べ替え操作に使用される場合もあります。

- プログラムラベルとプログラム列挙制約 (`TableRefDisplay`) を指定する外部キー、その表示がターゲットテーブルで指定された `UILabelRenderer` に依存する外部キー、または

`ConstraintEnumeration` によって制約されるフィールド)。別の解決策 (表示パターンと外部キー) の使用をお勧めします。

- 別のノード (

明らかに、フィールドが `UIWidget` (または `UIBean`) を介して表示される場合、直感的なユーザーインターフェイスを維持するために、カスタムコンポーネントではラベル (このフィールドで値のラベル付けが有効でない場合は値) の表示が期待されます。

制限事項

次のフィールドは、検索やその他の演算子に対して最適化されていません。

- 計算フィールド:
 - `ValueFunction` API を実装する。
 - コンテナテーブルの外部のフィールドに依存する計算ルールとスクリプトを使用する (非ローカル依存関係)。
- 継承フィールド

結果として、これらは通常、[クイック検索](#) および [テーブルの xpath 検索](#) (`osd:search('','pattern')` 経由) から除外されます。

継承されたデータセット、履歴ビュー、またはマップされたテーブルの特定のケースでは、レガシー検索が使用されます。これは、テーブルのサイズをすぐに見積もることができず、UI に表示されない可能性があることを意味します。また、次のようなクイック検索も意味します。

- すべての検索可能なフィールド (ローカル依存関係のない計算フィールドを含む) を考慮します。
- 「contains」のように動作します (Lucene 構文は使用できません)。
- 関連性による並べ替えをサポートしません。
- 大量のデータを含むテーブルではパフォーマンスが低下する可能性があります。

表示パターンを定義するノードの場合、デフォルトの検索テンプレートでは並べ替え不可能な検索戦略は禁止されています。ただし、他の検索テンプレートでは許可されています。

文字列フィールドの検索戦略

文字列の基本的なビルトイン戦略

「テキスト」	「テキスト」検索戦略は、説明、テキスト、コメントなどの複数の単語を含める場合に使用します。この戦略は、全文検索とあいまい検索をサポートします。並べ替え、および「equals」や「starts-with」演算子などの一部の関数は無関係であり、サポートされません。この戦略は軽量で、ディスクスペースをほとんど消費しません。 クイック検索も参照してください。
「コード」	「コード」検索戦略は、コードと識別子を対象としています。値は単一のトークンと見なされ、大文字と小文字を区別するあらゆる種類のフィルターを使用できます。全文検索は無関係であり、デフォルトでは「contains」に置き換えられます。これは、カスタムの「コード」戦略を定義することにより、「starts-with」に変更できます。大規模の検索では、パフォーマンスの向上のため、「starts-with」の使用をお勧めします。
「名前」	「名前」検索戦略は、数語のみを含む名前やラベルを対象としています。「テキスト」と同じ検索機能に加えて、「名前」戦略では並べ替えも可能で、「コード」と同じフィルターをサポートします。この戦略は最も多くの機能を備えていますが、より多くのディスクスペースを消費します。フィールドで利用できる場合は、この戦略ではなく、「テキスト」、「コード」、または「検索から除外」の戦略を選択することをお勧めします。

文字列の高度な戦略

高度な検索戦略は、別のファジーアルゴリズムを使用した検索をサポートすることを目的としており、並べ替えやフィルタリング操作はサポートしていません。カスタム検索テンプレートでそれらを使用し、デフォルトの検索テンプレートで基本的な戦略を使用することをお勧めします。

「レーベンシュタイン」	「レーベンシュタイン」戦略は、少数の単語を含む名前とラベルを対象としています。この戦略は、編集距離アルゴリズムに基づくあいまい検索のみをサポートします。検索構文 (+、-、...) はサポートされていません。この戦略は、大容量ではうまく機能しません。戦略のパラメーターにより、最大編集距離を 1 に設定し、不変のプレフィックスを使用することで、パフォーマンスを優先することができます。
「Soundex」	「Soundex」戦略は、少数の単語を含む名前とラベルを対象としています。この戦略は、Soundex 近似音声アルゴリズムに基づくあいまい検索のみをサポートします。このアルゴリズムは英語用で、数字を無視します。
「Double Metaphone」	「Double metaphone」戦略は、少数の単語を含む名前とラベルを対象としています。この戦略は、Double metaphone 近似音声アルゴリズムに基づくあいまい検索のみをサポートします。このアルゴリズムは、Soundex よりも最新であり、特に英語以外の言語でより良い結果が得られます。数字は無視されます。
「NGram 戦略」	「NGram」戦略は、少数の単語を含む名前とラベルを対象としています。この戦略は、距離アルゴリズムに基づくあいまい検索のみをサポートします。このアルゴリズムは、値を「グラム」と呼ばれる小さなシーケンスに分割します。「レーベンシュタイン」検索戦略よりも優れたパフォーマンスを発揮しま

すが、より多くのディスクスペースを消費します。

文字列フィールドのデフォルト戦略

「名前」戦略は、以下を除いて、デフォルトで文字列フィールドに適用されます。

- フィールドが主キーの一部である場合、デフォルトで「コード」に設定されます。
- フィールドが外部キーの場合、「コード」に強制され、変更できません。
- フィールドに `xs:string` を拡張するビルトインのデータ型が含まれる場合、このデータ型に関連する戦略 (例、`osd:text`、`xs:Name`、`osd:email`、`osd:html`) が使用できます。

デフォルトの「名前」戦略が無関係であり、より多くのディスクスペースを消費する可能性があるため、データモデルのコンパイルでは、戦略が意図的に定義されていることを確認するために、「名前」戦略がデフォルトとして設定されているフィールドで警告が報告されます。概算で、期待される値の長さが 80 バイトを超える場合は、「テキスト」戦略を選択することをお勧めします。長い値 (UTF-8 エンコードでは 32766 バイト超) は、「名前」または「コード」戦略で完全にインデックス付けできません。クイック検索は影響を受けませんが、並べ替えでは最初の 1000 バイトのみが考慮され、一部の演算子 (「equals」および「ends-with」、SQL DISTINCT および COUNT DISTINCT、...) は正しい結果を返しません。

高度なカスタム戦略

一部の戦略は、たとえばストップワードや特定の言語を定義するためのパラメーターを受け入れます。これは、「検索」データモデル拡張機能の「検索戦略」テーブルにレコードを作成することによって行われます。新しいパラメーター化された戦略は、互換性のあるフィールドのタブで選択できるようになります。

同義語とストップワード

「検索」データモデル拡張機能でストップワードと同義語のリストを定義できます。「ストップワードリスト」または「同義語リスト」テーブルに新しいレコードを作成し、「カスタム検索戦略」テーブルのタブで作成したリストを選択します。

主キーフィールドの検索戦略

主キーフィールドには、デフォルトの検索テンプレートで定義された並べ替え可能な検索戦略が必要です。これには、すべてのデータ型の「Void」戦略と文字列の「テキスト」戦略は含まれません。デフォルトの検索テンプレート以外の検索テンプレート内で定義されている場合は、主キーフィールドに並べ替え不可能な検索戦略を使用することに注意してください。

外部キーフィールドの検索戦略

外部キーフィールドには、次の 2 つのレベルの検索があります。

- 最初に、該当する場合、外部キーの表示ラベルの各フィールドに対して検索が実行されます。各フィールド戦略は、ターゲットテーブルのフィールドから継承されます。この最初のレベルは、検索文字列をターゲットフィールドのデータ型に変換できない場合など、常に適用できるとは限りません。
- 次に、検索の第 1 レベルを適用できない場合、検索は対象の主キーの文字列表現に対して実行されます。データモデルアシスタントの タブで外部キーフィールドの検索戦略を変更すると、この第 2 レベルの検索にのみ影響します。「コード」検索戦略 (ビルトインまたはカスタム) のみにすることができます。

以下も参照してください。:

- `PrimaryKey.syntax`

以下も参照してください。:

- 値のラベル付け

関連付けの検索戦略

関連付けの場合、検索は関連付けレコードのラベルで使用する各フィールドに対して実行されます (各フィールド戦略はターゲットテーブルのフィールドから継承されます)。適用するには、ターゲットフィールドが検索用に最適化されており、検索基準がその型に変換可能にする必要があります。

関連付けの検索はラベルの最適化されたフィールドにのみ適用できるため、継承されたデータセットでは機能しません (制限事項で説明されているように、最適化された検索はサポートされていません)。

検索からのフィールドの除外 (「Void」インデックス)

「検索から除外」 (または `Void`) 戦略は、インデックス付けを無効にし、フィルターの作成、検索、または並べ替えを不可能にします。すべてのデータ型で使用でき、クエリされないフィールド

を対象としています。値には、レコードからアクセスできます。インデックス付けを無効にすると、消費されるディスクスペースが減り、データインポートなどの一部の処理が高速化されます。

検索からのフィールドの除外 (プロパティ)

プロパティ `osd:defaultView/hiddenInQuickSearch="true|false"` を使用して、クイック検索ツールからフィールドを除外することもできます。

詳細については、[デフォルトビュー](#)を参照してください。

フィールドへの検索戦略の割り当て

[検索テンプレート](#)を使用して、検索戦略をフィールドに関連付けることができます。これは、データモデルアシスタントで、フィールドの [タブ](#)で実行されます。フィールドに複数の検索戦略を割り当てるには、追加の検索テンプレートをモジュールに登録する必要があります。追加の検索テンプレートに関係するのは、アドオンの EBX® Information Search と EBX® Match and merge のみです。

管理ガイド

- 管理の概要
- インストールと構成
 - サポートされる環境
 - ディスク要件
 - Jakarta EE の展開
 - EBX® のインストールに関する注意事項
 - JBoss EAP 8.0.X のインストールに関する注意事項
 - WebSphere Application Server Liberty 23.X のインストールに関する注意事項
 - Tomcat 10.1.X のインストールに関する注意事項
 - EBX® メタデータ管理アプリケーションのインストールに関する注意事項
 - EBX® Container Edition のインストールに関する注意事項
 - オンプレミス Tomcat 10.1.X のインストールに関する注意事項
 - TIBCO EBX® メイン構成ファイル
 - 初期化と初回起動アシスタント
 - TIBCO EBX® アドオンの管理
 - ユーザー認証
 - パフォーマンスとチューニング
 - 構成に関する注意事項
- EBX® Container Edition
 - イメージのビルド
 - イメージの実行
 - イメージのカスタマイズ
- 技術管理
 - リポジトリ管理

- UI 管理
- UI – ワークフローランチャー
- ユーザーとロールのディレクトリ
- データモデルの管理
- データベースマッピング管理
- 外部サービスの設定
- ワークフロー管理
- タスクスケジューラ
- バックグラウンドタスク
- 監査証跡
- コラボレーション機能 (チームアップ)
- AI アシスタント
- その他
- 分散データ配信 (D3)
 - D3 の概要
 - D3 ブロードキャストおよび配信データスペース
 - D3 JMS 構成
 - D3 管理
- ステージング
- 新しいユーザーインターフェイスのプレビュー
- EBX® メタデータ管理アプリケーション
 - メタデータ管理のパーспекティブ
 - ユーザーロール
 - 権限
 - ユーザーインターフェイスの構成
 - ワークフロー
 - Web サービスと API
 - メタデータエージェント
 - TIBCO® メタデータエージェントのインストール

- TIBCO TDV メタデータアダプタのインストール
- TIBCO® EBX® メタデータアダプタのインストール
- TIBCO® メタデータエージェントフレームワーク
- TIBCO® メタデータエージェントの管理
- テクニカルユーザー
- TIBCO® メタデータエージェント
- メタデータエージェントによって管理される項目
- 収集の構成

管理の概要

TIBCO EBX® の管理セクションは、すべての管理タスクの主要なエントリポイントです。この概要には、管理者が習得する必要があるすべてのトピックがリストされています。ドキュメント内の対応する章または段落にアクセスするには、関心のあるトピックをクリックしてください。

- リポジトリ管理
 - オブジェクトキャッシュ
 - 制限事項
 - 使われなくなったコンテンツ
 - ワークフロー
- ディスク容量管理
 - ログの削除
 - XML 監査証跡 (非推奨)
- データモデル
 - パブリケーションの管理
 - データモデルの更新
- データ検証
 - 検証レポートのリセット
- パースペクティブ
- ホームページ

- ログの構成
 - ログメカニズムの順序と組み合わせ

リポジトリ管理

ストレージを最適化するには、リポジトリ (永続性 RDBMS) を必要最小限に維持することをお勧めします。このため、スナップショットと廃止されたデータスペースの削除を定期的に行い、バックアップファイルシステムの使用を検討することをお勧めします。

データスペース、スナップショット、履歴のクリーンアップおよびデータスペース、スナップショット、履歴の削除も参照してください。

ストレージコストを削減するために、ファイルシステムタイプのファイルをアーカイブすることもできます。[EBX® の監視](#)を参照してください。

管理タスクは、ビルトインタスクのタスクスケジューラを使用してスケジュールできます。[タスクスケジューラ](#)を参照してください。

オブジェクトキャッシュ

EBX® はオブジェクトキャッシュをメモリに保持します。オブジェクトのキャッシュサイズは、リポジトリの状態レポートを継続的に監視しながら、特定のニーズと要件 (プリロードオプションと参照データスペース、参照ポイント、および監視の事前検証) に従ってケースバイケースで管理する必要があります。

以下も参照してください。:

- [メモリ管理](#)

制限事項

EBX® には以下の制限事項が適用されます。G_DSP テーブルの dsp_id 列の値は 2,147,483,648 に制限されています。この値はデータスペースが作成されるたびに増加し、最後に作成されたデータスペースを削除するパージ後にアプリケーションが再起動した場合にのみ減少します。単一の JVM 実行で最大 2,147,483,648 の HomeKey インスタンスを作成できます。G_DST テーブルの dst_id 列の値は 65,536 に制限されています。この値は、新しいデータセット参照でデータセットが作成され永続化されるたびに増加します。データスペースには最大 8,192 のデータセットを含め

ることができます。パフォーマンスを考慮すると、32 を超えるデータセットは推奨されません。データモデルは最大 65,535 のテーブルを宣言できます。パフォーマンスを考慮すると、256 を超えるテーブルは推奨されません。

使われなくなったコンテンツ

古くなったコンテンツをリポジトリに保持すると、サーバーの起動が遅くなり、インターフェイスの応答が遅くなる可能性があります。古いコンテンツを削除することを強くお勧めします。

例：削除されたデータモデルまたは展開済みでないアドオンモジュールを参照するデータセット。TIBCO EBX® アドオンの管理を参照してください。

ワークフロー

クリーンアップ

ワークフロー履歴と関連する実行データは、定期的にクリーンアップする必要があります。

ワークフロー履歴には、完了したワークフロー、それぞれのステップおよびコンテキストに関する情報が格納されます。これにより、古い履歴を含むデータベースが増え続けるため、定期的に削除しないとデータベースのパフォーマンスが低下する可能性があります。詳細については、[ワークフロー履歴](#)を参照してください。

メール設定

ワークフローメール通知を実装できるようにするには、ワークフローメールを事前に設定する必要があります。詳細については、[構成](#)を参照してください。

ディスク容量管理

ログの削除

ログファイルのサイズはログレベル (および選択した重大度レベル) に応じて変化するため、それに応じてディスク領域を管理する必要があります。

EBX® メイン構成ファイルのメカニズム

EBX® には自動削除が用意されており、ログファイルを保存する日数を定義できます。定義された期間の後、ログファイルは削除されます。

ログ削除のカスタム管理 (バックアップ、アーカイブなど) は管理者の責任です。

詳細については、[カスタム 'ebxFile' アペンダー](#) を参照してください。

XML 監査証跡 (非推奨)

EBX® には XML 監査証跡マネージャーが付属していますが、現在はデフォルトで無効になっています。カスタム管理 (削除、バックアップなどを含む) は管理者の責任です。

詳細については、[レガシー XML 監査証跡のアクティブ化 \(非推奨\)](#)および[レガシー XML 監査証跡 \(非推奨\)](#) を参照してください。

データモデル

パブリケーションの管理

[埋め込みデータモデル](#)のパブリケーション管理。これらのパブリケーション管理および実行可能な管理タスク (削除、インポート、エクスポート) についての詳細については、[データモデルの管理](#)を参照してください。

データモデルの更新

EBX® で管理されていない XML スキーマドキュメントを使用しているデータモデルを更新することができます。詳細については、[データモデル更新ツール](#)を参照してください。

データ検証

検証レポートのリセット

検証レポートは、管理メニューからリセットできます。リポジトリに含まれるすべてのデータセットの検証レポートをリセットすることも、表形式のビューで選択したデータセットの検証レポートのみをリセットすることもできます。したがって、関連するデータセットは、次の明示的な検証要求で完全に検証されます。

詳細については、`Adaptation.resetValidationReport` を参照してください。

パースペクティブ

EBX® は、広範な UI カスタマイズオプションを提供します。システムにアクセスする各プロファイル専用の簡略化されたインターフェイス ([推奨されるパースペクティブ](#)) は、管理者がパラメータ化できます。ログインしているユーザーのプロファイルに応じて、インターフェイスは多かれ少なかれオプションとメニューを提供します。これにより、合理化された作業環境が可能になります。

詳細については、[詳細パースペクティブ](#)を参照してください。

ホームページ

EBX® はカスタマイズ可能なホームページを提供しており、『パースペクティブ』メニューから、または `/ebx-hub` URL を使用して直接アクセスできます。

現状では、このページに表示されるリンクは、新しいタブでターゲットを開きます。この動作は、EBX® の将来のバージョンで変更される可能性があります。

注意:

ホームページには、`ebx-hub` war が展開されている必要があります。後者が展開されていない場合は利用できません。

ログの構成

EBX® ログは、Apache Log4j™ 2、`ebx.properties` メイン構成ファイル、またはそれらの組み合わせを使用して構成できます。

EBX® ログ管理は Apache Log4j™ 2 に大きく依存しています。つまり、ログは初期化および自動再構成メカニズムで Log4j 自動構成を利用できます。これらのメカニズムの詳細については、[Apache Log4j™ 2 構成のドキュメント](#) および [EBX® ログの構成](#) を参照してください。

`ebx.properties` メイン構成ファイルは、EBX® 構成プロパティを一元管理します。ただし、初期化時の Apache Log4j™ 2 自動構成は、このメカニズムよりも優先されます。さらに、Web アプリケーションサーバーの起動プロセスにより、アプリケーションの初期化順序が予測不能になります。これらの要因により、ログメッセージが `ebx.properties` ファイルで指定された構成に従って生成されない可能性があります。詳細については、[EBX® ログの構成](#) を参照してください。

ログメカニズムの順序と組み合わせ

Web アプリケーションサーバーの起動時に、Apache Log4j™ 2 自動構成が最初に行われます。次に、Web アプリケーションの初期化フェーズ中に、`ebx.properties` メイン構成ファイルが処理され、ログが再構成されます。再構成には、構成全体の置き換えが伴います。

Apache Log4j™ 2 自動再構成メカニズムが使用されている場合、最初の初期化後 (つまり、EBX® メイン構成ファイルが処理される前または後) いつでも、Log4j 構成コンテキストに基づいてログを再構成できます。`ebx.properties` ファイルが更新された場合、最初の処理の後、いつでもファイルの内容に基づいてログを再構成することができます。

これらのメカニズムは排他的ではないため、初期化時に Apache Log4j™ 2 自動構成を使用して初期構成を設定し、それを `ebx.properties` ファイルの構成に置き換えることができます。

注意:

ログの再構成をバイパスするには、`ebx.log4j.rootCategory` とすべての `ebx.log4j.category.*` プロパティを `ebx.properties` ファイルから削除します。

インストールと構成

- サポートされる環境
- ディスク要件
- Jakarta EE の展開
- EBX® のインストールに関する注意事項
 - JBoss EAP 8.0.X のインストールに関する注意事項
 - WebSphere Application Server Liberty 23.X のインストールに関する注意事項
 - Tomcat 10.1.X のインストールに関する注意事項
- EBX® メタデータ管理アプリケーションのインストールに関する注意事項
 - EBX® Container Edition のインストールに関する注意事項
 - オンプレミス Tomcat 10.1.X のインストールに関する注意事項
- TIBCO EBX® メイン構成ファイル
- 初期化と初回起動アシスタント

- TIBCO EBX® アドオンの管理
- ユーザー認証
- パフォーマンスとチューニング
- 構成に関する注意事項

サポートされる環境

- ブラウザー環境
 - サポートされる Web ブラウザー
 - 画面の解像度
 - ページの更新
 - [戻る] および [次へ] ボタン
 - ズームのトラブルシューティング
 - ブラウザーの構成
 - 制限事項
- サポートされるアプリケーションサーバー
- サポートされるデータベース

ブラウザー環境

サポートされる Web ブラウザー

TIBCO EBX® Web インターフェイスは、コンピューター上の次のブラウザーのみをサポートします。モバイルデバイスではサポートされていません。

Microsoft Edge Chromium	Microsoft Edge Chromium は頻繁に更新され、自動更新を無効にすることはできないため、Cloud Software Group, Inc. はテストのみを行い、利用可能な最新バージョンをサポートするために最善の努力を行っています。
Mozilla Firefox ESR	Mozilla Firefox ESR は頻繁に更新され、自動更新を無効にすることはできないため、Cloud Software Group, Inc. はテストのみを

	行い、利用可能な最新バージョンをサポートするために最善の努力を行っています。
Google Chrome	Google Chrome は頻繁に更新され、自動更新を無効にすることはできないため、Cloud Software Group, Inc. はテストのみを行い、利用可能な最新バージョンをサポートするために最善の努力を行っています。

画面の解像度

EBX® の画面解像度は 1024x768 以上です。

ページの更新

ブラウザーページの更新は、EBX® ではサポートされていません。ページの更新が実行されると、最後のユーザーアクションが再実行されるため、問題が発生する可能性があります。したがって、ページを更新する代わりに、EBX® が提供するアクションボタンとリンクを使用することが不可欠です。

および ボタン

ブラウザーの および ボタンは、EBX® ではサポートされていません。ページ履歴をナビゲートするときに、古いユーザーアクションが再実行されるため、問題が発生する可能性があります。したがって、ブラウザーのボタンではなく、EBX® が提供するアクションボタンとリンクを使用することが不可欠です。

ズームのトラブルシューティング

ズームインまたはズームアウトすると、表示上の小さな問題が発生する場合があります (余分なスクロールバーや位置ずれなど)。これらの問題は、提供されたナビゲーションリンクを使用して画面を更新することで修正できます。

ブラウザーの構成

ユーザーインターフェイスが正しく機能するためには、ブラウザーの構成で次の機能を有効にする必要があります。

- JavaScript
- Ajax
- ポップアップ

- Cookie

注意:

EBX® の正常な機能を妨げる可能性があるため、ブラウザーの拡張機能やプラグインの使用は避けてください。

制限事項

一部のブラウザーでは、埋め込み可能な iframe の数に制限がある場合があります。この場合、階層リンクにプッシュ可能な項目数に制限されます。詳細については、ブラウザーのドキュメントを確認してください。

サポートされるアプリケーションサーバー

このセクションの情報は、EBX® と互換性のあるアプリケーションサーバーの詳細を提供します。サポートされるバージョンは、承認前に徹底的にテストされ、継続的な互換性を確保するために積極的にテストされています。最新テクノロジーを採用する際にも、この同じアプローチが使用されます。EBX® は現在 Java 25 をサポートしており、このバージョンが LTS リリースとして最終決定される前からテストが開始されました。このテストは良好な結果をもたらしました。たとえば、Java 23 で導入されたバグ修正は、EBX® 5 から EBX® 6 への移行に影響を与えました。この問題はその後解決されました。しかし、潜在的な移行問題を防止するために警告を発することができました。

EBX® は次の構成をサポートします。

- Java ランタイム環境 (JRE): OpenJDK ベースの長期サポート (LTS) バージョン 25、21、または 17、あるいは WebSphere Liberty JDK 17 または 21。JRE 25 は 21 または 17 よりも推奨されます。報告されたバグや問題が対処可能であることを保証するため、それらは OpenJDK で再現可能である必要があります。ご使用の JDK が OpenJDK ベースであるか不明な場合は、JDK ベンダーにご確認ください。

EBX® は JDK 17 でコンパイルされており、セキュリティとパフォーマンスの向上を活用するために、JRE を最新のサポートされている LTS バージョン (OpenJDK 21.0.9 に関する上記の注記を参照) にアップグレードすることをお勧めします。最新の JRE アップデートがあることを確認するために、定期的にチェックを行う必要があります。

EBX® のサポートは JRE とアプリケーションサーバー (例: Tomcat、JBoss) に基づいており、オペレーティングシステムには依存しません。したがって、JRE とアプリケーションサーバーが特定の OS でサポートされている限り、EBX® はその OS をサポートします。

注意:

Unicode CLDR は JRE バージョンで変更され、整数や 10 進表現などのロケールに依存するフォーマットに影響を与えます。 `java.locale.providers` システムプロパティを使用して、ロケールに依存するサービスの検索順序を構成します。値 `COMPAT` を使用すると、JRE 8 との互換性を確保できます。詳細については、`java.util.spi.LocaleServiceProvider` を参照してください。

- Servlet 6.0 以上に準拠する任意の Servlet/JSP コンテナ。たとえば、Tomcat 10.1.X (ただし、`unpackWARs` オプションが `true` に設定されている場合のみ)。詳細については、[Tomcat アプリケーションサーバーと Java 仮想マシンの構成](#)を参照してください。Java SE 25、21、または 17 LTS をサポートし、Jakarta EE 10 以上を使用する任意の Java アプリケーションサーバー。たとえば、Jakarta EE 10 の場合は WebSphere Liberty 23 以上、JBoss EAP 8.0.0 以上。[Jakarta EE 展開](#)を参照してください。
- アプリケーションサーバーは、JSON Processing 1.1 ([JSR 374](#)) をサポートするか、`ebx.jar` ライブラリに組み込まれた実装の使用を許可する必要があります。たとえば、Tomcat はこの仕様をサポートするライブラリを提供していません (組み込まれたライブラリのみ使用できます)。WebSphere Application Server Liberty および JBoss EAP では、使用可能なライブラリを含めたり除外したりできます。
- アプリケーションサーバーは、EBX® からの HTTP クエリ文字列に UTF-8 エンコードを使用する必要があります。これは、アプリケーションサーバーレベルで設定できます。

たとえば、Tomcat では、`server.xml` 構成ファイルで、`<Connector>` の `URIEncoding` を UTF-8 に設定することで、サーバーが常に UTF-8 エンコードを使用するように設定できます。別の方法として、`server.xml` でパラメーター `useBodyEncodingForURI` を `true` に設定してリクエスト本文のエンコードを使用するようにサーバーに指示することもできます。

注意:

- クラスタリングおよびホットデプロイメント/アンデプロイメントに関して制限が適用されます。

クラスタリング：EBX® にはキャッシュ同期メカニズムが含まれていないため、アクティブインスタンスのクラスターに展開することはできません。詳細については、[テクニカルアーキテクチャ](#)を参照してください。

ホットデプロイメント/アンデプロイメント：EBX® では、EBX® モジュールまたは EBX® のビルトイン Web アプリケーションとして登録された Web アプリケーションのホットデプロイメント/アンデプロイメントはサポートされません。

サポートされるデータベース

EBX® リポジトリは、適切な JDBC ドライバーを使用した以下のリレーショナルデータベース管理システムをサポートします。データベースベンダーの推奨に従い、データベース自体および JDBC ドライバーに関するポリシーを更新することが重要です。

Oracle データベース 19c、23c 以降。	<code>null</code> 値の区別には、特定の制限があります。単純な <code>xs:string</code> エLEMENT では、Oracle は空の文字列と <code>null</code> 値の区別がサポートされません。詳細については、空の文字列管理を参照してください。 EBX® によるデータベース接続ユーザーには、次の権限が必要です。 • CREATE SESSION • CREATE TABLE • ALTER SESSION • CREATE SEQUENCE • デフォルトテーブルスペースの null 以外のクォータ
PostgreSQL 14 以降。	EBX® によるデータベース接続ユーザーには、EBX® リポジトリをホストするデータベースに対する CONNECT 権限が必要です。これ以外には、このデータベースのパブリックスキーマに対するデフォルトの権限が適切です。

	また、マッピングモードでのデータモデルの進展については、この 制限事項 を参照してください。
Amazon Aurora PostgreSQL (PostgreSQL 14 以降と互換性あり)。	PostgreSQL に関する上記セクションのコメントが適用されます。
Google Cloud SQL for PostgreSQL (PostgreSQL 14 以降と互換性あり)。	PostgreSQL に関する上記セクションのコメントが適用されます。
Microsoft Azure Database for PostgreSQL (PostgreSQL 16 以降と互換性あり)。	PostgreSQL に関する上記セクションのコメントが適用されます。
Microsoft SQL Server 2016 以降。	EBX® を Microsoft SQL Server とともに使用する場合、デフォルトのデータベース照合を使用して、データベースに格納された文字列を比較および並べ替えます。これは、データモデル定義で使用する文字列、および履歴テーブルに格納されるデータに適用されます。デフォルトのデータベース照合は、データベースの作成時に指定できます。それ以外の場合は、データベースサーバーの照合が使用されます。名前の競合や予期しない動作を回避するには、大文字と小文字およびアクセント符号を区別する照合をデフォルトのデータベース照合として使用する必要があります (照合名の末尾に「CS_AS」が付いているか、照合がバイナリ)。 SQL Server でトランザクション分離を強制するデフォルト設定は、ペシミスティックモデルに従います。読み取り/書き込みの同時アクセスを回避するために行はロックされます。これにより、マッピングされたテーブル (履歴またはリレーショナル) でライブリネスの問題が発生する可能性があります。このような問題を回避するには、SQL Server デー

	データベースでスナップショット分離を有効にすることをお勧めします。 EBX® によるデータベース接続ユーザーには、次の権限が必要です。 • EBX® リポジトリをホストするデータベースでの CONNECT、SELECT、CREATE TABLE • デフォルトスキーマでの ALTER、CONTROL、UPDATE、INSERT、DELETE
Microsoft Azure SQL データベース	EBX® は、Microsoft Azure SQL Database v12 (12.00.700) で認定されており、現在のバージョンの Azure データベースサービスとの互換性を検証するために定期的にテストされています。 EBX® を Microsoft Azure SQL と併用すると、デフォルトのデータベース照合を使用して、データベースに格納されている文字列を比較および並べ替えます。これは、データモデル定義で使用される文字列、および履歴テーブルに格納されているデータに適用されます。デフォルトのデータベース照合は、データベースの作成時に指定できます。それ以外の場合は、データベースエンジンサーバーの照合が使用されます。名前の競合や予期しない動作を回避するには、大文字と小文字をおよびアクセント符号を区別する照合をデフォルトのデータベース照合として使用する必要があります (照合名の末尾に「CS_AS」が付いているか、照合がバイナリ)。 EBX® によるデータベース接続ユーザーには、次の権限が必要です。

	• EBX® リポジトリをホストするデータベースでの CONNECT、SELECT、CREATE TABLE • デフォルトスキーマでの ALTER、CONTROL、UPDATE、INSERT、DELETE
H2 2.2.224 以降	H2 は、実稼働環境ではサポートされていません。 デフォルトの H2 データベース設定では、レコード変更時の一貫した読み取りができません。

その他のリレーショナルデータベースについては、弊社サポートチーム (<https://support.tibco.com>) にお問い合わせください。

注意:

EBX® リポジトリの整合性を保証するために、データベースに直接変更を加えることは固く禁じられています (例、SQL への直接書き込み)。

以下も参照してください。:

- リポジトリ管理
- EBX® リポジトリのデータソース
- EBX® リポジトリの構成

ディスク要件

- 場所とディスクの種類
- ウイルス対策
- ディスクのサイズ設定

場所とディスクの種類

`ebx.properties` ファイルの `ebx.repository.directory` プロパティを使用して、EBX® ルートディレクトリの場所を設定します。このディレクトリには Lucene インデックスが含まれ、ローカルの低遅延ハードディスクでホストする必要があります。SSD ディスクが推奨されます。この指示は、アクセス速度の低下やインデックス管理のエラーを防ぐために、リポジトリおよびドキュメントのインデックスに適用されます。EBX® はリレーショナルデータベースに保存されているデータからインデックスを再構築できるため、インデックスにローカルハードディスクを使用すると、データ損失のリスクはありません。ただし、回復操作には時間がかかる場合があります。

注意:

EBX® ルートディレクトリは、複数の EBX® インスタンスで共有しないでください。

NFS、SMB、SAMBA などのプロトコルに基づくネットワーク接続ボリューム上に EBX® インデックスを保存しないでください。同じ制限が、オブジェクトベースのストレージ (Amazon S3、Google Cloud Storage、Azure Blob) またはオブジェクトストレージ上にマウントされたファイル システム (Amazon S3 ファイルゲートウェイ) に適用されます。これにより、安定性とパフォーマンスの問題が発生する可能性があります。より一般的な症状は、検索速度が遅くなり、開いているファイルが不足することです。

特にファイバーチャネル (FC) などの高性能テクノロジーが使用されている場合は、高性能ストレージエリアネットワーク (SAN) が適切なソリューションになります。

クラウドでは、EBX® は次のディスクでテストされています。

- Amazon Web Services (AWS): Amazon Elastic Block Store (EBS) 汎用 SSD ボリューム (gp3)
- Microsoft Azure: Azure Standard SSD ローカル冗長ストレージ (LRS)
- Google クラウド: Google バランス永続ディスク (pd-balanced)

ウイルス対策

ウイルス対策ソフトウェアは、ルートディレクトリでは使用しないでください。ファイルが隔離または削除されると、パフォーマンスに重大な影響を与え、インデックスリポジトリが破損する可能性があります。Lucene が使用する拡張子はいつでも変更される可能性があるため、拡張子による除外は不可能です。

ディスクのサイズ設定

ディスク サイズの経験則として、リレーショナルデータベースのテーブル G_BLK が占めるスペースの 10 倍を計画します。

Jakarta EE の展開

- はじめに
- ソフトウェアコンポーネント
- 埋め込みサードパーティライブラリ
- 必須のサードパーティライブラリ
 - データ圧縮ライブラリ
 - データベースドライバ
 - SMTP とメール
 - Secure Socket Layer (SSL)
 - Java メッセージサービス (JMS)
- Web アプリケーション
 - EBX® ビルトイン Web アプリケーション
 - EBX® アドオン Web アプリケーション
 - カスタム Web アプリケーション
- 展開の詳細
 - はじめに
 - ユーザーインターフェイスと Web アクセス
 - EBX® リポジトリのデータソース
 - メールセッション
 - JMS 接続ファクトリー
 - データサービス用の JMS
 - JAR ファイルスキャナー
- インストールに関する注意事項

はじめに

ここでは、Java アプリケーションサーバーでの TIBCO EBX® の展開仕様について詳しく説明します。サポートされるアプリケーションサーバーと固有の制限事項に関する具体的な情報については、[サポートされる環境](#)を参照してください。

安全な展開をナビゲートする方法については、[セキュリティのベストプラクティス](#)を参照してください。

ソフトウェアコンポーネント

EBX® は次のコンポーネントを使用します。

- ライブラリ `ebx.jar`
- 埋め込みおよび必須のサードパーティ Java ライブラリ
- EBX® ビルトイン Web アプリケーションおよびオプションのカスタム Web アプリケーション
- EBX® メイン構成ファイル
- EBX® リポジトリ
- デフォルトのユーザーとロールのディレクトリ。EBX® リポジトリ内またはユーザー認証用のサードパーティシステム (LDAP、RDBMS) に統合。

以下も参照してください。:

- [サポートされる環境](#)

埋め込みサードパーティライブラリ

EBX® の独立性と相互運用性を高めるために、独自のサードパーティライブラリが埋め込まれています。一部が変更され、競合回避されている場合も、これらは正式な Java API であるため、その他は変更しないでおく必要があります。

競合を引き起こす可能性があるのは次のとおりです。

- 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

サードパーティライブラリのバージョンまたは詳細については、[TIB_ebx_6.2.3_license.pdf](#) を参照してください。

これらのライブラリはすでに統合されているため、カスタム Web アプリケーションにこれらを新たに追加しないでください。そうでない場合は、リンケージエラーが発生する可能性があります。さらに、同じ理由で、[ebx.jar](#) ライブラリ以外にこれらを展開しないでください。

必須のサードパーティライブラリ

EBX® には、いくつかのサードパーティ Java ライブラリが必要です。これらのライブラリは展開し、[ebx.jar](#) のクラスローダーからアクセス可能にする必要があります。使用されるアプリケーションサーバーおよび Java ランタイム環境によっては、これらのライブラリがすでに存在する場合や、手動で追加する必要がある場合があります。

データ圧縮ライブラリ

EBX® バージョン 6.2.3 以降、個別の圧縮ライブラリの展開は不要になりました。以下の 2 つの圧縮モードがサポートされています。

- ネイティブは最適な高速モードですが、Linux/Windows の x86_64 で Java 21 以降を使用している場合にのみ利用できます。
- Java 17 を使用している場合は、純粋な Java 実装も利用できます。

使用している圧縮タイプを確認するには、[kernel.log](#) を開き、以下を探してください。

- [Compression: Java](#)、または
- [Compression: Native](#)

ネイティブ圧縮が使用されていない場合、その理由はログに表示されます。

圧縮パフォーマンスに問題がある場合は、`ebx.properties` で以下のパラメータを使用して調整してください。

- `ebx.compression.dictSize` は辞書サイズを制御し、デフォルトは 16 KB です。推奨範囲は 1 KB から 64 KB です。ただし、辞書が大きくなると以下のトレードオフが発生することにご注意ください。圧縮率 (対数進行) は向上しますが、圧縮速度は低下します (線形増加)。

注意:

この値は事前に設定する必要があります。EBX® が特定のテーブル定義の辞書を作成した後、別の辞書は作成しません。新しい値は、辞書を持たないテーブル、または新しいテーブル定義にのみ適用されます。

- `ebx.compression.level` は圧縮レベルを制御します。可能な値は FAST、BALANCED、HIGH (デフォルト) です。圧縮を高速化するにはレベルを下げますが、圧縮率は低下します。この設定は、次の圧縮操作にすぐに適用されます。この設定は解凍には目立った影響を与えません。

データベースドライバー

EBX® リポジトリにはデータベースが必要です。一般に、必要なドライバーは、データソースが使用されている場合は、データソースとともに構成されます。メイン構成ファイルで定義されているデータベースに応じて、次のいずれかのドライバーが必要です。どのデータベースを使用する場合でも、JDBC クライアントドライバーのバージョンは、データベースサーバーのバージョン以上である必要があることに注意してください。

H2	バージョン 2.2.224 が検証されました。H2 は運用環境ではサポートされていないことに注意してください。 https://www.h2database.com/ H2 リポジトリがバージョン 1.0 ドライバーで構築されている場合は、この v2 プロセスへの移行 を適用する必要があります。
Oracle JDBC	Oracle データベース 23ai は、最新のパッチセットアップで検証されています。

	データベースサーバーのバージョンと Java ランタイム環境のバージョンに応じて、使用するドライバーを決定します。JDK 17 または 21 で認定されたライブラリ <code>ojdbc17.jar</code> をダウンロードします。 Oracle データベース JDBC ドライバーのダウンロード。
SQL Server JDBC	すべての修正パッチとメンテナンスパッチが適用された SQL Server 2016 以降が検証されています。 古いバージョンではいくつかの問題が発生しているため、最新の JDBC ドライバーを使用してください。 たとえば、使用する Java ランタイム環境のバージョンに応じて、<code>mssql-jdbc-13.2.1.jre11.jar</code> ライブラリを含めます。 Microsoft JDBC Driver 13.2.1 for SQL Server (zip) のダウンロード。
PostgreSQL	PostgreSQL Java 8 が検証されています。 データベースサーバーおよび Java ランタイム環境用にリリースされた最新の JDBC ドライバーバージョン 4.2 を追加します。 PostgreSQL JDBC ドライバーのダウンロード。

以下も参照してください。:

- [EBX® リポジトリのデータソース](#)
- [EBX® リポジトリの構成](#)

SMTP とメール

使用されている Web アプリケーションサーバーに応じて、電子メール管理用のライブラリ

`Jakarta Mail API` がすでに提供されている場合もあれば、手動で追加する必要がある場合もあります。

EBX® には、この API のバージョン 2.1 と互換性のあるライブラリが必要です。構成の詳細については、[SMTP とメールの有効化と構成](#) を参照してください。

手動インストールを容易にするために、`jakarta.mail-2.0.5.jar` が提供され、`ebx.software/lib/lib-mail` ディレクトリに配置されています。

以下も参照してください。:

- [Jakarta Mail 2.1](#)

Secure Socket Layer (SSL)

SSL 機能のアクティベーション、構成、管理は、使用される Web アプリケーションサーバーと密接にリンクされています。したがって、適切な Web アプリケーションサーバーのドキュメントを参照する必要があります (例、[Tomcat SSL/TLS Configuration How-To](#))。

以下も参照してください。:

- [TIBCO EBX® メイン構成ファイル](#)

Java メッセージサービス (JMS)

JMS を使用する場合は、バージョン 3.1 以上が必要です。

Jakarta EE アプリケーションサーバーまたは Servlet/Java Server Pages (JSP) 実装のどちらが使用されているかに応じて、必要なライブラリは次のようになります。

- Jakarta EE (Enterprise Edition) に基づくアプリケーションサーバーの場合、必要な JMS プロバイダーライブラリはデフォルトで使用できます。詳細については、<https://jakarta.ee/specifications/platform/> を参照してください。

- ApacheTomcat などの JavaSE (Java Platform Standard Edition) を使用した Servlet/JavaServer Pages (JSP) 実装の場合、[Apache ActiveMQ](#) などの JMS プロバイダーライブラリの追加が必要な場合があります。詳細については、<https://www.oracle.com/java/technologies/java-se-glance.html> を参照してください。

注意:

EBX® では、サポートされる JMS モデルは Point-to-Point (PTP) のみです。PTP システムでは、メッセージのキューが許可されます。

以下も参照してください。:

- [TIBCO EBX® メイン構成ファイル](#)
- [Jakarta Messaging 3.1](#)

Web アプリケーション

EBX® は、追加するカスタム EBX® モジュール Web アプリケーションがない場合に直接展開可能な、事前パッケージ化された EAR を提供します。カスタム Web アプリケーションを EBX® モジュールとして展開する場合は、ビルトイン Web アプリケーションと同一レベルでパッケージ化されたカスタムモジュールを含む EAR を再構築することをお勧めします。

注意:

☐ パスコンテキストでの Web アプリケーションの展開はサポートされなくなりました。パスコンテキストは空にすることはできず、☐ と等しくすることもできません。さらに、異なる深さのパスでの Web アプリケーションの展開は推奨されません。すべての Web アプリケーションパスのコンテキストは、パスの深さを同一に設定する必要があります。

詳細については、この章の末尾の [EBX® EAR の再パッケージ化](#) に関する注意事項を参照してください。

EBX® ビルトイン Web アプリケーション

EBX® には、次のビルトイン Web アプリケーションが含まれています。

WEB アプリケーション名	説明	必須
ebx	起動時に初期化処理する EBX® エントリポイント。詳細については、 展開の詳細 を参照してください。	はい
ebx-root-1.0	EBX® ルート Web アプリケーション。EBX® を使用するアプリケーションでは、ルート Web アプリケーションを展開する必要があります。	はい
ebx-ui	EBX® ユーザーインターフェイス Web アプリケーション	はい
ebx-authentication	EBX® 認証アプリケーション	はい
ebx-manager	EBX® ユーザーインターフェイス Web アプリケーション	はい
ebx-dma	EBX® データモデルアシスタント。ユーザーインターフェイスによるデータモデルの作成を支援します。 **注意： ** データモデルアシスタントでは、 <code>ebx-manager</code> ユーザーインターフェイス Web アプリケーションを展開する必要があります。	はい
ebx-dataservices	EBX® データサービス Web アプリケーション。データサービスでは、 SOAP 操作 および Web サービス記述言語 WSDL 生成標準 を使用するか、 ビルトイン RESTful サービス を使用して、EBX® リポジトリとの外部インタラクションを許可します。 **注意： **EBX® Web サービスジェネレーターでは、 <code>ebx-</code>	はい

WEB アプリケーション名	説明	必須
	<code>manager</code> ユーザーインターフェイス Web アプリケーションを展開する必要があります。	
<code>ebx-ide</code>	EBX® 統合開発環境 Web アプリケーション。	はい
<code>ebx-assistant</code>	EBX® アシスタント Web アプリケーション。	はい
<code>ebx-hub</code>	EBX® Hub Web アプリケーション。	いいえ

EBX® アドオン Web アプリケーション

アドオンアプリケーションを展開することで、EBX® の動作を拡張できます。

以下も参照してください。:

- [TIBCO EBX® アドオンの管理](#)

カスタム Web アプリケーション

EBX® モジュールの要件に準拠するカスタム Web アプリケーションを展開することにより、EBX® の動作を拡張およびカスタマイズできます。

以下も参照してください。:

- [TIBCO EBX® モジュールのパッケージ化](#)
- [モジュールの展開解除宣言](#)

展開の詳細

はじめに

このセクションでは、「`ebx`」Web アプリケーションの展開に使用可能なさまざまなオプションについて説明します。これらのオプションは、展開記述子 (`WEB-INF/web.xml`) で使用可能で、メイン構成ファイルで定義されたプロパティによって補完されます。

注意:

JBoss アプリケーションサーバーの場合、未使用のリソースはすべて `WEB-INF/web.xml` 展開記述子から削除する必要があります。

以下も参照してください。:

- TIBCO EBX® メイン構成ファイル
- サポートされるアプリケーションサーバー

ユーザーインターフェイスと Web アクセス

Web アプリケーション「`ebx`」(`ebx.war` としてパッケージ化) には、初期化を処理し、EBX® Web ツールの唯一のユーザーインターフェイスのエントリポイントとして機能する `FrontServlet` サブレットが含まれています。

FrontServlet の展開記述子の構成

Web アプリケーション「`ebx`」のファイル `WEB-INF/web.xml` で、次のエレメントを `FrontServlet` 用に構成する必要があります。

`/web-app/servlet/load-on-startup`

EBX® の起動時に `FrontServlet` が確実に初期化されるようにするには、`web.xml` 展開記述子でエレメント `<load-on-startup>1</load-on-startup>` を指定する必要があります。

`/web-app/servlet-mapping/url-pattern`

`FrontServlet` はパス「`/`」にマッピングする必要があります。

FrontServlet 用アプリケーションサーバーの構成

`FrontServlet` には、`ServletContext` など他のコンテキストへのアクセス許可が必要です。

たとえば、Tomcat では、この構成は、構成ファイル `server.xml` の属性 `crossContext` を使用して次のように実行されます。

```
<Context path="/ebx" docBase="(...)" crossContext="true"/>
```

EBX® リポジトリのデータソース

注意:

EBX® のメイン構成でプロパティ `ebx.persistence.url` が指定されている場合、以下の環境エントリは EBX® ランタイムにより無視されます。このオプションは、便宜上提供されているだけです。完全構成可能なデータソースを使用することを常にお勧めします。特に、接続プールのサイズは、同時ユーザーの数に応じて設定する必要があります。このプロパティの詳細については、[EBX® リポジトリの構成](#)を参照してください。

EBX® の JDBC データソースは、「`ebx`」Web アプリケーションの展開記述子 `WEB-INF/web.xml` で次のように指定されます。

予約済みリソース名	デフォルトの JNDI 名	説明
<code>jdbc/EBX_REPOSITORY</code>	JBoss : <code>java:/EBX_REPOSITORY</code>	EBX® リポジトリの JDBC データソース Java タイプ : <code>javax.sql.DataSource</code>

以下も参照してください。:

- [EBX® リポジトリの構成](#)

- データベースアクセスとユーザー権限のルール

メールセッション

注意:

EBX® のメイン構成で `ebx.mail.activate` が true に設定されていない場合、またはプロパティ `ebx.mail.smtp.host` が指定されている場合は、以下の環境エントリは、EBX® ランタイムにより無視されます。これらのプロパティの詳細については、EBX® メイン構成プロパティの [SMTP](#) を参照してください。

SMTP とメールは、次のように「`ebx`」Web アプリケーションの展開記述子 `WEB-INF/web.xml` で宣言されます。

予約済みリソース名	デフォルトの JNDI 名	説明
<code>mail/EBX_MAIL_SESSION</code>	JBoss : <code>java:/EBX_MAIL_SESSION</code>	EBX® からメールを送信するために使用される Java メールセッション Java タイプ : <code>jakarta.mail.Session</code>

JMS 接続ファクトリー

注意:

EBX® メイン構成がプロパティ `ebx.jms.activate` で JMS を有効にしない場合、以下の環境エントリは EBX® ランタイムにより無視されます。このプロパティの詳細については、EBX® メイン構成プロパティの [JMS](#) を参照してください。

JMS 接続ファクトリーは、次のように「`ebx`」Web アプリケーションの展開記述子 `WEB-INF/web.xml` で宣言されます。

予約済みリソース名	デフォルトの JNDI 名	説明	必須
<code>java:/</code> <code>EBX_JMSConnectionFactory</code>	JBoss : <code>java:/</code> <code>EBX_JMSConnectionFactory</code>	アプリケーションサーバーの運用環境で構成された JMS プロバイダーとの接続を作成するために EBX® によって使用される JMS 接続ファクトリー。Java タイプ： <code>jakarta.jms.ConnectionFactory</code>	はい

注意:

JNDI 機能を備えた WildFly JBoss アプリケーションサーバーに展開する場合は、必要なすべてのリソース名を JNDI 名に追加マッピングするために `EBX.ear` を更新する必要があります。

データサービス用の JMS

デフォルトの HTTP の代わりに JMS を使用するようデータサービスを構成するには、**JMS 接続ファクトリー**と、「`ebx`」Web アプリケーションの `WEB-INF/web.xml` 展開記述子で宣言された次のキューを構成する必要があります。これは、データサービス用の JMS を構成するための唯一の方法です。

ヘッダーフィールド `JMSReplyTo` が定義されている場合、SOAP リクエストを受信すると、オプションで SOAP 応答が返されます。この場合、フィールド `JMSCorrelationID` および `JMSType` は保持されます。

関連する EBX® の主な構成プロパティの詳細については、**JMS** を参照してください。

注意:

EBX® のメイン構成がプロパティ `ebx.jms.activate` を介して JMS をアクティブ化しない場合、以下の環境エントリは EBX® ランタイムによって無視されます。このプロパティの詳細については、EBX® のメイン構成プロパティの **JMS** を参照してください。

予約済みリソース名	デフォルトの JNDI 名	説明	必須
jms/ EBX_QueueIn	JBoss : java:/ jms/EBX_QueueIn	他のアプリケーションによって EBX® に送信される着信 SOAP リクエストの JMS キュー。Java タイプ: jakarta.jms.Queue	いいえ
jms/ EBX_QueueFailure	JBoss : java:/ jms/ EBX_QueueFailure	エラー時の JMS キュー。これには、エラー発生時の受信 SOAP リクエストが含まれています。これにより、必要に応じてこれらのメッセージを再生できます。Java タイプ: jakarta.jms.Queue **注意:**このプロパティの読み取りには、メイン構成で、プロパティ ebx.jms.activate.queueFailure によりエラー時のキューも有効化する必要があります。これらのプロパティの詳細については、EBX® メイン構成プロパティの JMS を参照してください。	いいえ

JAR ファイルスキャナー

Web アプリケーションサーバーの起動を高速化するには、JAR ファイルスキャナーの構成を変更して、少なくとも `ebx.jar` および `ebx-addons.jar` ライブラリを除外する必要があります。

たとえば、Tomcat では、`catalina.properties` ファイルの `tomcat.util.scan.StandardJarScanFilter.jarsToSkip` プロパティでこれを実行する必要があります。

インストールに関する注意事項

EBX® は、Servlet 6.0 以上をサポートする任意の Jakarta EE 10 以降のアプリケーションサーバーに導入できます。導入とインストールに関する注意事項については、次のドキュメントを参照してください。

- [Tomcat 10.1.X のインストールに関する注意事項](#)
- [JBoss EAP 8.0.X のインストールに関する注意事項](#)
- [WebSphere Application Server Liberty 23.X のインストールに関する注意事項](#)

注意:

- Jakarta EE 10 アプリケーションサーバーの EBX® インストールに関する注意事項では、各アプリケーションサーバーのネイティブドキュメントが置き換えられることはありません。
- 上記は一般的なインストールの推奨ではありません。インストールプロセスは、技術環境、アプリケーションの相互化、配信プロセス、組織の決定などのアーキテクチャ上の決定によって異なります。
- これらの例では、追加の EBX® モジュールは展開されていません。カスタムモジュールまたはアドオンモジュールを展開するためのベストプラクティスは、他の EBX® モジュールと同じレベルの Web アプリケーションとしてモジュールを使用して EAR を再構築することです。Web アプリケーションは、Java™ 2 Platform Enterprise Edition 仕様 v1.4 で指定されているように、クラスパスの依存関係を宣言する必要があります。

J2EE.8.2 オプションのパッケージサポート

(...)

JAR フォーマットのファイル (例、JAR ファイル、WAR ファイル、RAR ファイル) では、参照 JAR ファイルのマニフェストファイルの Class-Path ヘッダーで参照される JAR ファイルに名前を付けることで、JAR ファイルを参照できます。参照される JAR ファイルの名前は、参照 JAR ファイルの URL に相対的な URL を使用して付けられます。マニフェストファイルは、JAR ファイルでは META-INF/MANIFEST.MF という名前です。マニフェストファイルの Class-Path エントリのフォーマットは次のとおりです。

Class-Path: list-of-jar-files-separated-by-spaces

「工業化」プロセスでは、カスタム EBX® モジュール、EBX® Web アプリケーション、および必要なすべての共有ライブラリを使用して、EAR を自動的に構築するスクリプトの開発を強くお勧めします。

- 予期しない動作を回避するため、クラスローディングシステム内の `ebx.jar` または他のライブラリの重複を回避するガイドラインに従ってください。

EBX® のインストールに関する注意事項

- [JBoss EAP 8.0.X のインストールに関する注意事項](#)
- [WebSphere Application Server Liberty 23.X のインストールに関する注意事項](#)
- [Tomcat 10.1.X のインストールに関する注意事項](#)

JBoss EAP 8.0.X のインストールに関する注意事項

- [概要](#)
- [要件](#)
- [JBoss Application Server のインストール](#)
- [EBX® ホームディレクトリの構成](#)
- [JBoss Application Server と Java 仮想マシンの構成](#)
- [JNDI エントリの構成](#)
- [データソースと JDBC プロバイダーの構成](#)
- [EBX.ear アプリケーションの更新](#)
- [EBX.ear アプリケーションの展開](#)
- [EBX® アプリケーションの開始](#)

概要

注意:

- この章では、JBoss Application Server での TIBCO EBX® のクイックインストール例について説明します。
- このアプリケーションサーバーのドキュメントを置き換えるものではありません。
- インストールプロセスは、技術環境、アプリケーションの相互化、配信プロセス、組織の決定などのアーキテクチャ上の決定によって決定されるため、これらは一般的なインストールの推奨事項ではありません。
- EBX® に必要なコンポーネントの詳細な説明は、[Jakarta EE の展開](#) の章に記載されています。
- 予期しない動作を回避するために、クラスローディングシステム内の `ebx.jar` またはその他のライブラリの重複を避けることがガイドラインとなります。
- 安全な展開をナビゲートする方法については、[セキュリティのベストプラクティス](#) を参照してください。

要件

- Java SE 17 LTS
- JBoss Application Server EAP 8.0
- データベースと JDBC ドライバー
- EBX® CD
- EBX® ライセンスキー
- EBX® の追加モジュールに CDI 機能がないこと (CDI は自動的に無効になるため)

以下も参照してください。:

- [サポートされる環境](#)

JBoss Application Server のインストール

このクイックインストール例は、UNIX オペレーティングシステムに対して実行されます。

1. JBoss EAP 8.0.X `インストーラー` jar バージョンを次の場所からダウンロードします。

<https://developers.redhat.com/products/eap/download/>

2. `java -jar` コマンドラインを使用して `インストーラー` を実行します。

インストールの詳細については、[ドキュメント](#)を参照してください。

3. 標準インストールを実行します。
 - a. 言語を選択して をクリックします。
 - b. ライセンスに同意し、 をクリックします。
 - c. インストールパスを選択し、 をクリックします。
 - d. をそのままにして、 をクリックします。
 - e. 、 を入力し、 をクリックします。
 - f. で をクリックします。
 - g. で をクリックします。
 - h. で、選択内容をそのままにして、 をクリックします。
 - i. 「処理が終了しました」と表示されたら、 をクリックします。
 - j. JBoss EAP 8.0 インストールルートディレクトリに「インストールスクリプトとプロパティファイル」を生成します。
 - k. をクリックします。

EBX® ホームディレクトリの構成

1. `EBX_HOME` ディレクトリを作成します (例: `/opt/ebx/home`)。
2. `EBX® CD`、`ebx.software/files/ebx.properties` ファイルから `EBX_HOME` にコピーします。この例では、次のファイルです。

`/opt/ebx/home/ebx.properties`。

3. `EBX_HOME` に `ebx-license.properties` ファイルを次のように作成します。

`/opt/ebx/home/ebx-license.properties`

ファイルの内容:

```
#####

## EBX® ライセンスキー
```

```
##      EBX® がファイルを書き込み可能であることを確認してください

##      UI ライセンスキーウィザードプロセスを使用した更新を有効にするため。

#####

ebx.license=<your_license_key>
```

- 必要に応じて、`ebx.properties` ファイルを編集してデフォルトのデータベースをオーバーライドします。デフォルトでは、スタンドアロンの H2 データベースが定義されています。サポートされている他のデータベースでは、プロパティキー `ebx.persistence.factory` のコメントを解除し、`h2.standalone` のプロパティキーにコメントを付ける必要があります。

JBoss Application Server と Java 仮想マシンの構成

- `<JBOSS_HOME>/bin` にある `standalone.conf` 構成ファイルを開きます (または、サーバーをサービスとして実行する場合は、`<JBOSS_HOME>/bin/init.d` にある `jboss-eap.conf` ファイルを開きます)。
- 「`ebx.properties`」および「`ebx.home`」プロパティを、それぞれ `ebx.properties` ファイルのパスと `EBX_HOME` ディレクトリのパスで設定された「`JAVA_OPTS`」環境変数に追加します。
- 「`JBOSS_MODULES_SYSTEM_PKGS`」環境変数を次のように設定します。

```
JBOSS_MODULES_SYSTEM_PKGS="org.jboss.byteman,net.jpountz"
```

- `<JBOSS_HOME>/bin` にある `standalone.sh` スクリプトファイルを開きます。
- 次のように、フォアグラウンドおよびバックグラウンド実行の起動コマンドオプションを置き換えます。

```
if [ "$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=\ "$JBASS_BASE_DIR" \ \

"$SERVER_OPTS" "&"

...

fi

```

JNDI エントリの構成

1. `<JBASS_HOME>/standalone/configuration` にある `standalone-full.xml` ファイルを開きます。
2. `messaging-activemq` サブシステムの `server` タグに次の行を追加します。

```

<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"/>

```

注意: `connection-factory` エレメントの `connectors` 属性値を定義する必要があります。コネクタの種類は環境のインフラに強く依存するため、デフォルトの構成を提供することはできません。

詳細については、以前のバージョンのドキュメントで [メッセージングの構成](#) を参照してください。

3. `mail` サブシステムに少なくとも次の行を追加します。

```

<mail-session name="mail" debug="false" jndi-name="java:/
EBX_MAIL_SESSION"/>

```

データソースと JDBC プロバイダーの構成

1. JBoss Server の起動後、「-connect」または「-c」引数を使用せずに管理 CLI を実行します。
2. 「module add」管理 CLI コマンドを使用して、新しいコアモジュールを追加します。以下は PostgreSQL 構成の事例です。

```
module add \

--name=org.postgresql \

--resources=<PATH_TO_JDBC_JAR> \

--dependencies=wildflyee.api
```

3. 「connect」管理 CLI コマンドを使用して、実行中のインスタンスに接続します。
4. JDBC ドライバーを登録します。管理対象ドメインで実行する場合は、コマンドの前に必ず「/profile=<PROFILE_NAME>」を付けてください。PostgreSQL 構成の設定例を次に示します。

```
/subsystem=\

datasources/jdbc-driver=\

    postgresql:add(\

        driver-name=postgresql,\

        driver-module-name=org.postgresql,\

        driver-xa-datasource-class-name=org.postgresql.xa.PGXADatasource\

    )
```

5. 適切な引数値を指定して、「data-source add」コマンドを使用してデータソースを定義します。PostgreSQL 構成の例：

```
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>
```

注意:

データソースは、EBX® メイン構成ファイルから指定することもできます。詳細については、[EBX® リポジトリのデータソース](#) を参照してください。

EBX.ear アプリケーションの更新

1. EBX® の CD から `ebx.software/webapps/ear-packaging/EBX.ear` ファイルを作業ディレクトリにコピーします。
2. アプリケーション固有の必要なサードパーティライブラリや追加の Web モジュールを追加するために、ear アーカイブを解凍します。

メール：詳細については、[SMTP とメール](#)を参照してください。

SSL：詳細については、[Secure Socket Layer \(SSL\)](#) を参照してください。

JMS：詳細については、[Java メッセージサービス \(JMS\)](#) を参照してください。

3. 追加された Web モジュールに応じて、`/META-INF/application.xml` および `/META-INF/jboss-deployment-structure.xml` ファイルを更新します。
4. ear のアーカイブを新たに圧縮します。

EBX.ear アプリケーションの展開

1. EBX.ear を `JBOSS_HOME/standalone/deployments` ディレクトリにコピーします。

EBX® アプリケーションの開始

1. JBoss Application Server の起動後、`<JBOSS_HOME>/bin/standalone.sh -c standalone-full.xml` コマンドラインまたはサービスコマンドを使用して、ブラウザーに次の URL を入力して EBX® Web アプリケーションを実行します。 <http://localhost:8080/ebx/>

2. EBX® ウィザードは、最初の起動時に初期リポジトリのデフォルトのプロパティを構成するのに役立ちます。

WebSphere Application Server Liberty 23.X のインストールに関する注意事項

- 概要
- 要件
- WebSphere Application Server Liberty のインストール
- EBX® ホームディレクトリの構成
- EBX® およびサードパーティライブラリの展開
- JNDI エントリの構成
- データソースと JDBC プロバイダーの構成
- Java 仮想マシンの構成
- EBX® アプリケーションの展開
- EBX® アプリケーションの開始

概要

注意:

- この章では、WebSphere Application Server Liberty での TIBCO EBX® のクイックインストール例について説明します。
- このアプリケーションサーバーのドキュメントを置き換えるものではありません。
- インストールプロセスは、技術環境、アプリケーションの相互化、配信プロセス、組織の決定などのアーキテクチャ上の決定によって決定されるため、これらは一般的なインストールの推奨事項ではありません。
- EBX® に必要なコンポーネントの詳細な説明は、[Jakarta EE 展開](#) の章に記載されています。
- 予期しない動作を回避するためのガイドラインは、`ebx.jar`、またはクラスローディングシステム内の他のライブラリの重複を避けることです。
- 以下の説明では、変数名 `<WLP_HOME>` を使用して WebSphere Application Server Liberty インストールディレクトリを参照し、そこから相対パスが解決されます。

- 安全な展開をナビゲートする方法については、[セキュリティのベストプラクティス](#)を参照してください。

要件

- Java SE 17 または 21 LTS。IBM Semeru、または Eclipse Temurin Java ディストリビューションの使用をお勧めします。
- Jakarta EE 10 用の WebSphere Application Server Liberty 23.0.0.12 以上
- データベースと JDBC ドライバー
- EBX® CD
- EBX® ライセンスキー
- EBX® の追加モジュールには CDI 機能はありません (CDI は自動的に無効になるため)

以下も参照してください。:

- [サポートされる環境](#)

WebSphere Application Server Liberty のインストール

このクイックインストールの例は、Linux オペレーティングシステムに対して実行されます。

1. WebSphere Application Server Liberty 23.x の最新バージョン (zip ディストリビューション) を以下からダウンロードします。

<https://www.ibm.com/support/pages/230012-websphere-application-server-liberty-230012>.

次の例では、*Liberty with Jakarta EE 10 Web Profile* の zip ディストリビューションを使用します。

2. このパッケージのコンテンツをローカルマシンのディレクトリに抽出します。
3. 次のコマンドを実行して、新しいアプリケーションサーバー (例: 「ebxServer」) を作成します。

```
<WLP_HOME>/wlp/bin/server create ebxServer
```

4. `<WLP_HOME>/usr/servers/<SERVER_NAME>/server.xml` ファイルを編集します。この例では、`<SERVER_NAME>` は「ebxServer」です。

- `<feature>webProfile-10.0</feature>` タグを少なくとも次の行に置き換えます。

```
<feature>mail-2.1</feature>

<feature>jdbc-4.2</feature>

<feature>jndi-1.0</feature>

<feature>servlet-6.0</feature>
```

- 次の行を `<server>` タグに追加します。

```
<webContainer deferServletLoad="false"
  allowQueryParamWithNoEqual="true"/>
```

注意:

`deferServletLoad` 属性は `false` に設定する必要があります。これは、EBX® ではアプリケーションサーバーの起動時にサーブレットを初期化する必要があるためです。デフォルトでは、WebSphere Application Server Liberty は、関連付けられた Web アプリケーションに対するリクエストが受信されるまで、サーブレットのロードを延期します。詳細については、[Specifying when servlets are loaded and initialized](#) を参照してください。

注意:

Servlet API の仕様では、リクエストパラメーターの値は `文字列` として返す必要があります、パラメーターが存在しない場合は `null` となるため、`allowQueryParamWithNoEqual` 属性を `true` に設定する必要があります。したがって、EBX® では、既存のリクエストパラメーターに `null` 以外の値を期待します。

EBX® ホームディレクトリの構成

1. `EBX_HOME` ディレクトリを作成します (例: `/opt/ebx/home`)。

2. *EBX® CD*、`ebx.software/files/ebx.properties` ファイルから *EBX\HOME* にコピーします。この例では、次のファイルです。

```
/opt/ebx/home/ebx.properties
```

3. *EBX\HOME* に `ebx-license.properties` ファイルを以下のように作成します。

```
/opt/ebx/home/ebx-license.properties
```

ファイルの内容:

```
#####

## 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. 必要に応じて、`ebx.properties` ファイルを編集してデフォルトのデータベースをオーバーライドします。デフォルトでは、スタンドアロンの H2 データベースが定義されています。サポートされている他のデータベースでは、プロパティキー `ebx.persistence.factory` のコメントを解除し、`h2.standalone` のプロパティキーにコメントを付ける必要があります。

EBX® およびサードパーティライブラリの展開

1. *EBX\LIB* ディレクトリを作成します (例: `/opt/ebx/home/lib`)。
2. *EBX® CD* または他のソースからサードパーティライブラリを *EBX\LIB* ディレクトリにコピーします。この例と PostgreSQL データベースの場合、次のようになります。

```
postgresql-X.X.X-driver.jar (EBX® CD 以外のソースから取得)
```

これらのコンポーネントの詳細な説明は、[Jakarta EE 展開](#) の章に記載されています。これらのコンポーネントがクラスローディングシステムにすでに展開されている場合は、複製する必要はありません (例: このサーバーでは `mail-2.1` 機能が有効化されているため、Jakarta Mail API とランタイム実装のサポートが提供されます)。

JNDI エントリの構成

1. WebSphere Application Server Liberty での JMS 構成については、[Configuring JMS connection factories](#) を参照してください。
2. WebSphere Application Server Liberty での Java Mail 構成については、[Liberty での JavaMail の管理](#) を参照してください。

データソースと JDBC プロバイダーの構成

1. JDBC ドライバライブラリを作成するには、`<WLP_HOME>/usr/servers/<SERVER_NAME>/server.xml` ファイルを編集して、次の行を `<server>` タグに追加します。

```
<library id="jdbcDriver">

  <file name="{EBX_LIB}/To be completed"/>

</library>
```

注意： `file` タグの `name` 属性値を完成させる必要があります。この例では、`{EBX_LIB}/postgresql-X.X.X-driver.jar` にする必要があります。

2. JDBC データソースを作成するには、同じファイルと同じ XML タグを編集して、次の行を追加します。

```
<dataSource id="ebxRepository" jndiName="jdbc/EBX_REPOSITORY">

  <jdbcDriver libraryRef="jdbcDriver"/>

  <properties.toBeCompleted/>

</dataSource>
```

注意： `properties` タグを完成させる必要があります。タグの種類は使用する JDBC ドライバーに大きく依存するため、デフォルトの構成を提供することはできません。PostgreSQL ドライバーの場合、次のようになります。

```
<properties.postgresql databaseName="EBXDB" serverName="localhost"
  portNumber="5432" user="ebx" password="ebx"/>
```

注意:

WebSphere Application Server Liberty の `server.xml` ファイルで JDBC 接続のパスワードを非表示にするには、認証データ (authData)、データソース (dataSource) からの `containerAuthData`、および Liberty とそのアプリケーションの保護を参照してください。

注意:

データソースは、EBX® メイン構成ファイルから指定することもできます。詳細については、[EBX® リポジトリのデータソース](#) を参照してください。

Java 仮想マシンの構成

1. 次の行を `<WLP_HOME>/usr/servers/<SERVER_NAME>/server.env` ファイルに追加して、サーバーの環境変数を設定します。

```
EBX_HOME="<path_to_the_directory_ebx_home>"

EBX_LIB="<path_to_the_directory_ebx_lib>"

JAVA_HOME="<path_to_the_java_home>"
```

2. `<WLP_HOME>/usr/servers/<SERVER_NAME>/` ディレクトリの下に新しい `jvm.options` ファイルを作成して、Java オプションを設定します。このファイルには、少なくとも次の行が含まれます。

```
-Debx.home="${EBX_HOME}"

-Debx.properties="${EBX_HOME}/ebx.properties"
```

EBX® アプリケーションの展開

1. `<EBX_HOME>/ear/` ディレクトリを作成します。
2. EBX® CD の `ebx.software/webapps/ear-packaging/EBX.ear` を `<EBX_HOME>/ear/` ディレクトリにコピーします。この例では、次のようになります。

```
/opt/ebx/home/ear/EBX.ear
```

EBX® アプリケーションの開始

1. WebSphere Application Server Liberty の起動後、`<WLP_HOME>/wlp/bin/server start ebxServer` コマンド行を使用して、ブラウザに次の URL を入力して EBX® Web アプリケーションを実行します。

`http://localhost:9080/ebx/`
2. 最初の起動時に、EBX® ウィザードは初期リポジトリのデフォルトのプロパティを構成するのに役立ちます。

Tomcat 10.1.X のインストールに関する注意事項

- 概要
- 要件
- Tomcat Application Server のインストール
- EBX® ホームディレクトリの構成
- Tomcat Application Server と Java 仮想マシンの構成
- EBX® およびサードパーティライブラリの展開
- EBX® Web アプリケーションの展開
- EBX® アプリケーションの開始

概要

注意:

- この章では、Tomcat Application Server での TIBCO EBX® のクイックインストール例について説明します。
- このアプリケーションサーバーのドキュメントを置き換えるものではありません。
- インストールプロセスは、技術環境、アプリケーションの相互化、配信プロセス、組織の決定などのアーキテクチャ上の決定によって決定されるため、これらは一般的なインストールの推奨事項ではありません。
- EBX® に必要なコンポーネントの詳細な説明は、[Jakarta EE の展開](#) の章に記載されています。
- 予期しない動作を回避するために、クラスローディングシステム内の `ebx.jar` またはその他のライブラリの重複を避けることがガイドラインとなります。

- 以下の説明では、変数名 `$CATALINA_HOME` を使用して Tomcat インストールディレクトリを参照し、そこからほとんどの相対パスが解決されるようにします。ただし、`$CATALINA_BASE` ディレクトリが複数インスタンス構成に設定されている場合は、これらの各参照に使用する必要があります。
- 安全な展開をナビゲートする方法については、[セキュリティのベストプラクティス](#)を参照してください。

要件

- Java SE 17 LTS
- Apache Tomcat 10.1.X。EBX® バージョン 6.2.0 以降は Tomcat 11 と互換性がありません。
- データベースと JDBC ドライバー
- EBX® CD
- EBX® ライセンスキー

以下も参照してください。:

- [サポートされる環境](#)

Tomcat Application Server のインストール

1. Tomcat 10.1.X コアバイナリディストリビューションを次の場所からダウンロードします。

<https://tomcat.apache.org/download-10.cgi>,

2. インストーラーを実行するか、アーカイブを抽出して、デフォルトのオプションを使用して標準インストールを実行します。

EBX® ホームディレクトリの構成

1. `EBX_HOME` ディレクトリを作成します (例: `C:\EBX\home`、または `/home/ebx`)。
2. `EBX® CD` から `ebx.software/files/ebx.properties` ファイルを `EBX_HOME` にコピーします。この例では、次のファイルです。

`C:\EBX\home\ebx.properties`、または `/home/ebx/ebx.properties`

3. 以下に示すように、`EBX_HOME` に `ebx-license.properties` ファイルを作成します。

C:\EBX\home\ebx-license.properties、または /home/ebx/ebx-license.properties

ファイルの内容：

```
#####

## EBX® ライセンスキー

##      UI ライセンスキーウィザードプロセスを使用した更新を有効にするには、EBX® が
      ファイルを書き込み可能であることを確認してください。

#####

ebx.license=<your_license_key>
```

4. 必要に応じて、`ebx.properties` ファイルを編集してデフォルトのデータベースをオーバーライドします。デフォルトでは、スタンドアロンの H2 データベースが定義されています。サポートされている他のデータベースでは、プロパティキー `ebx.persistence.factory` のコメントを解除し、`h2.standalone` のプロパティキーにコメントを付ける必要があります。

Tomcat Application Server と Java 仮想マシンの構成

1. `$CATALINA_HOME/conf/server.xml` (または `$CATALINA_BASE/conf/server.xml`) ファイルを変更します。
 - a. `Connector` エlementに次の属性を追加します。
 - `encodedSolidusHandling="passthrough"`
 - `allowBackslash="true"`
 - b. `<Host>` Elementの `unpackWARs` オプションを `true` に設定します。

注意:

`unpackWARs` が *not* `true` に設定されていない場合、TIBCO EBX® は Web アプリケーションクラスをスキャンできず、適切に起動できません。このオプションでは `true` のみがサポートされています。

- c. `<Host>` Elementに次の行を追加します。

```
<Context path="/ebx" crossContext="true" docBase="ebx.war"/>
```

この例では、次のようになります。

```
<Connector encodedSolidusHandling="passthrough" allowBackslash="true" .../>
```

```
<Host unpackWARs="true" name=...>
```

```
... ..
```

```
<Context path="/ebx" crossContext="true" docBase="ebx.war"/>
```

```
... ..
```

```
</Host>
```

2. `$CATALINA_HOME/conf/catalina.properties` (または `$CATALINA_BASE/conf/catalina.properties`) ファイルの `tomcat.util.scan.StandardJarScanFilter.jarsToSkip` プロパティに次の行を追加します。

```
ebx.jar,\
```

```
ebx-addons.jar,\
```

3. Java 仮想マシンのプロパティを構成します。

- Windows のコマンドプロンプト起動の場合

`setenv.bat` ファイルを `$CATALINA_HOME\bin` または `$CATALINA_BASE\bin` に作成して、環境変数を設定します。このファイルには、少なくとも次の行が含まれます。

```
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%
```

ここで、`<$CATALINA_HOME_or_$CATALINA_BASE>` は、設定されている場合、`%CATALINA_HOME%` または `%CATALINA_BASE%` に置き換える必要があります。それ以外の場合は、Tomcat のインストールディレクトリのパスに置き換える必要があります。

- Tomcat をサービスとしてインストールした Windows ユーザーの場合

Tomcat サービスマネージャー GUI (タブ) を使用して Java オプションを設定します。

GUI の フィールドで、必ず別の行にオプションを設定してください。

```
-Debx.home="<path_to_the_directory_ebx_home>"

-Debx.properties="<path_to_the_directory_ebx_home>\ebx.properties"
```

ここで、<\$CATALINA_HOME_or_ \$CATALINA_BASE> は、設定されている場合、`%CATALINA_HOME%` または `%CATALINA_BASE%` に置き換える必要があります。それ以外の場合は、Tomcat のインストールディレクトリのパスに置き換える必要があります。

- UNIX シェル起動の場合

`setenv.sh` ファイルを `$CATALINA_HOME/bin` または `$CATALINA_BASE/bin` に作成して、環境変数を設定します。このファイルには、少なくとも次の行が含まれます。

```
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}"
```

<\$CATALINA_HOME_or_ \$CATALINA_BASE> は、設定されている場合、`$`
`{CATALINA_HOME}` または `{CATALINA_BASE}` に置き換える必要があります。それ以外の場合は、Tomcat のインストールディレクトリのパスに置き換える必要があります。

注意：EBX® の起動に使用されるアカウントには、`EBX\_HOME` ディレクトリに対して作成/更新/削除の権限が必要です。

注意:

<path_to_the_directory_ebx_home> は、`ebx.properties` をコピーしたディレクトリです。この例では、`C:\EBX\home`、または `/home/ebx` です。

EBX® およびサードパーティライブラリの展開

1. サードパーティライブラリを EBX® CD から `$CATALINA_HOME/lib/` (または `$CATALINA_BASE/lib/`) ディレクトリにコピーします。今回の例では、以下のようになります。

`$CATALINA_HOME/lib/jakarta.mail-2.0.5.jar` は `ebx.software/lib/lib-mail` ディレクトリから取得されます。

`$CATALINA_HOME/lib/h2-2.2.224.jar` (デフォルトの永続化ファクトリー) は `ebx.software/lib/lib-h2` ディレクトリから取得されます。

これらのコンポーネントの詳細な説明は、[ソフトウェアコンポーネント](#)の章に記載されています。もちろん、これらのコンポーネントがすでにクラスローディングシステムに展開されている場合は、それらを重複して展開する必要はありません。

2. EBX® CD から `ebx.software/lib/ebx.jar` ファイルを `$CATALINA_HOME/lib/` (または `$CATALINA_BASE/lib/`) ディレクトリにコピーします。この例では、次のようになります。

`$CATALINA_HOME/lib/ebx.jar`

EBX® Web アプリケーションの展開

1. EBX® CD から `ebx.software/webapps/wars-packaging` にある war ファイルを `$CATALINA_HOME/webapps/` (または `$CATALINA_BASE/webapps/`) ディレクトリにコピーします。この例では、次のようになります。

`$CATALINA_HOME/webapps/ebx.war` : EBX® アプリケーションの初期化サーブレット

`$CATALINA_HOME/webapps/ebx-assistant.war` : AI アシスタント Web アプリケーション

`$CATALINA_HOME/webapps/ebx-authentication.war` : 中央認証アプリケーション

`$CATALINA_HOME/webapps/ebx-dataservices.war` : データサービス Web アプリケーション

`$CATALINA_HOME/webapps/ebx-dma.war` : データモデルアシスタント Web アプリケーション

`$CATALINA_HOME/webapps/ebx-hub.war` : Hub Web アプリケーション

`$CATALINA_HOME/webapps/ebx-manager.war` : マスターデータ管理 Web アプリケーション

`$CATALINA_HOME/webapps/ebx-root-1.0.war` : データモデルに共通のデフォルトモジュールの提供

`$CATALINA_HOME/webapps/ebx-ui.war` : ユーザーインターフェイス Web アプリケーション

`$CATALINA_HOME/webapps/ebx-ide.war` : 統合開発環境 Web アプリケーション

EBX® アプリケーションの開始

1. Tomcat の起動後、ブラウザーに次の URL を入力して EBX® Web アプリケーションを実行します。<http://localhost:8080/ebx/>
2. 最初の起動時に、EBX® ウィザードは、初期リポジトリのデフォルトのプロパティを構成するのに役立ちます。

EBX® メタデータ管理アプリケーションのインストールに関する注意事項

- [EBX® Container Edition のインストールに関する注意事項](#)
- [オンプレミス Tomcat 10.1.X のインストールに関する注意事項](#)

EBX® Container Edition のインストールノート

- [概要](#)
- [必須 EBX® アドオン](#)
- [EBX® メタデータ分類子データベース](#)
 - [要件](#)
 - [カスタマイズと実行](#)
- [EBX® メタデータエージェント](#)
- [検証](#)
- [制限事項](#)

概要

EBX® メタデータ管理アプリケーションは、TIBCO EBX® Container Edition の一部として利用できます。

必須 EBX® アドオン

対話型またはバッチインストーラーからメタデータを選択すると、以下の EBX® アドオンが Docker イメージに自動的にインストールされます。

- TIBCO EBX® Digital Asset Manager アドオン (DAMA)
- TIBCO EBX® Data Exchange アドオン (DINT)
- TIBCO EBX® Data Model and Data Visualization アドオン (DMDV)
- TIBCO EBX® Insight アドオン (DPRA)
- TIBCO EBX® Information Search アドオン (TESE)

注:

イメージの構築および EBX® メタデータ管理アプリケーションの組み込みについては、インストーラーの指示を参照してください。

EBX® メタデータ分類子データベース

要件

分類子データベースは、SQLite、PostgreSQL、Microsoft SQL Server、または Oracle Database (ORM SQLAlchemy 経由) のいずれかである必要があります。

詳細については、[サポートされるデータベース](#)を参照してください。

カスタマイズと実行

デフォルトでは、コンテナは組み込みの SQLite データベースを使用します。PostgreSQL または別のデータベースを使用するには、対応するドライバーをカスタム Docker イメージに追加し、必要な環境変数を使用してイメージを実行します。

詳細については、[メタデータ分類子データベースのドライバーの追加およびメタデータ分類子データベースの接続](#)を参照してください。

EBX® メタデータエージェント

詳細については、[TIBCO® メタデータエージェントのインストール](#)を参照してください。

検証

Docker コンテナが正常に起動した後、EBX® メタデータ管理アプリケーションが正しくインストールされていることを確認するには、以下を使用します。

- ログで次のメッセージを確認: `Module ebx-metadata repository initialization complete.`
- EBX® UI で、メインツールバーのパースペクティブのリストに**メタデータ**が含まれているかを確認します。含まれている場合、インストールは成功しています。

詳細については、[メタデータパースペクティブへのアクセス](#)を参照してください。

制限事項

コンテナ外への EBX® メタデータ分類子のインストールはサポートされていません。

オンプレミスTomcat 10.1.Xのインストールノート

- [概要](#)
- [EBX® メタデータ分類器と分類器データベース](#)
 - [要件](#)
 - [設定と起動](#)
- [EBX® メタデータ管理モジュール](#)
 - [要件](#)
 - [設定](#)
 - [デプロイ](#)
- [EBX® メタデータエージェント](#)
- [最終確認](#)
- [制限事項](#)

概要

この章では、TIBCO EBX® がすでにインストールされているTomcatのオンプレミス環境におけるEBX® メタデータ管理アプリケーションの簡易インストール例について説明します。

EBX® メタデータ管理アプリケーションには、以下が必要です。

- EBX® 6.2.3

- EBX® メタデータ分類器と分類器データベース
- EBX® アドオンWebアプリケーション
- EBX® メタデータエージェント
- EBX® メタデータ管理モジュール

上記のコンポーネントをインストールおよび設定した後、EBX® メタデータ管理アプリケーションが利用可能であることを確認するために、**最終確認**を実行します。

詳細については、[Tomcat 10.1.Xのインストールノート](#)を参照してください。

EBX® メタデータ分類器と分類器データベース

このセクションの情報をを使用して、分類器と分類器データベースをインストール、設定、および実行します。

要件

開始する前に、システムに以下がインストールされていることを確認してください。

- Python 3.11.x + pipenv v2026.5.2 または Docker CLI (「[設定と起動](#)」を参照)
- データベース: SQLite、PostgreSQL、Microsoft SQL Server、またはOracleデータベース (ORM SQLAlchemy経由)

詳細については、[サポートされているデータベース](#)を参照してください。

設定と起動

ターゲット環境でEBX® メタデータ分類器Pythonアプリケーションをインストール、設定、および実行するには、以下のいずれかの方法を使用します。

- [オプション1: スタンドアロンインストール](#)
- [オプション2: Dockerコンテナとして実行](#)

オプション1: スタンドアロンインストール

1. EBX® インストールCDから `ebx.software/ebx-metadata/classifier` ディレクトリを作業ディレクトリにコピーします。

この例では、ディレクトリは `/opt/ebx` にコピーされるため、EBX® メタデータ分類器ディレクトリは `/opt/ebx/classifier` に配置されます。

2. `/opt/ebx/classifier` ディレクトリにある `setEnv.sh` (Linux) または `setEnv.bat` (Windows) ファイルのシステムプロパティを更新して、正しいロギングディレクトリとデータベース接続を使用するようにします。

注意:

`EBX\METADATA\CLASSIFIER\LOG\DIR`で指定されたディレクトリが書き込み可能であることを確認してください。

注意:

SQLiteデータベースを使用する場合、
`EBX\METADATA\CLASSIFIER\SQLALCHEMY_URL`で定義されたディレクトリファイルが書き込み可能であることを確認してください。

以下の例はLinuxに適用されます。

```
#####

## 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
```

EBX® メタデータ分類器がSQLite以外のデータベースを使用するように設定されている場合、対応するドライバーをインストールする必要があります。例えばPostgreSQLの場合、以下のコマンドを実行して `psycopg2-binary` Pythonモジュールをインストールする必要があります。

```
export PIPENV_PIPFILE="/opt/ebx/classifier/ebx_metadata_classifier/
Pipfile"

pipenv install psycopg2-binary
```

注記:

詳細については、[SQLAlchemyデータベースURL](#)を参照してください。

- 必要なPythonモジュールを設定およびインストールするために、`install-classifier.sh` (Linux) または `install-classifier.bat` (Windows) を実行します。

注意:

EBX® が新しいバージョンに更新されるたびに、この手順を再度実行してください。

- 分類器サービスを起動してテストします。

- Linuxの場合: `start-classifier.sh`
- Windowsの場合: `start-classifier.bat`

サービスが起動した後、以下のURLにアクセスできることを確認してください: <https://:api>

- (オプション): 分類器サービスを停止するには、`stop-classifier.sh` (Linux) または `stop-classifier.bat` (Windows) を実行します。

オプション2: Dockerコンテナとして実行

Dockerfileベースのインストールを使用して、ホストマシン上でEBX® メタデータ分類器Dockerコンテナを実行します。

1. EBX® インストールCDから `ebx.software/ebx-metadata/classifier` ディレクトリを作業ディレクトリにコピーします。

この例では、ディレクトリは `/opt/ebx` にコピーされるため、EBX® メタデータ分類器ディレクトリは `/opt/ebx/classifier` に配置されます。

2. EBX® メタデータ分類器Dockerイメージを作成するには、`/opt/ebx/classifier` ディレクトリで以下のコマンドを実行します。

```
docker build -f docker/Dockerfile -t ebx-metadata-classifier:6.2.3 .
```

デフォルトでは、このイメージは以下のDockerボリュームを定義します。

- `/app/ebx/logs` — EBX® メタデータ分類器のログファイル用
- `/app/ebx/data/sqlite` — SQLiteデータベースファイルの保存用

3. EBX® メタデータ分類器Dockerイメージを起動するには、以下のコマンドを実行します。

```
docker run -p 5000:5000 -v /my_custom/data:/app/ebx/data -d ebx-metadata-classifier:6.2.3
```

EBX® メタデータ分類器Dockerコンテナは、以下の環境変数を使用して設定できます。

環境変数名	説明	デフォルト値
EBX_METADATA_CLASSIFIER_LOG_LEVEL	EBX® メタデータ分類器モジュールのロギングレベル。	INFO
EBX_METADATA_CLASSIFIER_SQLALCHEMY_URL	SQLAlchemy形式のデータベース接続URL。	sqlite:///app/ebx/data/sqlite/ebx-metadata-classifier.db

注記:

ホスト上で複数のコンテナを実行できますが、エントリポートは異なる必要があります (例: `-p 5001:5000`)。

EBX® メタデータ管理モジュール

このセクションの以下のトピックでは、EBX® メタデータ管理モジュールを設定およびデプロイする方法について説明します。

- 要件
- 設定
- デプロイ

要件

EBX® メタデータ管理モジュールを設定およびデプロイする前に、Tomcat 10.1.Xのインストールノートおよびサポートされているアプリケーションサーバーの指示に従って、EBX® 6.2.3と以下の追加機能がインストールされていることを確認してください。

- TIBCO EBX® Digital Asset Managerアドオン (DAMA)
- TIBCO EBX® Data Exchangeアドオン (DINT)
- TIBCO EBX® Data Model and Data Visualizationアドオン (DMDV)
- TIBCO EBX® Insightアドオン (DPRA)
- TIBCO EBX® Information Searchアドオン (TESE)

詳細については、[TIBCO EBX® アドオンの管理](#)を参照してください。

設定

1. EBX® インストールCDから `ebx.software/ebx-metadata/config/ebx-metadata.properties` を `EBX_HOME` ディレクトリにコピーします。
2. `EBX_HOME` ディレクトリにある `ebx-metadata.properties` ファイルを次のように更新します。

```
#####
#####

###

### 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. EBX® インストールCDから `ebx.software/ebx-metadata/config/init` フォルダを、`ebx-metadata.properties` で指定された `ebx-metadata.init.directory` にコピーします。

注意:

`ebx-metadata.dama.directory` で指定されたディレクトリが書き込み可能であることを確認してください。

4. 以下の行を `ebx.properties` ファイルに追加して、EBX® メタデータのロギングを設定します。

```
#Metadata Log4J Properties:

ebx.log4j.category.log.wbp.ebx-metadata=          INFO, Console,
ebxFile:ebx-metadata
```

5. `$CATALINA_HOME/bin` または `$CATALINA_BASE/bin` にある `setenv.sh` (Linux) または `setenv.bat` (Windows) ファイルを修正して、環境変数 `ebx-metadata.properties` と `ebx-metadata-classifier.url` を追加します。

- Unixシェル起動の場合:

```
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>"
```

- Windowsコマンドプロンプト起動の場合:


```
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>"
```

デプロイ

TomcatにEBX® メタデータモジュールをデプロイするには:

1. EBX® インストールCDから `ebx.software/ebx-metadata/webapps/wars-packaging/ebx-metadata.war` ファイルを `$CATALINA_HOME/webapps/` (または `$CATALINA_BASE/webapps/`) ディレクトリにコピーします。
2. Tomcat起動後、ブラウザで以下のURLを入力してEBX® Webアプリケーションを実行します:
<http://localhost:8080/ebx/>

EBX® メタデータエージェント

詳細については、[TIBCO® メタデータエージェントのインストール](#)を参照してください。

最終確認

Tomcatが正常に起動した後、EBX® メタデータ管理アプリケーションが正しくインストールされたことを確認するために、以下を使用します。

- ログに以下のメッセージが表示されていることを確認します: `Module ebx-metadata repository initialization complete.`
- EBX® UIで、メインツールバーのパーспекティブのリストに** Metadata**が含まれていることを確認します。含まれている場合、インストールが成功したことを示します。

詳細については、[メタデータパーспекティブへのアクセス](#)を参照してください。

制限事項

EBX® Container Editionを使用する場合、EBX® メタデータ分類器をコンテナ外にインストールすることはサポートされていません。

TIBCO EBX® メイン構成ファイル

- 概要
 - ファイルの場所
 - カスタムプロパティと変数置換
- EBX® ライセンスキープロパティファイルの設定

- 初回起動時の自動インストールの設定
- EBX® ルートディレクトリの設定
- EBX® リポジトリの構成
- ユーザーとロールのディレクトリの構成
 - 同期ディレクトリ
 - ビルトイン LDAP ディレクトリ
 - SCIM ディレクトリ
 - SAML 2.0
 - OpenID Connect
 - ベアラー認証
- EBX® ローカリゼーションの構成
- 一時ファイルディレクトリの設定
- 監査証跡のアクティブ化
- レガシー XML 監査証跡のアクティブ化 (非推奨)
- EBX® ログの構成
 - EBX® メインファイルメカニズム
 - EBX® メインファイルメカニズムから Apache Log4j™ 2 への移行とその逆
- SMTP とメールの有効化と構成
- データサービスの構成
- JMS の有効化と構成
- 分散データ配信 (D3) の構成
- REST Toolkit サービスの構成
- キーストアの機密プロパティファイルの場所
- ステージングの有効化と構成
- エンドユーザーブラウザーからの Web アクセスの構成
 - HTTP 認証ヘッダーポリシー
 - URL コンピューティング
 - プロキシモード
 - リバースプロキシモード

- セキュリティ構成
- フェールオーバーの構成
- EBX® リポジトリの調整
- 高度なインデックス構成
 - 起動時のインデックスロード
- その他
 - データワークフローの有効化
 - ユーザータスクレガシーモードの無効化
 - 階層プランビューの無効化
 - API によるワークフロー起動のための委任の有効化
 - ログプロシージャの開始
 - ログ検証の開始
 - リクエストキャッシュの有効化
 - ログのリクエスト期間のしきい値
 - ログのリクエスト反復のしきい値
 - ログのリクエスト最適化のしきい値
 - リクエストログの SQL 表現
 - 外部キーラベルでテーブルを並べ替えるためのリクエストの最適化
 - 展開サイトの識別
 - メイン構成の動的リロード
 - モジュールの展開解除宣言
 - モジュールのパブリックパスプレフィックス
 - EBX® 実行モード
 - リソースフィルタリング
 - HTML ヘッダー
 - 検証レポートページ
 - 検証レポートログ
 - 子データセットの検証モード
 - APV は無効な入力を DataContext に格納する

概要

EBX® メイン構成ファイル (デフォルト名は `ebx.properties`) には、EBX® を実行するための基本パラメーターのほとんどが含まれています。これは、標準の単純な行指向フォーマットを使用する Java プロパティファイルです。

メイン構成ファイルは、[Jakarta EE 展開記述子](#) を補完します。管理者は、ユーザーインターフェイスを通じてさらに構成を行うこともできます。構成は EBX® リポジトリに保存されます。

以下も参照してください。:

- [展開の詳細](#)
- [UI 管理](#)

ファイルの場所

メイン構成ファイルへのアクセスパスは、いくつかの方法で指定することができます。優先度は降順です。

1. Java システムプロパティ「`ebx.properties`」を定義します。たとえば、このプロパティは、オプション `-Debx.properties=<filePath>` を `java` コマンドラインコマンドに追加することで設定できます。詳細については、[Java ドキュメント](#)を参照してください。
2. サーブレット初期化パラメーター「`ebx.properties`」を定義します。

この標準 Jakarta EE 設定は、Web アプリケーション「`ebx`」の `web.xml` ファイルで指定する必要があります。EBX® は、サーブレット `FrontServlet` のメソッド `ServletConfig.getInitParameter ("ebx.properties")` を呼び出すことによって、このパラメーターにアクセスします。

`ServletConfig` ドキュメントの [getInitParameter](#) を参照してください。

3. デフォルトでは、何も指定されていない場合、メイン構成ファイルは Web アプリケーション「`ebx`」の `WEB-INF/ebx.properties` に格納されます。

注意:

メイン構成ファイルでのプロパティの指定に加え、システムプロパティでプロパティの値を直接設定することもできます。たとえば、`java` コマンドラインコマンドの `-D` 引数を使用します。

カスタムプロパティと変数置換

プロパティの値には、構文 `${propertyKey}` を使用する 1 つ以上の変数を含めることができます。ここで、`propertyKey` は、システムプロパティまたはメイン構成ファイルで定義済みのプロパティのいずれかです。

たとえば、EBX® で提供されるデフォルトの構成ファイルは、カスタムプロパティ `ebx.home` を使用してデフォルトの共通ディレクトリを設定し、それが他のプロパティに含まれます。

EBX® ライセンスキープロパティファイルの設定

以下も参照してください。:

- [初期化と初回起動アシスタント](#)

ライセンスキーは [TIBCO eDelivery](#) サイトから取得できます。

```
#####

## 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
```

ライセンスキープロパティファイルには以下が含まれている必要があります。

```
#####

## 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>
```

EBX® サーバーを再起動せずにライセンスキーを更新できます。サーバーが変更を検出すると、ライセンスステータスを含むカーネルログエントリが書き込まれます。キーが無効な場合、ユーザーがインターフェイスにアクセスするとライセンスウィザードが表示されます。

注意:

期間限定のライセンスキーの場合、有効期限が切れる数日前に、ライセンスが間もなく期限切れになることを知らせる毎日のカーネル WARN ログが通知されます。

初回起動時の自動インストールの設定

リポジトリは、初回起動時に自動的にインストールできます。

```
#####

## 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=ff297ae08f7eeb63230b55f7c45a720a017bc
71d22eaaec...
```

EBX® ルートディレクトリの設定

EBX® ルートディレクトリには、Lucene インデックスディレクトリ、アーカイブ、およびリポジトリが H2 スタンドアロンモードで永続化されている場合は H2 データベースファイルが含まれます。また、レガシー XML 監査証跡 (非推奨) も含まれています。

```
#####

## Path for EBX® XML repository

#####

ebx.repository.directory=${ebx.home}/ebxRepository
```

以下も参照してください。:

- ファイルシステムの監視とクリーンアップ

EBX® リポジトリの構成

EBX® リポジトリの永続性プロパティを構成する前に、リポジトリ管理の章のテクニカルアーキテクチャセクションをよくお読みください。

サポート対象の各データベースに必要なライブラリ (ドライバー) については、データベースドライバーの章で説明しています。

以下も参照してください。:

- リポジトリ管理
- データベースアクセスとユーザー権限のルール
- サポートされるデータベース
- EBX® リポジトリのデータソース
- データベースドライバー

```
#####

## The maximum time to set up the database connection,

## in milliseconds.

#####

ebx.persistance.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=yyyyyyyyy
```

```
#####

## 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=yyyyyyyyy

## 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=yyyyyyyyy

#####

## 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=yyyyyyyyy

#####

## 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=yyyyyyyyy

```

ユーザーとロールのディレクトリの構成

ディレクトリを構成するには、次の専用プロパティを設定および更新します。

以下も参照してください。:

- ユーザーとロールのディレクトリ
- `DirectoryFactory`

```
#####

##

## 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

```

ビルトインロールの「ADMINISTRATOR」を無効にすることもできます。

```

#####

## Specifies whether the built-in role ADMINISTRATOR is disabled.

## Default value is false.

#####

#ebx.directory.disableBuiltInAdministrator=true

```

同期ディレクトリ

同期ディレクトリを構成するには、次の専用プロパティを設定および更新します。

```

#####

## 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=

```

以下も参照してください。:

- 同期ディレクトリ

ビルトイン LDAP ディレクトリ

メインの EBX® 構成ファイルを通じて、ビルトインの LDAP (Lightweight Directory Access Protocol) ディレクトリ実装をアクティブ化して構成します。

LDAP を有効にするには、*ebx.directory.factory* プロパティを次のように更新します。

```
ebx.directory.factory=com.orchestranetworks.service.directory.ldap.LdapDirectoryFactory
```

LDAP を構成するには、次の専用プロパティを設定および更新します。

```
#####

## 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.expirationInSeco
nds=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=gro
```



```
upOfNames)

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
```

以下も参照してください。:

- LDAP ディレクトリ

SCIM ディレクトリ

メインの EBX® 構成ファイルを通じて、SCIM (System for Cross-domain Identity Management) 機能をアクティブ化して構成します。

SCIM を構成するには、次の専用プロパティを設定および更新します。

```
#####

## 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

```

以下も参照してください。:

- SCIM ディレクトリ

SAML 2.0

SAML 2.0 は、メイン構成ファイルを通じてアクティブ化および構成できます。

```
#####

## 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

```

以下も参照してください。:

- [SAML 2.0](#)

OpenID Connect

OpenID Connect は、メイン構成ファイルを通じてアクティブ化および構成できます。

```

#####

## 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
```

以下も参照してください。:

- [OpenID Connect](#)

ベアラー認証

ベアラー認証は、メイン構成ファイルを通じてアクティブ化および構成できます。

注意:

JWT の場合、「sub」(サブジェクト) クレームに EBX® で使用するユーザー名が含まれていることを確認してください。

```
#####

## 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=rNvm8412z0o9ZZFv0irUZe0JYIyRz8cK]
```

EBX® ローカリゼーションの構成

このパラメーターは、実行時に使用されるロケールの構成に使用されます。このリストには、エンドユーザーに公開されているすべてのロケールが含まれています。EBX® は、このリストで宣言されていない言語でラベルおよびメッセージを表示することはできません。

デフォルトのロケールは、リストの最初のロケールになります。

```
#####

## 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
```

以下も参照してください。:

- TIBCO EBX® ローカリゼーションの拡張

一時ファイルディレクトリの設定

一時ファイルは次のように保存されます。

```
#####

## 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
```

監査証跡のアクティブ化

デフォルトでは、監査証跡のログは無効になっています。INFO レベルのメッセージをログに記録するように「監査」ログカテゴリを構成することでアクティブ化できます。たとえば、以下は特定の「[ebxFile](#)」アペンダーを使用して監査証跡メッセージをログに記録します。

```
ebx.log4j.category.log.audit = INFO, ebxFile:audit
```

以下も参照してください。:

- [EBX® ログの構成](#)
- [監査証跡](#)

レガシー XML 監査証跡のアクティブ化 (非推奨)

デフォルトでは、XML 監査証跡は有効化されていません。次の変数を使用して有効化できます。

```
#####

## 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
```

以下も参照してください。:

- [レガシー XML 監査証跡](#)

EBX® ログの構成

以下も参照してください。:

- [ログの構成](#)

EBX® メインファイルメカニズム

最も重要なロギングカテゴリは次のとおりです。

ebx.log4j.category.log.kernel	EBX® の主な機能、モジュールとデータモデルの処理、例外、およびコンパイル結果のログ。
ebx.log4j.category.log.workflow	ワークフローに関する主な機能、警告、および例外のログ。
ebx.log4j.category.log.persistence	基礎となるデータベースとの通信に関連するログ。
ebx.log4j.category.log.setup	モジュールとデータモデルを除くすべての EBX® オブジェクトのコンパイル結果のログ。
ebx.log4j.category.log.validation	データセットの検証結果のログ。
ebx.log4j.category.log.mail	サーバーから送信されたメールに関連するアクティビティのログ (SMTP とメールの有効化と構成を参照)。 注意：このカテゴリでは、無限ループを回避するためにカスタム SMTP アペンダーを使用しないでください。
ebx.log4j.category.log.d3	EBX® の D3 イベントのログ。
ebx.log4j.category.log.dataservices	EBX® のデータサービスイベントのログ。
ebx.log4j.category.log.monitoring	メモリ監視の未加工ログ。
ebx.log4j.category.log.request	EBX® の リクエスト および クエリ イベントのログ：期間が <code>ebx.logs.request.durationThreshold</code> ミリ秒を超える実行、最適化フェーズが <code>ebx.logs.request.optimizationThreshold</code> ミリ秒を超えるリクエスト/クエリ、および結果の反復中に内部的に例外が発生する実行。ログレベルが <code>DEBUG</code> に設定されてい

	る場合、すべてのリクエスト/クエリは期間に関係なくログに記録されます。
ebx.log4j.category.log.restServices	REST Toolkit からのイベントを含む、EBX® の REST サービスイベントのログ。
ebx.log4j.category.log.audit	監査証跡機能のログ。
ebx.log4j.category.log.assistant	アシスタント機能のログ。

これらのカテゴリの一部は、`LoggingCategory` インターフェイスを使用してカスタムコードで書き込むこともできます。

```
#####

## 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.PatternLay
out

#-----

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

```

カスタム ebxFile アペンダー

トークン `ebxFile:` は、デフォルト設定で毎日のローリングファイルアペンダーを定義するためのショートカットとして使用できます。この後にファイル名を付ける必要があります。次に、ディレクトリ `ebx.logs.directory` 内のファイルに書き込むアペンダーを有効にします。この場合のしきい値は `DEBUG` に設定します。

プロパティ `ebx.log4j.appender.ebxFile.backup.Threshold` では、毎日のロールオーバーのバックアップファイルの最大サイズ (メガバイト単位) を定義できます。

```
#####

## 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
```

カスタム SMTP アペンダー

アペンダー `com.onwbp.org.apache.log4j.net.SMTPAppender` は、非同期のメール送信者を指定します。

以下も参照してください。:

- SMTP とメールの有効化と構成

カスタムモジュールログのしきい値

デフォルトでは、カスタムモジュールに関連付けられているログカテゴリのログレベルのしきい値は、`INFO` に設定されています。

このしきい値は、カスタムモジュール `xxxxxx` のプロパティ `ebx.log4j.category.log.wbp.xxxxxx` を設定することでカスタマイズできます。

例： `ebx.log4j.category.log.wbp.mycompany-module=DEBUG`

以下も参照してください。：

- `ModuleContextOnRepositoryStartup.getLoggingCategory`

アドオンモジュールのログしきい値

デフォルトでは、アドオンモジュールのログレベルのしきい値は、`INFO` に設定されています。

ログレベルのしきい値は、アドオンモジュール `ebx-addon-xxxxxx` のプロパティ `ebx.log4j.category.log.addon.xxxxxx` を設定することでカスタマイズできます。

例： `ebx.log4j.category.log.addon.daqa=DEBUG`

EBX® メインファイルメカニズムから Apache Log4j™ 2 への移行とその逆

Apache Log4j™ 2 がプログラムで構成されていないと仮定すると、あるアプローチから別のアプローチに移行することは、以前に使用していたファイルをクリーンアップし、すべてのロガー構成を新しい形式に変換し、ターゲットファイルを適切な場所にインストールするのと同じくらい簡単です。

EBX® ロガー構成を定義するには、[EBX® ログの構成](#)を参照してください。

Apache Log4j™ 2 ロガー構成を定義するには、EBX® ロガー名から `ebx.log4j.category` プレフィックスを削除し、`ebx.log4j.rootCategory` 構成をルートロガーに置き換えます。残りの手順については、[Apache Log4j™ 2 構成ドキュメント](#)を参照してください。

基礎となる EBX® ログ構成を考慮すると、次のようになります。

```
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
```

Apache Log4j™ 2 と同等の構成は次のようになります。

```
{
  "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"
    }
  }
}

```

注意:

カスタム `ebxFile` アペンダーは Apache Log4j™ 2 にネイティブには存在しませんが、その動作は模倣することができます。

注意:

あるメカニズムから別のメカニズムに移行する場合は、機能範囲をできるだけ近づけるために、すべてのロガーを移行してください。

SMTP とメールの有効化と構成

内部メールマネージャーは、非同期でメールを送信します。これは、ワークフローエンジンとカスタム SMTP アペンダー `com.onwbp.org.apache.log4j.net.SMTPAppender` で使用されます。

以下も参照してください。:

- メールセッション

```
#####

## 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
```

データサービスの構成

```
#####

## 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 B0 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
```

JMS の有効化と構成

以下も参照してください。:

- データサービス用の JMS

```
#####

## 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

```

分散データ配信 (D3) の構成

D3 に関連するメイン構成ファイルのプロパティについては、[D3 ノードの構成](#)を参照してください。

以下も参照してください。:

- [分散データ配信用の JMS \(D3\)](#)
- [D3 の概要](#)

REST Toolkit サービスの構成

```

#####

## 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
```

キーストアの機密プロパティファイルの場所

機密プロパティは、キーストアファイルの場所と、シークレットキーのエイリアスやパスワードなどのさまざまな情報を指定します。

```
#####

## 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
```

以下も参照してください。:

- キーストアファイル

ステージングの有効化と構成

ステージングは構成によって有効化/無効化されます。メモリとディスクの使用を最適化するために、いくつかのオプションを設定できます。プロパティは次のように構成されます。

```
#####

## 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

```

以下も参照してください。:

- ステージング

エンドユーザーブラウザからの Web アクセスの構成

HTTP 認証ヘッダーポリシー

EBX® は、HTTP ヘッダーを使用して資格情報を送受信するための 3 つのポリシーをネイティブに提供します。

<code>standard</code>	これは、 RFC 2617 で説明されている HTTP 認証 ヘッダーを使用した認証スキームに対応します。
<code>ebx</code>	HTTP 認証ヘッダーのオーバーライド防止のため、このポリシーは <code>standard</code> と同様に機能しますが、資格情報は EBX® 固有の HTTP ヘッダーに保存されます。
<code>both</code>	これは、前述の 2 つのポリシーの組み合わせです。

```
#####

## 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
```

URL コンピューティング

デフォルトでは、EBX® は「スタンドアロン」モードで実行され、外部リソース (例、画像、JavaScript) はアプリケーションサーバーによって提供されます。

また、デフォルトでは、メイン構成ファイルの URL 関連のパラメーターを設定する必要はありません。

この場合、サーバー名とポートは、EBX® に送信された初期リクエストから取得されます。

以下も参照してください。:

- URL ポリシー (非推奨)

```
#####

## 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=

```

プロキシモード

プロキシモードでは、フロントエンド HTTP サーバーを使用して静的リソース (例、画像、CSS、JavaScript) を提供できます。このアーキテクチャは、静的 HTTP リクエストでのアプリケーションサーバーの負荷を軽減します。この構成では、フロントエンドサーバーで SSL セキュリティを使用することもできます。

Web サーバーは、URL 内のパスに従ってアプリケーションサーバーにリクエストを送信します。

`servletAlias` および `uiServletAlias` パスは、メイン構成ファイルで指定されます。

Web サーバーは、すべての外部リソースを提供します。これらのリソースは、

`resourcesAlias` パスを使用してアクセス可能な専用ディレクトリに保存されます。

EBX® は、ファイルシステムから外部リソースにもアクセスする必要があります。これを行うには、プロパティ `ebx.webapps.directory.externalResources` を指定する必要があります。

`ebx.servlet.http.useHttpsSettings` および/または

`ebx.externalResources.http.useHttpsSettings` プロパティが `true` に設定されている場合、受信リクエストは常に HTTPS であると見なされます。デフォルト値は `false` です。これらのプロパティは、EBX® がリバースプロキシまたは HTTPS 暗号化を処理するファイアウォールの背後にある場合に役立ちます。

メイン構成ファイルは、次のように構成できます。

```
#####

## 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
```

注意:

プロキシモードを使用する場合、`ebx-dataservices` モジュールへの URL はリネージ管理パネルから構成する必要があります。指定された URL は、パスを `/ebx-dataservices` で終了する必要があることに注意してください。

リバースプロキシモード

EBX® で生成されたリクエストおよび外部リソースの URL に、受信リクエストとは異なるプロトコル、特定のサーバー名、特定のポート番号、または特定のパスプレフィックスが必要な場合、プロパティは次のように構成できます。

```
#####
```

```
#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=
```

注意:

リバースプロキシモードを使用する場合、`ebx-dataservices` モジュールへの URL はリネージ管理パネルから構成する必要があります。指定された URL は、パスを `/ebx-dataservices` で終了する必要があることに注意してください。

セキュリティ構成

これらのパラメーターは、EBX® がユーザーをリダイレクトできる外部ページと、認証 Cookie に関するいくつかのオプションを構成するために使用されます。

```
# 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
```

これらのパラメーターは、CSP 除外に追加するルールを追加するために使用されます。

```
# 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=
```

フェールオーバーの構成

これらのパラメーターは、フェールオーバーモードとアクティベーションキー、および **DEBUG** モードでのハートビートロギングを構成するために使用されます。

以下も参照してください。:

- ホットスタンバイによるフェールオーバー

```
#####

## 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

```

EBX® リポジトリの調整

一部のオプションは、メモリ使用量を最適化するように設定できます。

プロパティは次のように構成されます。

```

#####

## 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

```

以下も参照してください。:

- [検証レポートページ](#)

高度なインデックス構成

起動時のインデックスロード

インデックスは、EBX® の起動時にロードして、初期テーブルアクセスを高速化できます。インデックスは、欠落している場合 (`open` 再構築戦略)、またはインデックスが最新のモデルと一致しない場合 (`update` 再構築戦略) に再構築されます。

プロパティは次のように構成されます。

```
# 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=
```

注意:

rebuild 戦略は、細心の注意を払って使用する必要があります。これは、起動時に選択された各インデックスの完全な再構築を強制し、すべての再構築が完了するまで待機します。このモードは、カスタマーサポートから指示された場合にのみ使用し、再起動後に **open** または **update** に戻す必要があります。

その他

データワークフローの有効化

このパラメーターは、データワークフローを有効にするかどうかを指定します。このパラメーターは、即時受容されません。値の変更のたびにサーバーを再起動する必要があります。

```
#####

## Workflow activation.
```



```
## Default is true.

#####

ebx.workflow.activation = true
```

ユーザータスクレガシーモードの無効化

このパラメーターは、レガシーモードでのユーザータスクの作成サービスをワークフローモデリングで提供するかどうかを指定します。デフォルト値は `false` です。

詳細については、`UserTask.UserTaskMode.LEGACY_MODE` を参照してください。

```
#####

## 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
```

階層プランビューの無効化

このパラメーターは、階層プランビューを非表示にするかどうかを指定します。デフォルト値は `true` です。

```
#####

## Activate or deactivate Workflow hierarchy plan view

#####

ebx.manager.workflow.hierarchyPlanView.hidden=false
```

API によるワークフロー起動のための委任の有効化

このパラメーターは、API によるワークフロー起動の委任を有効にします。デフォルト値は `false` です。これは、このパラメーターが有効化されていない限り、権限が常にチェックされることを意味します。

```
#####

## Activate delegation to launch workflows by API

#####

ebx.workflow.api.processLauncher.permissionCheckDelegated=true
```

ログプロシージャの開始

このパラメーターは、プロシージャ実行の開始をログに記録するかどうかを指定します。

```
#####

## Specifies whether transaction starts are logged. Default is false.

#####

ebx.logs.logTransactionStart = true
```

ログ検証の開始

このパラメーターは、データセット検証の開始をログに記録するかどうかを指定します。

```
#####

## Specifies whether validation starts are logged. Default is false.

#####

ebx.logs.logValidationStart = true
```

リクエストキャッシュの有効化

このパラメーターは、`リクエスト` の実行時にクエリエンジンで必要とされる内部のリクエストからクエリへの変換のためにキャッシュをアクティブにするかどうかを指定します。このキャッシュを有効にすると、特に複数回実行される場合に、特定の種類のリクエストのパフォーマンスが向上します。

```
# 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
```

ログのリクエスト期間のしきい値

このパラメーターは、ログに記録される `リクエスト` および `クエリ` の期間のしきい値をミリ秒単位で指定します。ログは、ログカテゴリ `ebx.log4j.category.log.request` のレベルが `INFO` 以下の場合に生成されます。レベルが `DEBUG` の場合、すべての `リクエスト` と `クエリ` がログに記録されます。

```
#####

## 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
```

ログのリクエスト反復のしきい値

このパラメーターは、**リクエスト**と**クエリ**の2つのログ間の期間しきい値を超える遅延をミリ秒単位で指定します。この値が0より大きく、クエリ期間が期間しきい値を超えた場合、各ログの間でこの遅延以上になると、繰り返しログに記録されます。ログメッセージには期間が含まれているため、長いクエリ期間を追跡するのに役立ちます。

```
#####

## 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
```

ログのリクエスト最適化のしきい値

このパラメーターは、ログに記録される**リクエスト**および**クエリ**の最適化フェーズのしきい値をミリ秒単位で指定します。ログは、ログカテゴリ **ebx.log4j.category.log.request** のレベルが **INFO** 以下の場合に生成されます。

```
#####

## Specifies in milliseconds the threshold of optimization of Requests and
Queries

## to be logged. Default value is 1000 ms.
```

```
#####
```

```
#ebx.logs.request.optimizationThreshold=1000
```

リクエストログの SQL 表現

このパラメーターは、特定の **リクエスト** のログに、対応するリクエストの SQL 表現を含めるかどうかを指定します。この変換はパフォーマンスの面で負担が大きく、すべてのリクエストを SQL に変換できるわけではないことに注意してください。デフォルト値は **false** です。

```
#####
```

```
## Specifies if the logs of requests shall include the SQL representation.
```

```
## Default is false.
```

```
#####
```

```
#ebx.log4j.category.log.request.enableSQLConversion=false
```

外部キーラベルでテーブルを並べ替えるためのリクエストの最適化

このパラメーターは、クエリオプティマイザーが、並べ替え操作と結合操作の特定の組み合わせを、並べ替えられた方法で結合を実行する特殊な演算子に書き換えるかどうかを指定します。

```
#####
```

```
## 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
```

展開サイトの識別

このパラメーターを使用すると、テクニカルログメールの送信先メールアドレスを指定できます。

```
#####

## Unique Site Name

## --> used by monitoring emails and by the repository

#####

ebx.site.name= name@domain.com
```

メイン構成の動的リロード

一部のパラメーターは、EBX® を再起動せずに動的にリロードできます。パラメーター `thisfile.checks.intervalInSeconds` は、メイン構成ファイルの確認頻度を示します。

```
#####

### Checks if this file has been updated

### If value <= 0, no more checks will be done

#####

thisfile.checks.intervalInSeconds=1
```

開発モードでは、このパラメーターを 1 秒まで設定できます。変更の頻度が少ないと予想される実稼働システムでは、値を大きくするか、0 に設定してホットリロードを完全に無効にすることができます。

このプロパティは、モジュールが WAR として展開されている場合は、アプリケーションサーバーに依存するため、常にサポートされるとは限りません。

モジュールの展開解除宣言

アプリケーションサーバーの起動時に、展開済み Web アプリケーション / EBX® モジュールの初期化と EBX® リポジトリの初期化は非同期で実行されます。EBX® リポジトリを適切に初期化するには、少なくとも 1 つのデータセットで使用するすべてのデータモデルをコンパイルする必要があります。したがって、EBX® は参照されるモジュールが登録されるまで際限なく待機することになります。

モジュールがデータモデルによって参照されているが、未展開 (または展開解除) の場合は、このモジュールを展開解除宣言し、待機ロックを解除して起動プロセスを続行する必要があります。

注意:

`kernel` ロギングカテゴリは、待機中のモジュールを示します。

注意:

展開解除宣言されたモジュールは、プロパティ `ebx.module.undeployedModules` から削除されるまで、EBX® に登録できません。

注意:

未登録のモジュールに基づくデータモデルでは、「展開解除モジュール」のコンパイルエラーが発生します。

以下も参照してください。:

- [モジュール登録](#)
- [メイン構成の動的リロード](#)

```
#####
```

```
## 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=

```

モジュールのパブリックパスプレフィックス

EBX® モジュールのパブリックパスは、各モジュールの `module.xml` ファイルで宣言されます。次のプロパティを指定することで、`module.xml` のコンテンツを変更することなく、すべてのモジュールに対してコンテキストプレフィックスを宣言できます。

このプレフィックスは、コア、アドオン、特定モジュールを含むすべての EBX® モジュールに適用されます。

プロキシモードおよび/またはリバースプロキシモードが使用されている場合、

`ebx.servlet.http[s].path` および `ebx.servlet.http[s].ui.path` プロパティは、このモジュールのパブリックパスプレフィックスの設定を考慮に入れる必要があります。逆に、`ebx.externalResources.http[s].path` プロパティは、潜在的なプレフィックスの直前でパスを終了する必要があります。

```

#####

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/
```

詳細については、[URL コンピューティング](#)を参照してください。

EBX® 実行モード

このプロパティは、EBX® の実行方法を定義します。*開発*、*統合*、*実稼働*の 3 つの実行モードを使用できます。

*開発*モードで実行すると、[開発ツール](#)が EBX® でアクティブ化されるため、一部の機能に完全にアクセスできるようになり、より多くの技術情報が表示されます。

注意:

管理者は、使用するモードに関係なく、いつでもこの情報にアクセスできます。

開発モードでの実行時にアクセス可能な追加機能には、次のものがあります (非包括リスト)。

ドキュメントペイン	計算値の場合、Java クラス名が表示されます。ノードのパスへのアクセスを許可するボタンが表示されます。
コンパイル情報	モジュールとスキーマのコンパイル情報は、データセット検証レポートに表示されます。
Java バインディング	データセットのスキーマに 1 つ以上のバインディングが記述されている場合、Java バインディングの生成を利用できます。
Web コンポーネントリンクジェネレーター	Web コンポーネントリンクジェネレーターは、データセットとデータスペースで使用できます。
データモデルアシスタント	データモデルの構成と、サービス、ビジネスオブジェクトとルール、Java バインディング、ツールバー、および一部の高度なプロパティなどの追加オプション。
ワークフローモデリング	特定のスクリプトタスクを宣言します。
ログ	ログには、開発者向けの追加の技術情報が含まれています。たとえば、UI Bean の非列挙ノードでドロップダウンリストが定義されている場合、警告がログに書き込まれます。
製品ドキュメント	製品ドキュメントは、管理と開発の章を含む常に完全 (詳細) なドキュメントです。

```
#####
```

```
## Server Mode
```

```
## Value must be one of: development, integration, production
```

```
## Default is production.
```

```
#####
```

```
backend.mode=integration
```

注意:

統合モードと実稼働モードに違いはありません。

リソースフィルタリング

このプロパティを使用すると、リソースディレクトリのコンテンツ (リソースタイプノード、および使用可能なリソースの格納先ディレクトリを示す関連ファセット) 内の特定のファイルとディレクトリをフィルタリングできます。

```
#####
```

```
## 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 ヘッダー

このプロパティを使用すると、HTML ページに追加の JavaScript ファイルを宣言できます。

```
#####
```

```
## 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=
```

検証レポートページ

検証レポートページには、重大度ごとに有限数の項目を表示できます。この数値は、このプロパティで調整できます。

```
#####

## Defines the maximum item displayed for each severity in the validation
report page.

## Default value is 100.

#####

ebx.validation.report.maxItemDisplayed=200
```

以下も参照してください。:

- [EBX® リポジトリの調整](#)

検証レポートログ

このプロパティを使用すると、データセットまたはテーブルの検証時にログに表示する検証メッセージの数を指定できます。

```
#####

## 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
```

デフォルトでは、データセットまたはテーブルの検証の内容がログに記録されます。検証レポートのコンテンツのログ記録は、次のプロパティを使用して無効にすることができます。

```
#####

## Specifies whether the content of validation reports is logged when
validating

## a dataset or a table.

## Default is true.

#####

ebx.validation.report.logContent=true
```

以下も参照してください。:

- [EBX® リポジトリの調整](#)

子データセットの検証モード

このプロパティは、子データセットを検証するアルゴリズムの動作に影響します。パフォーマンスにのみ影響します。結果として得られる検証レポートの内容は、プロパティから独立しています。

```
# 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 は無効な入力を DataContext に格納する

ユーザーが検証エラーのある値を入力した場合の APV の動作を指定します。true の場合、値は DataContext に格納されます。false の場合、値は DataContext に格納されません。以前の値が保持されます。デフォルト値は false です。

```
# Specifies the behavior of APV when the user enters a value that has
validation errors.

# If true, the value will be stored in the DataContext.

# If false, the value will not be stored in the DataContext. The previous
value will be kept.

# The default value is false.
```



```
#ebx.manager.apv.storeInvalidInputInDataContext=false
```

初期化と初回起動アシスタント

成果物は **TIBCO eDelivery** で見つかります (eDelivery にアクセスするにはアカウントが必須です。アカウントをリクエストするには <https://support.tibco.com> にお問い合わせください)。

TIBCO EBX® 構成アシスタントは、EBX® リポジトリの初期構成に役立ちます。EBX® の起動時にリポジトリがインストールされておらず、**自動インストール** が有効になっていない場合、構成アシスタントが自動的に起動されます。

リポジトリの構成を開始する前に、EBX® がアプリケーションサーバーに正しく展開されていることを確認してください。**Jakarta EE の展開** を参照してください。

注意:

EBX® メイン構成ファイルも適切に構成されている必要があります。**TIBCO EBX® メイン構成ファイル**を参照してください。

- ライセンスキー
- 構成手順
 - ライセンス契約の検証
 - リポジトリの構成
 - デフォルトディレクトリでのユーザーの定義
 - 入力した情報の検証
 - EBX® リポジトリのインストール

ライセンスキー

EBX® の起動時に有効なライセンスキーがない場合、つまり、EBX® メイン構成ファイルにライセンスキーが入力されていない場合、または現在のライセンスキーの有効期限が切れている場合、ライセンスキーページが自動的に表示されます。

ライセンスキーは **TIBCO eDelivery** サイトから取得できます (eDelivery にアクセスするにはアカウントが必須です。アカウントをリクエストするには <https://support.tibco.com> にお問い合わせください)。

構成手順

EBX® 構成アシスタントは、次の手順をガイドします。

1. ライセンス契約を検証します
2. リポジトリを構成します
3. デフォルトのユーザーおよびロールディレクトリでユーザーを定義します (カスタム ディレクトリが定義されていない場合)
4. 入力した情報を検証します
5. EBX® リポジトリをインストールします

ライセンス契約の検証

構成を続行するには、製品ライセンス契約を読んで同意する必要があります。

リポジトリの構成

このページには、EBX® メイン構成ファイルで定義されているプロパティの一部が表示されます。また、この手順ではリポジトリのいくつかの基本的なプロパティも定義します。

リポジトリの ID (<code>repositoryId</code>)	リポジトリを一意に識別する必要があります (エンタープライズの範囲内)。識別子の長さは 48 ビット (6 バイト) で、通常は 12 桁の 16 進数で表されます。この情報は、リポジトリで作成されたエンティティのユニバーサル一意識別子 (UUID) と、履歴に記録されたトランザクションの生成に使用されます。この識別子は、RFC 4122 で指定されている「UUID ノード」として機能します。
リポジトリラベル	リポジトリの目的とコンテキストを示すユーザーフレンドリーなラベルを定義します。

以下も参照してください。:

- TIBCO EBX® メイン構成ファイル

デフォルトディレクトリでのユーザーの定義

カスタムユーザーおよびロールディレクトリが EBX® メイン構成ファイルで定義されていない場合、構成アシスタントはデフォルトユーザーおよびロールディレクトリのデフォルトユーザーを定義することを許可します。

管理者ユーザーを定義する必要があります。オプションで、2 番目のユーザーを作成できます。

以下も参照してください。:

- ユーザーとロールのディレクトリ

入力した情報の検証

リポジトリのインストールを続行する前に、リポジトリの構成と「構成の概要」ページに入力された情報を確認できます。情報を変更する必要がある場合は、構成アシスタントの**** ボタンを使用して前のページに戻ることができます。

構成を確認したら、 ボタンをクリックしてインストールを続行します。

EBX® リポジトリのインストール

リポジトリのインストールは、提供された情報を使用して実行されます。インストールが完了すると、リポジトリのログインページにリダイレクトされます。

TIBCO EBX® アドオンの管理

- 概要
- アドオンの互換性
- アドオンモジュールの展開
- アドオンモジュールの登録
- アドオンモジュールのアクティブ化
- アドオンモジュールの削除

概要

次のセクションでは次の内容について説明します。

- アドオンの互換性

- アドオンモジュールの展開
- アドオンモジュールの登録
- アドオンモジュールのアクティブ化
- アドオンモジュールの削除

アドオンの構成および使用方法については、各アドオンのドキュメントを参照してください。

アドオンの互換性

各アドオンバンドルは、EBX® の特定のバージョンおよび修正リリースと互換性があります。ただし、バージョン 6.1.0 以降、アドオンと EBX® のメジャーバージョンとマイナーバージョンは同じにする必要があります。たとえば、EBX® バージョン 6.1.0 は、同じバージョン 6.1.0 のアドオンとのみ互換性があります。そうでない場合、アドオンの登録は中止されます。他のリリースについては、*『TIBCO EBX® Add-ons Versioning and Packaging Guide』*にバージョン互換性マトリックスが含まれています。[公式ドキュメントサイト](#)に移動し、[バンドルバージョンメニュー](#)を使用して、アドオンバンドルに対応するこのガイドのバージョンを表示します。

アドオンモジュールの展開

アドオンの共通 JAR ファイル (`lib/ebx-addons.jar`) を EBX® クラスパスにコピーします。ebx.jar クラスローダーからアクセスできる必要があります。

アドオン共通 WAR ファイル (`wars/ebx-addon-common.war` という名前) をアプリケーションサーバーの webapps フォルダーまたは EAR に EBX® ビルトイン Web アプリケーションとしてコピーします。

アドオンを使用する場合は、EBX® アドオン WAR ファイル (`wars/ebx-addon-<name>.war` という名前) を EBX® ビルトイン Web アプリケーションと同じフォルダーまたは EAR にコピーします。

詳細については、[Web アプリケーション](#)、[展開の詳細](#)、および[インストールに関する注意事項](#)を参照してください。

注意:

アドオンログレベルは、[メイン構成ファイル](#)で管理できます。

アドオンモジュールの登録

アドオンを登録すると、その構成が管理セクションで利用できるようになります。アドオン機能は、アドオンがアクティブ化されている場合にのみエンドユーザーが利用できます。

EBX® アドオンはリポジトリに自動的に登録され、有効化されます。アドオンがアクティブであるかどうかを設定するには、次の手順を実行します。

1. 領域に移動します。
2. ナビゲーションペインの下矢印をクリックして、> を選択します。
3. ページで、設定するアドオンレコードをクリックし、必要な変更を加えます。
4. 登録するアドオンを選択します。
5. をクリックします。

注意:

アドオンの登録を解除しても、既存の構成は削除されませんが、アドオンが再度登録されるまでUIで使えるようになります。

アドオンモジュールのアクティブ化

アドオンをアクティブ化すると、その機能がエンドユーザーに利用可能になります。登録されたアドオンのみをアクティブ化できます。

リポジトリで EBX® アドオンをアクティブ化するには、次の手順を実行します。

1. 領域に移動します。
2. ナビゲーションペインの下矢印をクリックして、> を選択します。
3. アクティベートする登録済みアドオンを選択し、フィールドを有効にします。
4. をクリックします。

アドオンモジュールの削除

EBX® リポジトリからアドオンモジュールを削除するには、次の手順を実行します。

1. パネルに移動します。
2. ナビゲーションペインの下矢印をクリックして、> を選択します。
3. ページで、削除するアドオンに対応するボックスにチェックを入れます。

4. メニューで、を選択します。
5. アプリケーションサーバーを停止します。
6. 削除したアドオンのモジュール名 (ebx-addon-<name>) を `ebx.properties` ファイルの `ebx.module.undeployedModules` プロパティに追加します。詳細については、[モジュールの展開解除宣言](#)を参照してください。
7. アプリケーションサーバーの `webapps` フォルダーからアドオンの WAR ファイル (ebx-addon-<name>.war) を削除するか、EBX® ビルトイン Web アプリケーションとして使用されている場合は EAR ファイルを削除します。
8. アプリケーションサーバーを再起動します。
9. 削除されたアドオンに関連する管理データセットとデータスペースを閉じてから削除します。

アドオンの展開が解除されると、管理データセットに対応するデータスペースがデータスペースの下参照子のリストに表示されます。このため、アドオンに関連するすべてのデータ/データスペースを手動で閉じ、削除する必要があります。

ユーザー認証

- [/ebx-authentication サブレット](#)
- [認可トークン](#)
 - [Cookie のプロパティ](#)
- [認証プロセスのカスタマイズ](#)
 - [ディレクトリのカスタマイズ](#)
 - [ログインページのカスタマイズ](#)
- [シングルサインオン \(SSO\)](#)
 - [SAML 2.0](#)
 - [OpenID Connect](#)

/ebx-authentication サブレット

/ebx-authentication サブレットは、EBX®でのユーザー認証を処理する中心的なポイントです。その役割は次のとおりです。

- ユーザーが初めてログインするときに、[認可トークン](#)を作成します。
- ユーザーがログアウトするときに、[認可トークン](#)を取り消します。

- 許可されたユーザーを適切なページにリダイレクトします。

さらに、構成に応じて、次のこともできます。

- ログイン画面を表示してログインプロセスを処理します。
- 権限のないユーザーをカスタムログイン画面にリダイレクトします。
- ログアウトしているユーザーをカスタム終了ページにリダイレクトします。

認可トークン

/ebx-authentication サブレットはユーザーの認証に成功すると、認証情報を含むトークンを作成し、それをセッション Cookie に保存します。

これは、認証がブラウザーのすべてのタブ間で共有されることを意味します。

- ユーザーがブラウザーのタブですでに認証されており、別のタブで EBX® を開いた場合、再度認証を求められることはありません。
- 同じブラウザーの異なるタブに異なるユーザーでログインすることはできません。

Cookie のプロパティ

トークンを含む Cookie には次のプロパティがあります。

HttpOnly	カスタマイズできません。ブラウザーの JavaScript コードでは Cookie を読み取ることができません。
Secure	<code>ebx.properties</code> のプロパティ <code>ebx.security.authorizationCookie.attribute.secure</code> を介してカスタマイズできます。Cookie を HTTPS 経由のみで送信できるか、HTTP 経由でも送信できるかを定義します。
SameSite	<code>ebx.properties</code> のプロパティ <code>ebx.security.authorizationCookie.attribute.sameSite</code> を介してカスタマイズできます。外部サイトからの閲覧時に Cookie を送信できるかどうかを定義します。

以下も参照してください。:

- セキュリティ構成

認証プロセスのカスタマイズ

デフォルトでは、EBX® はビルトインのディレクトリとログインページを使用します。ビルトインのディレクトリをカスタムディレクトリに置き換えることを強くお勧めします。

ディレクトリのカスタマイズ

EBX® にはビルトインのディレクトリが用意されていますが、これを置き換えることを強くお勧めします。

1. `Directory` をオーバーライドするクラスを作成します。
2. `DirectoryFactory` をオーバーライドするクラスを作成します。
3. `ebx.properties` のプロパティ `ebx.directory.factory` を使用して、ファクトリーを宣言します。

以下も参照してください。:

- セキュリティ構成

ログインページのカスタマイズ

EBX® にはビルトインのログインページが用意されていますが、これを置き換えることも可能です。これを行うには、`ebx.properties` のプロパティ `ebx.security.loginPage.url` を使用してカスタムログインページを宣言します。

ユーザーの認証は、カスタムログインページに依存します。ユーザーが認証されると、`/ebx-authentication/login` ページにリダイレクトされます。次に、EBX® は認可トークンを作成する前に `Directory.authenticateUserFromHttpRequest` メソッドを呼び出します。これが機能するには次の点が必要です。

- `/ebx-authentication/login` を指すリクエストには、Cookie、HTTP ヘッダー、クエリパラメーターなど、ユーザーを認証するのに十分な情報が含まれている必要があります。

- ディレクトリはオーバーライドする必要があります (ディレクトリのカスタマイズを参照)、`Directory.authenticateUserFromHttpRequest` メソッドの実装は、受信リクエストから情報を読み取る必要があります。適切な `UserReference` を返すようにします。

注意:

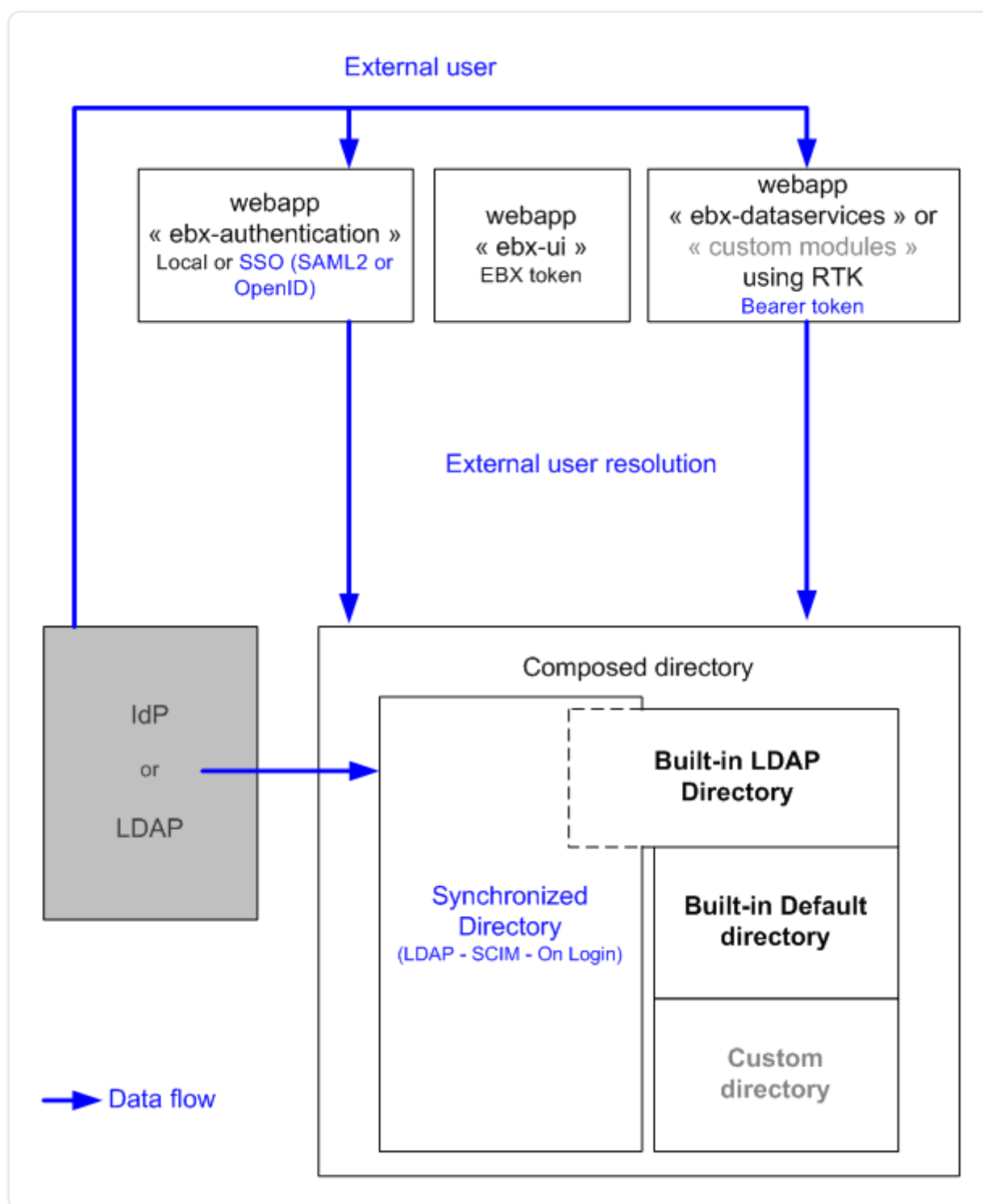
EBX® がユーザーをカスタムログインページにリダイレクトする場合、`resume` クエリパラメーターも URL に追加します。この `resume` クエリパラメーターには、`/ebx-authentication/login` を指す URL といくつかの追加パラメーターが含まれており、ユーザーを `/ebx-authentication/login` にリダイレクトするために優先的に使用する必要があります。

以下も参照してください。:

- セキュリティ構成

シングルサインオン (SSO)

シングルサインオンにより、ユーザーは個別のログイン資格情報を管理する代わりに、一元化されたIDプロバイダー (IdP) を介して一度認証するだけで、複数のアプリケーションやサービスにアクセスできます。この統合により、ログインプロセスが効率化され、パスワード関連のセキュリティ侵害のリスクが軽減され、シームレスなアクセスが可能になることでユーザーエクスペリエンスが向上します。



以下も参照してください。:

- 同期ディレクトリ

EBX® は SAML 2.0 および OpenID Connect プロトコルをサポートしています。これらの機能を有効にするには、IDプロバイダーでEBX®をアプリケーションとして宣言および構成する必要があります。構成要件は、有効になっているプロトコルによって異なります。SAML 2.0 を使用する場合、起動時に *sp-descriptor.xml* が生成され、シームレスな構成を確実にするためにIDプロバイダーにインポートする必要があります。

リバースプロキシ構成の場合、サーバーのベースURLは両方のプロトコルのプロパティで構成する必要があります。

注意:

一度に1つのプロトコルのみを構成できます。

注意:

IdPから開始されたログアウトプロセスは、ログアウトに使用されたブラウザーにのみ適用されます。

注意:

デフォルトのセキュリティ設定では、クロスサイトPOSTリクエスト中にCookieが送信されるのを防ぐため、SSO認証がブロックされます。この問題を回避するには、認可Cookieを `Lax` に設定します: `ebx.security.authorizationCookie.attribute.sameSite=Lax`。

構成されたプロトコルを介してログイン/ログアウトを開始するために使用されるエンドポイントは次のとおりです。

```
http[s]://<host>[:<port>]/ebx-authentication/rest/sso/v1/login
```

注意:

`resume` クエリパラメーターは、認証が完了した後にクライアントがリダイレクトされるべきエンコードされたURLを保持するために使用できます。

```
http[s]://<host>[:<port>]/ebx-authentication/rest/sso/v1/logout
```

内部ディレクトリを使用する組み込みまたはカスタムログインページにアクセスまたはリダイレクトする必要がある場合は、次のURLを使用できます。

```
http[s]://<host>[:<port>]/ebx-authentication/login?method=loginPage
```

SAML 2.0

SAML 2.0 を有効にするには、**サービスプロバイダーのプロパティ** または EBX® Container Edition の **環境変数** を構成し、リポジトリフォルダーにキーストア、`idp-descriptor.xml`、および `sp-descriptor.xml` を配置します。

- **キーストア:** すべての署名および暗号化操作には、キーストアを構成する必要があります。新しいキーストアを生成するには、次のJava keytoolコマンドを使用します。

```
keytool -genkeypair -alias ebx.saml -keypass keypass -keystore  
keystore.p12 -storepass keypass -keyalg RSA -keysize 2048 -validity 3650
```

- **idp-descriptor.xml:** IDプロバイダーからエクスポートされた *IDPSSODescriptor* XML ファイル。
- **sp-descriptor.xml:** 構成されたプロパティを使用してEBX®が起動時に生成する *SPSSODescriptor* XML ファイル。

構成された `nameId` はアサーションから抽出され、EBX®ディレクトリと照合されてユーザーを認証します。したがって、`nameId` 属性または形式を正確に構成することが重要です。

`urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST` バインディングタイプは、すべての操作に使用されます。

アサーションコンシューマーサービスとシングルログアウトサービスの両方に同じエンドポイントが使用されます。

```
http[s]://<host>[:<port>]/ebx-authentication/rest/sso/v1/saml2/callback
```

注意:

プロパティの変更により、新しい *sp-descriptor.xml* の生成と、既存の *keystore.jks* および *idp-descriptor.xml* の読み込みがトリガーされます。

OpenID Connect

OpenID Connect を有効にするには、[クライアントプロパティ](#) または EBX® Container Edition の [環境変数](#) を構成します。OpenID を使用する場合:

- 使用されるクライアント認証方法は *Client Secret Post* です。
- レスポンスモードは *form_post* で、レスポンスタイプは *code* です。

リダイレクト操作とログアウト操作の両方に同じエンドポイントが使用されます。

```
http[s]://<host>[:<port>]/ebx-authentication/rest/sso/v1/oidc/callback
```

パフォーマンスとチューニング

- [環境](#)
 - [メモリ管理](#)
 - [CPU](#)
 - [サーバー起動時のスキャン](#)
- [データベース](#)
 - [データベーステーブルの再編成](#)
- [データモデリング](#)
 - [集約リスト](#)
 - [継承されたフィールド](#)
- [データ検証](#)
- [テーブルへのアクセス](#)

- 機能
- テーブルのクエリ
 - アーキテクチャとデザイン
 - マルチスレッドインデックスビルドの設定
- REST ビルトインオブジェクトとビジネスオブジェクト
- 履歴テーブルへの REST アクセス
- フェッチサイズの設定
- 他の Java カスタマイズのパフォーマンスチェックリスト
 - 高度なプログラム拡張
 - 不要なインデックスの更新
 - 一括更新のトランザクションしきい値
 - トリガー
 - ディレクトリ統合

環境

メモリ管理

メモリ監視	EBX® ロードアクティビティの兆候は、基盤となるデータベースを監視することによって、また「監視」ログカテゴリによって提供されます。 <i>cleared</i> および <i>built</i> オブジェクトの数が長期間高いままである場合、これは EBX® がアプリケーションサーバーでスワップしていることを示しています。その場合、アプリケーションサーバーに割り当てるメモリを増やす必要があります。
ガベージコレクター	ガベージコレクターを調整すると、全体的なパフォーマンスにもメリットがあります。このチューニングは、使用するユースケースと

特定の Java ランタイム環境に適合させる必要があります。

CPU

アプリケーションサーバーで使用可能な CPU の数は、処理される同時 HTTP 要求の数、暗黙のタスクの複雑さ (CPU コスト)、および Java ガベージコレクションを含むバックグラウンドアクティビティを考慮して定義する必要があります。

サーバーの負荷と使用可能なプロセッサの数の違いにより、トランザクション内でインデックス作成を効率的に並行して実行できる場合、大規模なインポート、およびより一般的には多くの作成、更新、削除を伴う大規模なトランザクションがより速く完了します。永続性 ログカテゴリには、次のエントリが含まれます。

- *「強制同期インデックスを false に設定 (...)」*は、インデックスが同時に実行されることを示します。
- *「強制同期インデックスを true に設定 (...)」*は、インデックスが同時に実行されないことを示します。

上記の違いは 10 秒ごとに評価され、Java クラス

`java.lang.management.OperatingSystemMXBean` のメソッド `getSystemLoadAverage()` および `getAvailableProcessors()` を使用して計算されます。両方の数値は、上記のログエントリの最後に入力されます。

サーバー起動時のスキャン

Web アプリケーションサーバーの起動を高速化するには、JAR ファイルスキャナーを構成する必要があります。

データベース

データベーステーブルの再編成

他のデータベースと同様に、大量のデータを挿入および削除すると、データが断片化され、時間の経過とともにパフォーマンスが低下する可能性があります。この問題を解決するには、影響を受けるデータベーステーブルを再編成する必要があります。リレーショナルデータベースの監視とクリーンアップを参照してください。

EBX® の特徴は、データスペースとスナップショットを作成すると、テーブル `GRS_DTR` と `GRS_SHR` に新しいエントリが追加されることです。パフォーマンスが低下した場合、多くのデー

タスペースが作成および削除される大規模なリポジトリでは、これらのテーブルの再編成をスケジュールする必要がある場合があります。

以下も参照してください。:

- [リレーショナルデータベースの監視とクリーンアップ](#)

データモデリング

集約リスト

データモデルでは、エレメントのカーディナリティ制約 `maxOccurs` が 1 より大きく、このエレメントで `osd:table` が宣言されていない場合、Java `リスト` として実装されます。このタイプのエレメントは、テーブルではなく、[集約リスト](#)と呼ばれます。

反復、ユーザーインターフェイスの表示などの観点から、集約リストにアクセスする際に特定の最適化がないことを考慮することが重要です。パフォーマンスの問題に加えて、集約リストは、テーブルでサポートされる多くの機能に関して制限されます。これらの機能のリストについては、[テーブルの概要](#)を参照してください。

注意:

上記の理由により、集約リストは、小規模な単純データ (1 ダースまたは 2 ダースのレコード) にのみ使用する必要があり、識別、ルックアップ、アクセス許可などの高度な要件はありません。大規模なデータ (またはより高度な機能) の場合、`osd:table` 宣言を使用することをお勧めします。

継承されたフィールド

データモデルでは、データセットの継承ではなく、関係に基づく高度な継承のために [継承フィールド](#) を使用できます。

継承されたフィールドはインデックスの最適化の恩恵を受けることができないため、値を解決するために計算フィールドとして動作するため、注意して使用するか、使用を避ける必要があります。その結果、継承されたフィールドで実行されるすべての操作 (クエリ、検証、解決モードでのデータ比較など) は最適化されず、パフォーマンスに大きな影響が出る可能性があります。

データ検証

内部検証フレームワークは、データセットまたはテーブルの検証レポートを更新するための連続したリクエスト中に必要な作業を最適化します。インクリメンタル検証プロセスは次のように動作します。

- データセットまたはテーブル検証レポートへの最初の呼び出しは、データセットまたはテーブルの完全な検証を実行します。
- 検証レポートへの次の呼び出しは、最後の検証以降に実行された変更を計算します。検証レポートは、これらの変更に応じて更新されます。
- 検証レポートは、TIBCO EBX® リポジトリに永続的に保存されます。これにより、データセットに大量の検証メッセージがある場合に、検証レポート専用のメモリの量が削減されます。また、アプリケーションサーバーの再起動時に検証レポートが失われることはありません。
- 検証レポートは、管理者ユーザーがユーザーインターフェイスで手動でリセットできます (このオプションは、EBX® の検証レポートセクションから利用できます)。結果として、関連付けられたデータセットまたはテーブルは、次の検証レポートの呼び出し時に完全に再検証されるため、検証レポートのリセットは注意して使用する必要があります。

詳細については、`Adaptation.resetValidationReport` を参照してください。

最後の検証以降に更新が発生していなくても、特定の制約は体系的に再検証されます。これらは、**不明な依存関係**の制約です。次の場合、エレメントには不明な依存関係があります。

- デフォルトの**不明な依存関係**モードでプログラムによる制約を指定します。
- **計算値**を宣言するか、それ自体が**計算値**であるエレメントに依存する動的ファセットを宣言します。
- これは**継承されたフィールド**であるか、それ自体が**継承されたフィールド**であるノードに依存する動的ファセットを宣言します。

注意:

大規模なテーブルを検証する際のパフォーマンスの問題を回避するために、継承されたフィールドの検証を無効にすることができます。詳細については、**継承されたフィールド**を参照してください。

したがって、大規模なテーブルでは、次のことをお勧めします。

- 依存関係が不明な制約は避けてください (または少なくともそのような制約の数を最小限に抑えるため)。プログラムによる制約の場合、開発者は、増分検証コストを大幅に削減する2つの代替モードを指定できます。ローカル依存関係モードと明示的依存関係です。詳細については、[依存関係と検証](#)を参照してください。
- フィールドレベルで定義された[プログラムによる制約](#)の代わりに、[テーブルの制約](#)を使用します。実際、テーブルがフィールドレベルで制約を定義している場合、検証プロセスはすべてのレコードに対して繰り返され、関連付けられたフィールドの値が制約に準拠しているかどうかを確認します。[テーブルの制約](#)を使用すると、テーブル全体で最適化されたクエリを実行できます。このようなクエリを最適化する方法に関する推奨事項については、[下記](#)を参照してください。ただし、トレードオフがあります。[テーブルの制約](#)は増分検証の恩恵を受けず、常に完全に検証されます。通常、検証の間に作成または削除されるレコードがわずかしかない場合は、[プログラムによる制約](#)の方がパフォーマンスが向上する可能性があります。ベンチマークまたはサンプリングプロファイラーは、決定を下すのに役立ちます。

次のプロパティを使用して、検証レポートをログに記録するときのパフォーマンスへの影響を最小限に抑えることができます。

- `ebx.validation.report.logContent` プロパティを `false` に設定して、個々の検証メッセージをログに記録しないようにします。メッセージ数を含む検証レポートの概要は引き続きログに記録されます。Java API によって開始された検証のログを完全に回避することもできます。詳細については、`ValidationSpec.setResultLogged` を参照してください。
- `ebx.validation.report.maxItemDisplayed` プロパティをデフォルトの 100 よりも低い値に設定すると、前の推奨事項よりも効果が低くなりますが、ログに記録する検証メッセージの量が減るため、関連する計算作業が軽減されます。

テーブルへのアクセス

機能

テーブルには通常、EBX® UI、データサービス、[リクエスト](#)、[クエリ](#) API を介してアクセスされます。このアクセスには、*動的解決プロセス*を含む独自の機能セットが含まれます。このプロセスは次のように動作します。

- **継承**：データセットツリーの継承では、再帰的なプロセスを使用して、親データセットで定義されているレコードと値が考慮されます。また、ルートデータセットでは、レコードは、`xs:default` 属性で定義されたデータモデルのデフォルト値からその値の一部を継承できます。
- **値の計算**：`osd:function` として宣言されたノードは、値にアクセスすると常にオンザフライで計算されます。`ValueFunction.getValue` を参照してください。

- **フィルター**：XPath 述語、プログラムフィルター、またはレコードレベルの権限ルールでは、レコードを選択する必要があります。
- **並べ替え**：結果レコードの並べ替えを実行できます。

テーブルのクエリ

アーキテクチャとデザイン

EBX® エンジンでは、テーブルでの操作速度を向上させるために、永続的な Lucene インデックスを管理します。これは、次の方法で行われます。

- **インデックス再構築チェーンの最適化**: インデックスの再構築プロセスはチェーンとして考えられます。このチェーンは、テーブルの履歴を構成する一連のテーブルリビジョン、つまりリンクで構成されています。これらの各リンクは、スナップショットまたは子データスペースの作成の結果です。可能な場合、インデックスの再構築は再構築チェーン内のリンクをスキップし、最大 5 つのリンクに削減します。これにより、ベースインデックスの効果的な再利用を通じて、再構築期間とインデックス構造が改善されます。再構築チェーンで一部のリンクがスキップされた場合、次のメッセージが `persistence.log` ファイルに記録されます。 `"Index rebuild chain size is" + <fullSize> + ", reducing to 5."`

このプロセスは中間スナップショットの再構築時間を短縮しますが、スキップされたスナップショットへのアクセスは、インデックスが最も近い利用可能なポイントから増分的に再構築される必要があるため、わずかに時間がかかる場合がありますことに注意してください。

- **増分再構築の実現可能性の動的評価**: EBX® は、増分インデックス再構築を行うか、完全なインデックス再構築を行うかを決定するために、テーブルリビジョン間の「距離」を推定します。ベースインデックスとターゲットインデックスの差が大きすぎる場合、完全な再構築が開始されます。これにより、再構築が高速化され、インデックス作成がより効率的になります。この分岐点よりも古いスナップショットにアクセスすると、新しい再構築チェーンを最初から計算する必要があるため、時間がかかる場合があります。再構築チェーンで完全な再構築が必要な場合、次のメッセージが `persistence.log` ファイルに記録されます。 `"Breaking revision chain at table revision " + aRevision.getId() + ": too different from the rebuild target."`

注意:

チェーンの最初のリビジョンが最初から再構築される場合、またはテーブルリビジョンが小さすぎて増分再構築の複雑さに見合わない場合は、上記のメッセージは記録されません。

- **インデックス再構築の処理のバッチ化:** インデックス再構築操作は、効率を最適化するために削除アクションと作成アクションをグループ化します。
- **マルチスレッド再構築:** マルチスレッド化により、並列処理を通じて再構築パフォーマンスが向上します。インデックス再構築中に使用されるスレッドの数を制御できます。詳細については、[マルチスレッドインデックスビルドの設定](#)を参照してください。

注意:

インデックスの準備ができて OS メモリキャッシュに保持されている場合は、テーブルへのより高速なアクセスが保証されます。[上記](#)で述べたように、OS には十分なスペースが割り当てられていることが重要です。

パフォーマンスの考慮事項

クエリオプティマイザーは、リクエスト結果を計算するときにインデックスの使用を優先します。クエリがインデックスを利用できない場合、Java メモリで解決され、大容量でパフォーマンスが低下します。次のガイドラインが適用されます。

注意:

- インデックスの最適化の恩恵を受けることができるのは、XPath 述語と SQL クエリだけです。
- [制限事項](#)のセクションで説明されているように、一部のフィールドと一部のデータセットはインデックスに登録できません。
- `osd:label` 関数を使用する XPath 述語は、インデックスの最適化の恩恵を受けることができません。

インデックスがまだ作成されていない場合は、テーブルへの最初のアクセス時に、インデックスを作成して永続化するために追加の時間が必要です。

テーブルデータブロックへのアクセスは、クエリをインデックスに対して計算できない場合 (ルールの解決、フィルター、または並べ替えのいずれの場合でも)、およびインデックスの構築に必要な

です。テーブルブロックがメモリに存在しない場合、データベースからそれらを取得するために追加の時間が必要です。

それとは別に、リクエスト/クエリのパフォーマンスに影響を与える可能性があるその他の考慮事項があります。

- フィルター値のバリエーションだけで同じリクエスト/クエリが複数回実行される場合は、上記のフィルターでパラメーターを使用することをお勧めします。
 - `request.setXPathFilter("./field=$param");`
 - `SELECT * FROM myTable WHERE field=?`
- 可能であれば、`RequestResult.isSizeGreaterOrEqual` ではなく `RequestResult.getSize` を使用してください。
- 可能であれば、`RequestResult` サイズ関連の呼び出しは避けてください。特に、`RequestResult` で不要なコード パターン「走査する前に isEmpty をチェックする」を避けてください。

```
try (RequestResult result = ...) {

    if (result.isEmpty())

        return; // unnecessary and inefficient!

    for (Adaptation record : result) {

        ...

    }

}

// simply do:

try (RequestResult result = ...) {

    for (Adaptation record : result) {
```

```

        ...
    }
}

```

メモリ監視を介して情報を取得し、ログカテゴリを要求することができます。

テーブルへのアクセスと変更

次のアクセスはパフォーマンスの低下につながるため、回避する必要があります。

- いくつかの変更を繰り返した後、テーブルにアクセスします。これは、変更のたびにインデックスの状態が更新されることを意味します。更新のコストにより、このパターンは無効になります。代わりに、単一のクエリを実行し、結果を参照するときに変更を適用します。
- 前のケースと同時に、同じテーブルへの継続的なアクセスがある場合、古いインデックスファイルが削除されるのを防ぎます。その結果、ディスク上のインデックスのサイズが大きくなり、極端な場合にはサーバーのディスク容量が不足する可能性があります。同時アクセスが閉じられると、インデックスサイズは通常に戻ります。これは通常、リクエストまたはクエリが適切に閉じられていないことを示しています。

以下も参照してください。:

- `RequestResult.close`
- `QueryResult.close`

テーブルに対するその他の操作

新しいレコードの作成またはレコードの挿入は、主キーインデックスによって異なります。したがって、このインデックスがすでにロードされている場合、作成はほぼ即時になります。

マルチスレッドインデックスビルドの設定

インデックス再構築中に使用されるスレッドの数を調整するには、`ebx.properties` ファイルの `ebx.index.threads` プロパティを使用します。このプロパティのデフォルト値は「2」であり、ほとんどの運用シナリオで最適であるはずですが、スレッド数を増やすと、サーバーでの同時アクティビティに悪影響を及ぼす可能性があるため、この方法はテスト環境に限定することをお勧め

します。ただし、製品のアップグレード、移行、またはデータモデルへの大幅な変更により、広範なインデックス再構築が必要な場合は、デプロイフェーズ中に次の点を考慮してください。

- 高レベルの同時実行を可能にする高性能ハードウェア構成がある場合は、`ebx.index.threads` を `CPU count/2` の値に設定し、最大 10 までにすることを目指します。
- 追加のスレッドは、十分に大きなテーブルに対してのみ開始されます。正確な値は固定されていませんが、通常は 200,000 レコードよりも大きいです。

REST ビルトインオブジェクトとビジネスオブジェクト

ビジネスオブジェクトで選択操作を使用する場合、EBX® は一時フォルダーを使用して大きな応答コンテンツを処理します。管理者は、`ebx.dataservices.rest.bo.maxResponseSizeInKB` 構成パラメーターを使用してサイズ制限を設定できます。一時コンテンツ専用のサーバーの使用可能なファイルシステム領域と、要求処理スレッドの最大数を考慮して、この設定を調整します。

詳細については、`ebx.dataservices.rest.bo.maxResponseSizeInKB` および [一時ファイルディレクトリの設定](#) を参照してください。

履歴テーブルへの REST アクセス

履歴テーブル (`merge_info` フィールド) のマージ情報は、アクセスコストが高くなる可能性があります。パフォーマンスを向上させるために、クライアントコードがこのフィールドを必要としない場合は、`includeMergeInfo` パラメーターを `false` に設定する必要があります。

詳細については、[履歴](#) を参照してください。

フェッチサイズの設定

パフォーマンスを向上させるには、テーブルに対するリクエストの結果の予想サイズに応じてフェッチサイズを設定する必要があります。フェッチサイズが設定されていない場合は、デフォルト値が使用されます。

- 履歴テーブルでは、デフォルト値が JDBC ドライバーによって割り当てられます。Oracle の場合は 10、PostgreSQL の場合は 0 です。

注意:

PostgreSQL では、デフォルト値の 0 は、結果セット全体を一度にフェッチするように JDBC ドライバーに指示します。これにより、大量のデータを取得するときに

`OutOfMemoryError` が発生する可能性があります。一方、PostgreSQL で `fetchSize` を使用すると、トランザクションの最後にサーバー側のカーソルが無効になります。同じスレッドで、最初に `fetchsize` を使用して結果セットをフェッチし、次にトランザクションをコミットするプロシージャを実行すると、次の結果にアクセスすると例外が発生します。

以下も参照してください。:

- `Request.setFetchSize`
- `RequestResult`

他の Java カスタマイズのパフォーマンスチェックリスト

TIBCO EBX® は大量のデータをサポートするように設計されていますが、いくつかの一般的な要因によりパフォーマンスが低下する可能性があります。このセクションで説明する重要なポイントに対処することで、通常のパフォーマンスのボトルネックを解決できます。

高度なプログラム拡張

参考までに、次の表に、実装できるプログラム拡張機能の詳細を示します。

ユースケース	関与する可能性のあるプログラム拡張機能
検証	• プログラム上の制約 • 計算値
テーブルアクセス	• レコードレベルの権限ルール • プログラムフィルター
EBX® コンテンツ表示	• 計算値 • UI コンポーネント • ノードレベルの権限ルール

ユースケース	関与する可能性のあるプログラム拡張機能
データの更新	トリガー

大量のデータの場合、計算が非常に複雑なアルゴリズムを使用すると、パフォーマンスに深刻な影響を及ぼします。たとえば、制約のアルゴリズムの複雑さは $O(n^2)$ です。データサイズが 100 の場合、結果のコストは 10000 に比例します (これにより、通常、すぐに結果が得られます)。ただし、データサイズが 10000 の場合、結果のコストは 10000000 に比例します。

パフォーマンスが低下するもう 1 つの理由は、外部リソースの呼び出しです。ローカルキャッシングは通常、このタイプの問題を解決します。

上記のユースケースの 1 つでパフォーマンスが低下する場合は、コード分析または Java プロファイリングツールを使用して問題を追跡することをお勧めします。

不要なインデックスの更新

Lucene インデックスの更新には時間がかかります。可能な限り避ける必要があります。

更新のタイミング	トランザクションのコンテキストでは、テーブルが変更され、次のいずれかの条件が発生すると、インデックスの更新が発生します。 - 主キーによるルックアップの場合、検索されたキーが現在の プロシージャ (または TableTrigger) で「タッチ」(作成、変更、または削除) された場合、常に更新がトリガーされます。 - 標準の クエリ (または リクエスト) の場合、現在の プロシージャ (または TableTrigger) でテーブルが変更されていると、インデックスの更新が常に実行されます)。
----------	---

コーディングの推奨事項

1. 主キーによるルックアップによる更新のトリガーを回避するには、開発者は、`doCreateOccurrence` または `doModifyContent` への最後の呼び出しから返された `Adaptation` オブジェクトを登録する必要があります。ルックアップを実行する代わりに、このオブジェクトを再利用します。
2. 現在のプロシージャで削除されたレコードの主キーによるルックアップは避けてください。
3. 更新をトリガーするクエリの場合、開発者はこのクエリをプロシージャで回避できるかどうかを検討する必要があります。

一括更新のトランザクションしきい値

通常、トランザクション内のアトミック更新の数が 105 のオーダーを超える場合は、単一のトランザクションを使用することはお勧めしません。大規模なトランザクションには、EBX® および基盤となるデータベースからの多くのリソース、特にメモリが必要です。

トランザクションサイズを減らすために、次のことが可能です。

- プロパティ `ebx.manager.import.commit.threshold` を指定します。ただし、このプロパティは、EBX® ユーザーインターフェイスから実行されるインタラクティブなアーカイブのインポートにのみ使用されます。
- バッチプロシージャ内で **コミットしきい値** を明示的に指定します。
- タスクの一部に `プロシージャ` を実装し、必要な回数だけ実行することで、トランザクションの範囲を構造的に制限します。

一方、非常に小さいトランザクションサイズを指定すると、コミットごとに実行する必要のある永続的なタスクが原因で、パフォーマンスが低下する可能性もあります。

注意:

トランザクションのアトミック性が保証されなくなったために中間コミットが問題になる場合は、専用のデータスペース内で一括更新を実行することをお勧めします。このデータスペースは、一括更新の直前に作成されます。更新が正常に完了しない場合は、データスペースを閉じて、最初の失敗の理由を修正した後、更新を再試行する必要があります。成功すると、データスペースを元のデータスペースに安全にマージできます。

トリガー

必要に応じて、メソッド `ProcedureContext.setTriggerActivation` を使用してトリガーを非アクティブ化できます。

ディレクトリ統合

認証と権限の管理には、[ユーザーとロールのディレクトリ](#)が含まれます。

特定のディレクトリ実装が展開され、外部ディレクトリにアクセスする場合、ローカルキャッシュが確実に実行されるようにすると便利です。特に、最も頻繁に呼び出されるメソッドの 1 つは、

`Directory.isUserInRole` です。

構成に関する注意事項

- [アプリサーバーに割り当てられたメモリ](#)
- [OS に割り当てられたメモリ](#)

アプリサーバーに割り当てられたメモリ

クエリエンジンは永続ストレージから必要な情報を取得するため、Java 仮想マシンに割り当てられるメモリ (通常は `-Xmx` パラメータで指定) を低く抑えることができます。この数値は、OS で使用可能な合計メモリの 1/3 未満に設定することをお勧めします。また、32 GB 未満に抑えることをお勧めします。これは、すべての合理的なユースケースに適合し、[圧縮された通常のオブジェクトポインター](#)機能を利用できるようにする必要があります。

OS に割り当てられたメモリ

アプリケーションサーバーを実行している OS では、OS キャッシュに十分なスペースを残して、永続的な Lucene インデックスへのアクセスを最適化できるようにすることが重要です。実際、これらがファイルシステムからロードされると、OS はメモリキャッシュを使用して、この同じデータへの後続のアクセスを高速化し、ディスクから毎回リロードすることを回避します。これは、この目的のために十分な RAM が残っている場合にのみ可能です。

また、JVM プロセスが多数のメモリマップトファイルに必要なリソースを予約できるように OS を構成する必要があります (詳細については、この[記事](#)を参照してください)。Linux OS では、これは次のコマンドを発行することで実行できます。

```
ulimit -n 512000  
  
sysctl vm.max_map_count=262144
```

EBX® Container Edition

- [イメージのビルド](#)
- [イメージの実行](#)
- [イメージのカスタマイズ](#)

イメージのビルド

- [概要](#)
- [要件](#)
- [イメージのビルド](#)
 - [インストーラーのダウンロード](#)
 - [インタラクティブインストーラーの実行](#)
 - [バッチインストーラーの実行](#)
 - [イメージのテスト](#)
 - [イメージの共有](#)
 - [AMD64 以外のアーキテクチャ](#)

概要

TIBCO EBX® Container Edition イメージは、複数のアーキテクチャに対応した Linux OS ベースで、次のものが含まれています。

- [アプリケーションサーバー Apache Tomcat® 10.1](#)
- [Java JDK 25 LTS](#)

要件

イメージは、POSIX ネイティブシステム上で、または互換性のあるレイヤーを使用してビルドできます。

- オペレーティングシステム：Linux、macOS、Windows (10 バージョン 2004、Server 2022 以降)、WSL2 を使用
- プロセッサタイプ：AMD64 (Intel 64 ビット)、ARM64
- Docker Engine がインストールされ、実行されている
- インターネットへのアクセス

イメージのビルド

インストーラーのダウンロード

EBX® イメージを構築するには、TIBCO eDelivery からファイル `TIB\ebx\6.2.3\addon\6.2.X\container\edition.zip` をダウンロードする必要があります。

インタラクティブインストーラーの実行

Linux または macOS でインストーラーを起動するには、次の手順を実行します。

- `TIB\ebx\6.2.3\addon\6.2.X\container\edition.zip` を解凍します。
- ファイル `ebx-ce-installer.sh` が配置されているフォルダーでターミナルを開きます。
- コマンド `./ebx-ce-installer.sh` を実行します。

Windows でインストーラーを起動するには、次の手順を実行します。

- `TIB\ebx\6.2.3\addon\6.2.X\container\edition.zip` を解凍します。
- ファイル `ebx-ce-installer.bat` が配置されているフォルダーで Windows PowerShell を開きます。
- コマンド `ebx-ce-installer.bat` を実行します。

指示に従って、イメージに追加するオプションのコンポーネントを選択します。メタデータを選択した場合、対話型インストーラーによって、DAMA、DINT、DMDV、DPRA、および TESE のアドオンが自動的に追加されます。

次に、インストーラーはイメージを作成し、次のような要約を表示します。

```
*****
*****
```

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.

```
*****
*****
```

オプションを選択した場合、インストール時に、EBX® Container Edition を構築する追加のアーキテクチャの入力を求められます。その場合、Docker イメージタグには、構築されたアーキテクチャが含まれます。たとえば、**ebx:6.2.3-dama-dint-dmdv-dpra-mame-metadata-tese-6.2.X-amd64** です。

バッチインストーラーの実行

Linux または macOS でバッチインストーラーを起動するには、次の手順を実行します。

- **TIB\ebx\6.2.3\addon\6.2.X\container\edition.zip** を解凍します。
- **ebx-ce-installer-batch.sh** ファイルが存在するフォルダーでターミナルを開きます。
- 次のコマンドを実行します。 **./ebx-ce-installer-batch.sh**

```
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 (小文字で、空白で区切る)

                  for example: mame tese
```

Windows でバッチ インストーラーを起動するには、次の手順を実行します。

- **TIB\ebx\6.2.3\addon\6.2.X\container\edition.zip** を解凍します。
- **ebx-ce-installer-batch.bat** ファイルが存在するフォルダーで Windows PowerShell を開きます。
- 次のコマンドを実行します。 **ebx-ce-installer-batch.bat**

これにより、バッチインストーラーがイメージをビルドし、インタラクティブインストーラーと同じ概要を出力します。

イメージのテスト

イメージは、次のコマンドを使用してローカルで実行できます。

```
docker run -p 8080:8080 -d ebx:latest
```

実稼働環境 では、**latest** タグを使用しないことが推奨されます。インストーラーは、選択したオプションコンポーネントに応じて、常に別のタグを生成します。

前の例を使用すると、オプションのコンポーネントが選択されているため、タグは **6.2.3-dama-dint-dmdv-dpra-mame-metadata-tese-6.2.X** になります。

- メタデータ
- Digital Asset Manager (DAMA)
- Data Exchange (New) (DINT)
- Data Model and Data Visualization (DMDV)
- Insight (New) (DPRA)
- Match and Merge (MAME)
- Information Search (TESE)

その後、次のコマンドを使用してイメージを実行できます。

```
docker run -p 8080:8080 -d ebx:6.2.3-dama-dint-dmdv-dpra-mame-metadata-tese-6.2.X
```


オプションコンポーネントが選択されていない場合、タグは **6.2.3** になります。

イメージは次のコマンドを使用して実行できます。

```
docker run -p 8080:8080 -d ebx:6.2.3
```

イメージの共有

イメージを共有する手順は、会社のインフラストラクチャーによって異なります。

次の例では、イメージは **myregistry** という名前の Docker プライベートレジストリにプッシュされます。

```
docker tag ebx:6.2.3 myregistry:5000/ebx:6.2.3
```

```
docker push myregistry:5000/ebx:6.2.3
```

イメージが複数のアーキテクチャ用にビルドされる場合、詳細な手順が **files/** サブディレクトリにある **README-MULTIARCH.md** ファイルに記載されています。

AMD64 以外のアーキテクチャ

インストーラーは AMD64 (Intel 64 ビット) でテストされています。これらのアーキテクチャでは、生成されたイメージのアーキテクチャは **linux/amd64** になります。

インストーラーは、別のアーキテクチャを使用するシステムで実行した場合でも、成功する可能性があります。その場合、生成されるイメージプラットフォームは、そのアーキテクチャタイプ用の Linux になります。

たとえば、インストーラーが macOS M1 ワークステーションで実行される場合、生成されるイメージのプラットフォームは **linux/arm64** になります。

注意

AMD64 (Intel 64 ビット) または ARM64 アーキテクチャ用の Linux イメージの生成は完全にサポートされています。その他のプラットフォームは**実験的**であり、**実稼働環境ではサポートされていません**。

イメージの実行

- [サポートポリシー](#)

- Docker
- Kubernetes
- EBX® の開始
 - 初回起動アシスタント
 - 自動初期化
 - サポートされるブラウザー
- コンテナへのアクセス
- 環境変数
 - 初回起動アシスタントの無効化
 - 初回起動時のリポジトリの自動インストール
 - EBX® ライセンスキーの定義
 - EBX® 実行モード
 - URL 構成
 - EBX® データベース接続
 - メタデータ分類子データベース接続
 - ディレクトリ接続
 - 同期ディレクトリ
 - LDAP 接続
 - SSO SAML 2.0 接続
 - SSO OpenID Connect 接続
 - ベアラー認証
 - SCIM ディレクトリ
 - メール接続
 - メモリ構成
 - REST サービスの認証
 - ステージング
- 構成ファイル
 - ebx-default.properties ファイル

- ebx-container.properties ファイル
- ボリューム
- Linux ユーザーとグループ
- ホスト構成
- ログアクセス

サポートポリシー

Docker

TIBCO EBX® Container Edition は、Docker Engine バージョン 20.10 でテストされています。

Kubernetes

現在の EBX® Container Edition リリースは、認定された Kubernetes 実装バージョン 1.26 および Red Hat® OpenShift® 4.13.1 でテストされました。

EBX® Container Edition の Helm チャートサンプルが GitHub から提供されています。

詳細については、<https://github.com/TIBCOSoftware/ebx-container-edition> を参照してください。

EBX® の開始

初回起動アシスタント

埋め込み H2 データベースを含むデフォルト構成で EBX® を起動するには、次のコマンドを実行します。

```
docker run -p 8080:8080 -d ebx:6.2.3
```

ブラウザを使用して、URL <http://localhost:8080> で EBX® に接続できます。これにより、EBX® の構成に役立つ初回起動アシスタントが表示されます。

初回起動アシスタントの詳細については、[初期化と初回起動アシスタント](#)の章を参照してください。

自動初期化

最初の起動時に自動初期化と埋め込み H2 データベースを使用して EBX® を起動するには、次のコマンドを実行します。

```
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
```

EBX® リポジトリは、最初の起動時に自動的に作成されます。

ブラウザーを使用して、URL <http://localhost:8080> で EBX® に接続できます。これにより、EBX® ログイン画面が表示されます。管理者のユーザー名は **admin** で、パスワードは前のコマンドで指定したものです。

注意

管理者に別のユーザー名を指定することができます。詳細については、[初回起動時のリポジトリの自動インストール](#) を参照してください。

サポートされるブラウザー

サポートされているブラウザーの詳細については、[サポートされる Web ブラウザー](#) を参照してください。

コンテナへのアクセス

次のコマンドは、デフォルトユーザーを使用して EBX® コンテナ内で bash シェルを開始します。

```
docker exec -it <container-id> bash
```

root として接続するには、次のコマンドを使用します。

```
docker exec -it --user root <container-id> bash
```

環境変数

この章では、**EBX® Container Edition** でサポートされている環境変数について説明します。

すべてオプションです。

初回起動アシスタントの無効化

セキュリティ上の理由から、すべての状況で初回起動アシスタントを無効にすることをお勧めします。

これは、環境変数 **EBX\FLA\DISABLED** を **true** にすることで設定されます。

初回起動時のリポジトリの自動インストール

リポジトリがまだ初期化されておらず、初回起動アシスタントが無効になっている場合、次の必須変数が指定されていれば、EBX® は自動的にインストールをトリガーします。

名称	デフォルト	説明
EBX\INSTALL\ADMIN\LOGIN	admin	EBX® 管理者のログイン名を設定します。リポジトリ変数 EBX\FLA\DISABLED の値が true でない場合、またはリポジトリがすでに初期化されている場合、このパラメーターは無視されます。
EBX\INSTALL\ADMIN\PASSWORD		EBX® 管理者のログイン名を設定します。このパラメーターは、 EBX\FLA\DISABLED の値が true の場合は必須であり、リポジトリ変数 EBX\FLA\DISABLED の値が true でない場合や、リポジトリがすでに初期化されている場合は無視されます。
EBX\INSTALL\LICENSE		EBX® ライセンスキーがまだ定義されていない場合に設定します。

注意

必須変数が指定されていない場合、EBX® はエラーメッセージを表示します。

例

リポジトリを自動的にインストールするには、次のコマンドを使用して EBX® を起動します。

```
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
```

EBX® ライセンスキーの定義

専用のライセンスキーで EBX® を起動するには、次のコマンドを実行します。

```
docker run -d -p 8080:8080 \

-e "EBX_INSTALL_LICENSE=XXXXX-XXXXX-XXXXX-XXXXX" \

ebx:6.2.3
```

注意: **EBX\INSTALL\LICENSE** は、ライセンスキーが永続化された場所にすでに定義されている場合、無視されます。

ライセンスキーは、**/ebx/data/ebx-license.properties** の場所に永続化されます。

詳細については、[ボリューム](#) を参照してください。

EBX® 実行モード

このプロパティは、EBX® の実行方法を定義します。

名称	デフォルト	説明
EBX_BACKEND_MODE	開発	3 つの実行モードが利用可能です: <i>開発</i> 、 <i>統合</i> 、 <i>本番</i> 。

```
docker run -d -p 8080:8080 \

-e "EBX_BACKEND_MODE=production" \

ebx:6.2.3
```

詳細については、[EBX® 実行モード](#) を参照してください。

URL 構成

一部の EBX® 機能では、URL を生成する必要があります。これを行うには、特定の構成が必要になる場合があります。たとえば、EBX® がリバースプロキシの背後または Kubernetes クラスターで実行されている場合です。

名称	デフォルト	説明
EBX_IS_SECURED	着信リクエストが HTTPS の場合、「true」が想定され、そうでない場合は「false」が想定されます。	「true」の場合、プロトコル「HTTPS」が常に想定されます。この値は、コンテナがリバースプロキシ、ファイアウォール、または HTTPS 暗号化を処理する Ingress の背後にある場合に役立ちます。これにより、EBX 認証 Cookie も secure に設定されます。これは、最新のブラウザーを使用しているユーザーは HTTP を使用してログインできないことを意味します。「false」の場合、プロトコル「HTTP」が想定されます。
EBX_AUTHENTICATION_REDIRECT_TO_HTTPS	デフォルトは「true」です。	「true」の場合、ログインフォームへの受信リクエストが HTTP であれば、HTTPS を強制するためにリダイレクトが発生します。コンテナがリバースプロキシ、ファイアウォール、または HTTPS 暗号化を処理する Ingress の背後にある場合は、このプロパ

名称	デフォルト	説明
		ティを false に設定する必要があります。
EBX_HOSTNAME	着信 HTTP(S) リクエストによって指定されたホスト名	EBX® サーバーのホスト名
EBX_PORT	着信 HTTP リクエストによって指定されたポート番号。	EBX® サーバーの HTTP ポート番号。
EBX_PORT_SECURED	着信 HTTPS リクエストによって指定されたポート番号。	EBX® サーバーの HTTPS ポート番号。 注意 HTTPS サポートは、暗号化を処理するリバースプロキシまたは Ingress によって提供される必要があります。
EBX_ROOT_PATH	デフォルトでは、コンテキストパスは空です。	設定すると、すべての EBX® URL にこの値がプレフィックスとして付けられます。値が / の場合を除き、値には先頭に / を付ける必要があります、末尾に / を付けることはできません。たとえば、有効な値は /mdm/sales です。この変数の設定は、同じホスト名で EBX の複数のインスタンスを実行する場合に役立ちます。
EBX_URL_DEFAULT		この環境変数は、バックグラウンドタスクが EBX への URL を計算する必要がある場合に使用されます。パスコンポーネントを含まない完全な URL に設定する必要があります (EBX_ROOT_PATH はパスコンポーネントに適用されます)。たとえば、有効な値は https://host_name です。EBX_URL_DEFAULT が無指定で、かつ EBX_HOSTNAME が指定されている場合、デフォルトは

名称	デフォルト	説明
		次の仮定に基づいて計算されます。 - EBX\IS\SECURED が true の場合、ポート番号が EBX\PORT\SECURED (デフォルトは 443) に等しい HTTPS が想定されます。 - EBX\IS\SECURED が false または設定されていない場合は、ポート番号が EBX\PORT (デフォルトは 80) に等しい HTTP が想定されます。

EBX® データベース接続

サポートされているデータベースについては、[サポートされるデータベース](#)の章を参照してください。

デフォルトでは、埋め込みの H2 データベースが使用されます。この H2 データベースのデータは、場所 **/ebx/data/h2** に保持されます。

外部データベースは、次の変数を使用して構成できます。

名称	同等の TIBCO EBX® メイン構成ファイル	説明
EBX\DB\FACTORY	ebx.persistence.factory	データベースサーバーの種類を指定します。
EBX\DB\URL	ebx.persistence.url	JDBC URL。形式は次のとおりです: jdbc:<dialect>://<database\host>:<database\port_numb<database_name> 。
EBX\DB\USER	ebx.persistence.user	データベースのユーザー ID
EBX\DB\PASSWORD	ebx.persistence.password	データベースユーザーのパスワード

これらの変数の詳細については、[EBX® リポジトリの構成](#)の章の TIBCO EBX® メイン構成ファイルに相当するものを参照してください。

注意

コンテナには、H2、PostgreSQL、Microsoft SQL Server 用の JDBC ドライバーのみが含まれています。EBX® でサポートされている他のデータベースを使用するには、ドライバを追加する必要があります。

ドライバを追加する方法については、[新しい JDBC ドライバーの追加](#)を参照してください。

EBX® を JNDI データソースを使用するように構成できます。詳細については、[JNDI でのデータベースの構成](#)を参照してください。

例

PostgreSQL データベースに接続する EBX® のインスタンスを起動するには、次のコマンドを実行します。

```
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
```

メタデータ分類子データベース接続

メタデータ分類子は、SQLAlchemy ツールキットを使用してデータベースに接続します。SQLAlchemy ツールキットは、ほとんどの一般的なデータベースをサポートしています。

デフォルトでは、埋め込み SQLite データベースが使用されます。この SQLite データベースのデータは、**`/ebx/data/sqlite/ebx-metadata-classifier.db`** の場所に永続化されます。

外部データベースは、次の変数を使用して構成できます。

名称	説明
EBX_METADATA_CLASSIFIER_DB_URL	資格情報なしの SQLAlchemy URL。形式は次のとおりです: <code><dialect>://<database_host>:<database_port>/<database_name></code> 。
EBX_METADATA_CLASSIFIER_DB_USER	データベースのユーザー ID
EBX_METADATA_CLASSIFIER_DB_PASSWORD	データベースユーザーのパスワード

コンテナには、デフォルトデータベース (SQLite) 用のドライバーのみが含まれています。SQLAlchemy でサポートされている他のデータベースを使用するには、ドライバーを追加する必要があります。

ドライバーを追加する方法については、[メタデータ分類子データベース用のドライバーの追加](#) を参照してください。

例

PostgreSQL データベースに接続する Metadata のインスタンスを起動するには、次のコマンドを実行します。

```
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
```

注意

- メタデータ分類子データベース名 (`database\_name`) は、[データベース接続](#) と同じにすることができます。
- 特定のデータベース API ドライバーの使用法と、接続 URL で特殊文字をエスケープする方法については、SQLAlchemy のドキュメントを参照してください。
- H2 データベースは SQLAlchemy ではサポートされていません。

ディレクトリ接続

EBX® ディレクトリは、次の環境変数を使用して構成できます。

名称	同等の TIBCO EBX® メイン構成ファイル	説明
EBX_DIRECTORY_FACTORY	ebx.directory.factory	Java ディレクトリファクトリクラス名を指定します (オプション)。
EBX_DIRECTORY_SYNCHRONIZED_SOURCE	ebx.directory.synchronized.source	同期ディレクトリの実装 (LDAP、SCIM、または OnLogin) を有効にして指定します。
EBX_SSO_PROTOCOL	ebx.directory.sso.protocol	SSO プロトコル (OIDC または SAML2) を有効にして指定します。

詳細については、[ユーザーおよびロールディレクトリの構成](#) を参照してください。

同期ディレクトリ

EBX® 同期ディレクトリは、次の環境変数を使用して構成できます。

名称	同等の TIBCO EBX® メイン構成ファイル (スペースは削除する必要があります)
EBX_DIRECTORY_SYNCHRONIZED_ROLE_MAPPER	ebx.directory.synchronized.role.mapper
EBX_DIRECTORY_SYNCHRONIZED_ROLE_MAPPER_DEFAULT_ADMINISTRATOR	role.mapper.default.administrator
EBX_DIRECTORY_SYNCHRONIZED_ROLE_MAPPER_DEFAULT_READ_ONLY	role.mapper.default.read_oly
EBX_DIRECTORY_SYNCHRONIZED_MAPPING_LDAP_ATTRIBUTES	mapping.ldap.attributes
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAPPING_SAML2_ATTRIBUTES_ROLE_NAME	sso.mapping.saml2.attributes.role.name

名称	同等の TIBCO EBX® メイン構成ファイル (スペースは削除する必要があります)
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAP_INCLUDE_SAML2_ATTRIBUTES	ebx.directory.sso.synchronized.sso.mapping.saml2.attributes
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAP_INCLUDE_SYNCHRONIZED_ROLE_NAME	ebx.directory.sso.synchronized.sso.mapping.oidc.claims.role.name
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAP_INCLUDE_SYNCHRONIZED	ebx.directory.sso.synchronized.sso.mapping.oidc.claims
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAP_INCLUDE_BEARER_CLAIMS_ROLE_NAME	ebx.directory.sso.synchronized.sso.mapping.bearer.claims.role.name
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAP_INCLUDE_BEARER_CLAIMS	ebx.directory.sso.synchronized.sso.mapping.bearer.claims
EBX_DIRECTORY_SYNCHRONIZED_SSO_MAP_INCLUDE_ATTRIBUTE_MAIL	ebx.directory.sso.synchronized.sso.mapping.attribute.mail
EBX_DIRECTORY_SYNCHRONIZED_SSO_DISPLAY_USER	ebx.directory.sso.synchronized.sso.display.user
EBX_DIRECTORY_SYNCHRONIZED_SSO_LISTENERS	ebx.directory.sso.synchronized.sso.listeners

LDAP 接続

既存の LDAP ディレクトリへの EBX® 統合は、次の環境変数を使用して構成できます。

名称	同等の TIBCO EBX® メイン構成ファイル (スペースは削除する必要があります)
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

名称	同等の TIBCO EBX® メイン構成ファイル (スペースは削除する必要があります)
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_REFERENCEATTRIBUTE	ebx.directory.ldap.default.mapping.role.referenceAttribute
EBX_LDAP_DEFAULT_MAPPING_ROLE_MEMBERATTRIBUTE	ebx.directory.ldap.default.mapping.role.memberAttribute
EBX_LDAP_DEFAULT_MAPPING_USER_REFERENCEATTRIBUTE	ebx.directory.ldap.default.mapping.user.referenceAttribute
EBX_LDAP_DEFAULT_MAPPING_USER_REFERENCECOMPONENTINDN	ebx.directory.ldap.default.mapping.user.referenceComponentInDN
EBX_LDAP_DEFAULT_MAPPING_ROLE_MAILATTRIBUTE	ebx.directory.ldap.default.mapping.role.mailAttribute
EBX_LDAP_DEFAULT_MAPPING_USER_MAILATTRIBUTE	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

USER_SEARCH、USER_GROUPS_SEARCH、ALL_USERS_SEARCH、ALL_GROUPS_SEARCH の各リクエストは、次の環境変数を使用して構成できます。

たとえば、USER_SEARCH リクエストは次のようになります。

名称	同等の TIBCO EBX® メイン構成ファイル (スペースは削除する必要があります)
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCHBASEDN	ebx.directory.ldap.default.request.userSearch.baseDN

名称	同等の TIBCO EBX® メイン構成ファイル (スペースは削除する必要があります)
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_FILTER	com.tibco.ebx.directory.ldap.default.request.userSearch.filter
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_SCOPE	com.tibco.ebx.directory.ldap.default.request.userSearch.scope
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_CACHE_EXPIRATION	com.tibco.ebx.directory.ldap.default.request.userSearch.cache.expirationInSeconds
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_CACHE_REFRESH	com.tibco.ebx.directory.ldap.default.request.userSearch.cache.refreshInSeconds
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_CACHE_MAX_SIZE	com.tibco.ebx.directory.ldap.default.request.userSearch.cache.maxSize
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_PAGE_SIZE	com.tibco.ebx.directory.ldap.default.request.userSearch.pageSize
EBX_LDAP_DEFAULT_REQUEST_USER_SEARCH_TIMEOUT	com.tibco.ebx.directory.ldap.default.request.userSearch.timeLimitInSeconds

詳細については、[組み込み LDAP ディレクトリ](#) を参照してください。

例

LDAP 接続でインスタンスを起動するには、次のコマンドを実行します。

```
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=(amp(objectClass=groupOf
Names)(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=groupOfNam
es)" \

-e "EBX_LDAP_DEFAULT_REQUEST_ALL_USERS_SEARCH_BASEDN=dc=example,dc=org" \

-e
"EBX_LDAP_DEFAULT_REQUEST_ALL_USERS_SEARCH_FILTER=(objectClass=inetOrgPers
on)" \

ebx:6.2.3

```

注意

LDAP ディレクトリを有効にするには、**EBX\DIRECTORY\FACTORY** を **com.orchestranetworks.service.directory.ldap.LdapDirectoryFactory** に定義する必要があります。

SSO SAML 2.0 接続

EBX® シングルサインオン (SSO) SAML 2.0 組み込みは、次の環境変数を使用して構成できます。

名称	同等の TIBCO EBX® メイン構成ファイル
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

詳細については、[SAML 2.0](#) を参照してください。

SSO OpenID Connect 接続

EBX® シングルサインオン (SSO) OpenID Connect 組み込みは、次の環境変数を使用して構成できます。

名称	同等の TIBCO EBX® メイン構成ファイル
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

詳細については、[OpenID Connect](#) を参照してください。

ベアラー認証

EBX® ベアラー認証は、次の環境変数を使用して構成できます。

名称	同等の TIBCO EBX® メイン構成ファイル (スペースは削除する必要があります)
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

名称	同等の TIBCO EBX® メイン構成ファイル (スペースは削除する必要があります)
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEYSTORE_TYPE	jwt.encryption.keystore.type
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEYSTORE_PATH	jwt.encryption.keystore.path
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEYSTORE_PASSWORD	jwt.encryption.keystore.password
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEYSTORE_KEY_ALIAS	jwt.encryption.keystore.key.alias
EBX_BEARER_DEFAULT_JWT_ENCRYPTION_KEYSTORE_KEY_PASSWORD	jwt.encryption.keystore.key.password
EBX_BEARER_DEFAULT_INTROSPECTION_REQUEST	introspection.bearer.default.request

詳細については、[ベアラー認証](#) プロパティを参照してください。

SCIM ディレクトリ

EBX® SCIM ディレクトリは、次の環境変数を使用して構成できます。

名称	同等の TIBCO EBX® メイン構成ファイル (スペースは削除する必要があります)
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

名称	同等の TIBCO EBX® メイン構成ファイル (スペースは削除する必要があります)
EBX_SCIM_DEFAULT_REQUEST_GROUPS_HEADERS	ebx.dir.scim.default.request.groups.headers
EBX_SCIM_DEFAULT_REQUEST_GROUPS_PAGE_SIZE	ebx.dir.scim.default.request.groups.page.size
EBX_SCIM_DEFAULT_REQUEST_GROUPS_CACHE_MAX_SIZE	ebx.dir.scim.default.request.groups.cache.max.size
EBX_SCIM_DEFAULT_REQUEST_GROUPS_CACHE_EXPIRATION	ebx.dir.scim.default.request.groups.cache.expiration

詳細については、[SCIM 機能](#) プロパティを参照してください。

メール接続

EBX メールサービスは、次の環境変数を使用して構成できます。

名称	同等の TIBCO EBX® メイン構成ファイル	デフォルト	説明
EBX_SMTP_HOST	ebx.mail.smtp.host		SMTP サーバーのホスト名
EBX_SMTP_PORT	ebx.mail.smtp.port		SMTP サーバーのポート番号
EBX_SMTP_CONNECTION_TIMEOUT	ebx.mail.smtp.connectionTimeout	10000	SMTP ソケット接続のタイムアウト値 (ミリ秒)。
EBX_SMTP_TIMEOUT	ebx.mail.smtp.timeout	600000	SMTP ソケット読み取りタイムアウト値 (ミリ秒)。
EBX_SMTP_WRITE_TIMEOUT	ebx.mail.smtp.writeTimeout	600000	SMTP ソケット書き込みタイムアウト値 (ミリ秒)。
EBX_SMTP_LOGIN	ebx.mail.smtp.login		SMTP サーバーのログイン ID

名称	同等の TIBCO EBX® メイン構成ファイル	デフォルト	説明
EBX_SMTP_PASSWORD	ebx.mail.smtp.password		SMTP サーバーのログインパスワード
EBX_SMTP_SSL_ENABLED	ebx.mail.smtp.ssl.activate	true	SSL を有効にします。値は「true」または「false」にすることができます。
EBX_WORKFLOW_MAIL_SENDER	ebx.workflow.mail.sender		ワークフローの送信者のメール。設定されていない場合、ワークフローは通知を送信できません。

使用されるプロパティの詳細については、[SMTP とメールの有効化と構成](#)の章を参照してください。

メモリ構成

環境変数 **JAVA_MEMORY_PERCENT** を使用して、EBX® を実行する JVM に割り当てられるコンテナメモリの割合を構成できます。0 から 100 の整数値にする必要があります。

設定されていない場合、起動時にデフォルト値が使用されます。

注意

この変数は高度な使用法のためのものです。設定が低すぎたり高すぎたりすると、ランタイムの問題が発生する可能性があります。

REST サービスの認証

REST サービスの基本認証はデフォルトでは有効になっていません。

この機能を有効にするには、環境変数 **EBX_REST_AUTHENTICATION_BASIC** を **true** に設定します。

ステージング

環境変数 **EBX_STAGING_ACTIVATED** は、値が **true** の場合はステージング機能をアクティブ化し、値が **false** の場合は非アクティブ化します。

デフォルトでは、ステージングはアクティブ化されています。

構成ファイル

現在、EBX®の構成には 2 つの Java プロパティファイルが使用されています。

起動時に、EBX® は次の順序でプロパティファイルを読み取ります。

- `/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties`
- `/myl_custom/conf/ebx-container.properties`

ebx-default.properties ファイル

ファイル `/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties` は、コンテナのデフォルトの EBX® 構成プロパティを設定します。

EBX® を次のバージョンに簡単に更新できなくなる可能性があるため、実行時に**決して**変更しないでください。代わりに `/opt/ebx/conf/ebx-container.properties` を使用してください。

ebx-container.properties ファイル

ファイル `/opt/ebx/conf/ebx-container.properties` はデフォルトで空です。ここで指定されたプロパティ値は、**ebx-default.properties** で設定された値をオーバーライドします。

このファイルは、実行時にプロパティを変更するのに役立ちます。実行時にプロパティを変更するには、新しいファイルを作成します。たとえば、`/myl_custom/conf/ebx-container.properties` のように、新しいプロパティ値を含み、親フォルダーをホストからコンテナにマウントします。

```
docker run -v /my-custom/conf:/opt/ebx/conf -p 8080:8080 -d ebx:6.2.3
```

EBX® でサポートされているプロパティのリストについては、[TIBCO EBX® メイン構成ファイル](#) の章を参照してください。

ボリューム

このイメージは、次のボリュームを定義します。

位置	説明
<code>/ebx/data</code>	EBX® ルートディレクトリはこのボリュームにあります。これには、EBX® インデックスと、H2 埋め込みデータベースが使用されている場合は永続化されたデータが含まれます。

位置	説明
	SSD などの高性能ボリュームを使用することをお勧めします。NFS マウントなどの共有ネットワークボリュームの使用は推奨されず、パフォーマンスと安定性の問題が発生する可能性があります。
/ebx/logs	このボリュームはログファイルに使用されます。
/ebx/temp	このボリュームは一時ファイルに使用されます。

すべての専用ボリュームで EBX® を起動するには、次のコマンドを実行します。

```
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
```

注意

ボリューム **/ebx/data** は、外部データベースが使用されている場合でも永続ボリュームにマップする必要があります。そうでない場合、EBX® は起動時にインデックスを再構築する必要があり、起動時間が大幅に増加する可能性があります。

Linux ユーザーとグループ

コンテナはユーザー **ebx** (uid 1500) を使用して起動されます。ユーザー ebx のプライマリグループは **root** (guid 0) です。

注意

Red Hat® OpenShift® は、コンテナの起動時に 1500 以外の UID を使用する場合があります。詳細については、Red Hat® OpenShift® のドキュメントを参照してください。

ホスト構成

EBX コンテナが多数のメモリマップファイルに必要なリソースを予約できるように、ホストを構成する必要がある場合があります。

Linux OS の場合

- コマンド `ulimit -n` は、**512000** 以上の値を返す必要があります。
- コマンド `sysctl vm.max_map_count` は、**262144** 以上の値を返す必要があります。

ログアクセス

ログは stdout および stderr 出力ストリームに送信され、次のコマンドを使用して表示できます。

```
docker logs <container-id>
```

EBX® と Tomcat の両方のログが表示されます。

ログファイルは、フォルダー `/ebx/logs` にもあります。

- EBX® ログファイルはフォルダー `/ebx/logs/ebx` にあります。
- Tomcat ログファイルはフォルダー `/ebx/logs/tomcat` にあります。

イメージのカスタマイズ

EBX® Container Edition イメージを親イメージとして使用して、カスタマイズされたイメージを作成できます。

- Docker ファイルで指定されたユーザー
- デフォルト構成の設定
- カスタムモジュールの追加
- 新しいロケールの追加
- 新しい JDBC ドライバーの追加
- JNDIでのデータベース設定
- JNDIでのメール設定
- メタデータ分類子データベースのドライバー追加

Docker ファイルで指定されたユーザー

EBX® Container Edition イメージの Docker ファイルは、ユーザー `ebx` を指定します。

イメージを拡張する Docker ファイルでは、次のサンプルのように、現在のユーザーを root に設定し、後で **ebx** に戻す必要がある場合があります。

```
FROM ebx:6.2.3

USER root

# Do something requiring being root...

USER ebx
```

デフォルト構成の設定

デフォルトの EBX® 構成の設定は、**/opt/ebx/conf/ebx-container.properties** に基づくべきではありません。このファイルは、実行時にオーバーライドされる可能性があるためです。

代わりに、Docker ファイルで次のように進めます。

- ファイル **/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties** の名前を、たとえば **ebx-default-original.properties** に変更します。
- 新しいファイル **/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties** コンテナの新しいプロパティ値を作成します。このファイルは、**ebx-default-original.properties** などの元のプロパティファイルの新しい名前に設定されたプロパティ **ebx.file.previous** を定義する必要があります。

EBX® でサポートされているプロパティのリストについては、**TIBCO EBX® メイン構成ファイル** の章を参照してください。

以下は、ロケールを「en-US」に設定したサンプルの Docker ファイルを示しています。

```
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

```

カスタムモジュールの追加

カスタムモジュールを開発することで、EBX® を拡張できます。EBX® モジュールは、XML スキーマドキュメント、Java クラス、静的リソースなどのさまざまなリソースをパッケージ化した標準の Jakarta EE Web アプリケーションです。

EBX® モジュールの詳細については、[TIBCO EBX® モジュールのパッケージ化](#) の章を参照してください。

EBX® Container Editionでは、モジュールを「アンパック (展開)」 WAR として展開することをお勧めします。これにより、起動が速くなり、Tomcat が WAR ファイルを解凍する必要がないため、コンテナサイズが不必要に大きくなるのを防ぐことができます。

モジュールをイメージに追加するための推奨される方法は、次のとおりです。

- WAR をフォルダー **/opt/ebx/webapps** にコピーします。前述のように、展開形式が推奨されます。
- **module_war_name.xml** という名前の関連する Tomcat コンテキスト XML ファイルを作成し、フォルダー **/opt/ebx/contexts** にコピーします。
- オプションで、共有 JAR をフォルダー **/opt/ebx/lib** にコピーします。

Tomcat コンテキスト XML ファイルには次の内容が含まれている必要があります。

```

<?xml version="1.0" encoding="UTF-8"?>

<Context docBase="${ebx.container.base}/webapps/module_war_name"/>

```

環境変数 **EBX_ROOT_PATH** を正しくサポートするには、変数 **\${ebx.container.base}** を使用する必要があります。

Tomcat コンテキストの詳細については、https://tomcat.apache.org/tomcat-10.1-doc/config/context.html#Defining_a_context を参照してください。

以下にサンプルの Docker ファイルを示します。

```
FROM ebx:6.2.3

USER root

COPY "./module_name.xml" "/opt/ebx/contexts"

COPY "./module_name" "/opt/ebx/webapps"

USER ebx
```

新しいロケールの追加

1つ以上のロケールを追加するには、言語パックソフトウェアのzipファイルが必要です。以下は、Dockerfile を作成し、ターゲットイメージをビルドするシェルスクリプトです。このスクリプトは、ニーズに合わせて変更できます。

```
#!/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}"
```

この例では、lib とオプションのドキュメントがコピーされ、[デフォルト構成の設定セクション](#)で説明されている方法を使用してロケールが設定されます。

新しい JDBC ドライバーの追加

新しい JDBC ドライバーの追加は、新しいライブラリの追加に似ています。正しい権限で「/opt/ebx/lib」フォルダーにある jar ファイルをコピーするだけです。以下は、Oracle JDBC ドライバーの例を示しています。

```
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
```

詳細については、[データベースドライバー](#)を参照してください。

JNDIでのデータベース設定

EBX® 用の JNDI データソースを設定するには、Docker イメージをカスタマイズする必要があります。以下の Dockerfile のサンプルは、**ebx.xml** 設定を統合するために必要な手順を示しています。

```
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
```

ebx.xml ファイルは、Web アプリケーションのコンテキストを管理し、EBX® で使用される JNDI データソースの重要な設定情報を含んでいます。

重要な点: 以下にリストされている各プロパティは、データベース (例: Oracle、PostgreSQL、SQL Server など) またはクラウドプロバイダーに応じて、慎重に調整、追加、または削除する必要があります。

重要: このサンプルファイルは、デプロイメントイメージに統合する前に、特定の環境に合わせて調整する必要があります。**ebx.xml** ファイルには UTF-8 エンコーディングが必要です。

```
<?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>

```

注

JNDI データソースを設定することで、EBX® はその JDBC 接続方法を使用するように指示されます。その結果、EBX® は 'EBX\DB\' で始まる環境変数を考慮しなくなります。

JNDIでのメール設定

EBX® 用の JNDI メールセッションを設定するには、Docker イメージをカスタマイズする必要があります。以下の Dockerfile のサンプルは、**ebx.xml** 設定を統合するために必要な手順を示しています。

```

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
```

ebx.xml ファイルは、Web アプリケーションのコンテキストを管理し、EBX® で使用される JNDI メールセッションの重要な設定情報を含んでいます。

重要な点: 以下にリストされている各プロパティは、メールマネージャーまたはクラウドプロバイダーに応じて、慎重に調整、追加、または削除する必要があります。

重要: このサンプルファイルは、デプロイメントイメージに統合する前に、特定の環境に合わせて調整する必要があります。**ebx.xml** ファイルには UTF-8 エンコーディングが必要です。

```
<?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>

```

注

JNDI メールセッションを設定することで、EBX® はそのメール接続方法を使用するように指示されます。その結果、EBX® は 'EBX_SMTP_ ' で始まる環境変数を考慮しなくなります。

メタデータ分類子データベースのドライバー追加

メタデータ分類子データベースのドライバーを追加するには、対応する Python モジュールをインストールする必要があります。Python モジュールは Python 仮想環境内にインストールする必要があります。以下の例は、ドライバーを追加するために Docker イメージをカスタマイズする方法を示しています。

PostgreSQL データベースの場合:

```
FROM ebx:6.2.3
```

```
RUN pipenv install psycopg2-binary
```

Oracle データベースの場合:

```
FROM ebx:6.2.3
```

```
RUN pipenv install oracledb
```

Microsoft SQL Server データベースの場合:

```
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
```

SQLAlchemy で使用できるデータベースドライバの詳細については、[SQLAlchemy ダイアレクト](#)を参照してください。

技術管理

- リポジトリ管理
- UI 管理
- UI - ワークフローランチャー
- ユーザーとロールのディレクトリ
- データモデルの管理
- データベースマッピング管理
- 外部サービス構成
- ワークフロー管理
- タスクスケジューラ
- バックグラウンドタスク
- 監査証跡
- コラボレーション機能 (チームアップ)
- AI アシスタント
- その他

リポジトリ管理

- 技術アーキテクチャ
 - 概要
 - データベースアクセスとユーザー権限のルール
 - リポジトリごとの単一 JVM
 - ホットスタンバイによるフェールオーバー
 - リポジトリのステータス情報とログ
- Auto-increments
- リポジトリ管理
 - インストールとアップグレード
 - データベース間の移行
 - リポジトリのバックアップ
 - アーカイブディレクトリ

- リポジトリ属性
- レコードの重複排除
- 監視管理
 - リレーショナルデータベースの監視とクリーンアップ
 - ファイルシステムの監視とクリーンアップ
- データスペース
 - データスペース/スナップショット
 - データスペースの権限
 - リポジトリの履歴

技術アーキテクチャ

概要

TIBCO EBX® 技術アーキテクチャの主な原則は次のとおりです。

- EBX® を実行する Java プロセス (JVM) は、単一の EBX® リポジトリに制限されます。このリポジトリは、サポートされているリレーショナルデータベースインスタンス に物理的に保存され、構成されたデータソース を介してアクセスされます。
- リポジトリは、特定の時点で複数の JVM で共有することはできません。ただし、フェールオーバーアーキテクチャを使用することはできます。これらの側面については、[リポジトリごとに 1 つの JVM および ホットスタンバイによるフェールオーバー](#) のセクションで詳しく説明しています。さらに、水平方向のスケーラビリティを実現するには、[分散データ配信 \(D3\)](#) 環境を展開するという方法もあります。
- 単一のリレーショナルデータベース インスタンスは、複数の EBX® リポジトリ (異なる JVM によって使用される) をサポートできます。その場合、プロパティ `ebx.persistence.table.prefix` を使用して、異なるテーブルプレフィックスを指定する必要があります。

以下も参照してください。:

- [EBX® リポジトリの構成](#)
- [サポートされるデータベース](#)

- EBX® リポジトリのデータソース

データベースアクセスとユーザー権限のルール

注意:

永続化されたマスターデータの整合性を保証するために、データベースへの直接 SQL 書き込みを実行することは固く禁じられています。

構成されたデータソースによって指定されたデータベースユーザーに、テーブル、インデックス、およびシーケンスに対する 'create/alter' 権限が必要です。これにより、リポジトリの自動インストールとアップグレードが可能になります。

以下も参照してください。:

- 履歴への SQL アクセス
- SQL を使用したレプリカテーブルへのアクセス
- EBX® リポジトリのデータソース

リポジトリごとに単一の JVM

リポジトリは複数の JVM で共有できません。このような状況が発生すると、予期しない動作が発生し、リポジトリ内のデータが破損する可能性があります。

EBX® は、この制限を強制するためにチェックを実行します。リポジトリが使用可能になる前に、リポジトリはまずリレーショナルデータベースの排他的所有権を取得する必要があります。リポジトリを起動した後、JVM はリポジトリの所有権を保持しているかどうかを定期的にチェックします。

これらのチェックは、リレーショナルデータベース内の技術テーブルに繰り返しタグを付けることで実行されます。アプリケーションサーバーのシャットダウンコマンドにより、この技術テーブルのタグが確実に削除されます。サーバーが予期せずシャットダウンした場合、タグがテーブルに

残ることがあります。このような状況が発生した場合、サーバーは再起動時にさらに数秒間待機し、テーブルが別のライブプロセスによって更新されていないことを確認する必要があります。

注意:

次の起動時に追加の待機時間を回避するには、アプリケーションサーバーを常に適切にシャットダウンすることをお勧めします。

ホットスタンバイによるフェールオーバー

上記の除外メカニズムは、アクティブ/パッシブクラスターで一度に 1 つのサーバーのみがアクティブになるフェールオーバーアーキテクチャと互換性があります。これを実現するには、メインサーバーでプロパティ `ebx.repository.ownership.mode=failovermain` を宣言する必要があります。メインサーバーは、単一サーバーの場合と同様に、リポジトリデータベースの所有権を主張します。

バックアップサーバーは起動できますが、リポジトリにアクセスできません。バックアップサーバーとして機能するには、プロパティ `ebx.repository.ownership.mode=failoverstandby` を宣言する必要があります。 `ebx.repository.directory` プロパティで定義される EBX® ルートディレクトリをアクティブ EBX® およびスタンバイ EBX® と共有することは推奨されません。これは、スタンバイ EBX® が制御を引き継ぐと、空のディレクトリで起動し、インデックスを再構築する必要があり、テーブルデータへの最初のアクセスが遅くなることを意味します。これを回避するには、EBX® リポジトリフォルダーを複製できます。Linux では、フェールオーバーノードでタイマーを実行して、数分ごとに `rsync` を実行し、EBX® リポジトリフォルダーをアクティブノードからフェールオーバーノードにコピーすることがテストで成功しています。インデックスの同時変更を防ぐために、フェールオーバーノードを起動する前に同期を停止することが重要です。

注意:

Kubernetes では、ホットスタンバイで EBX® フェールオーバーを使用せず、代わりに Kubernetes 自体に依存することをお勧めします。Kubernetes は、障害が発生した POD を、以前の POD のディスクを引き継ぐ新しい POD に自動的に置き換えることができます。サンプルの HELM チャートは、[GitHub](#) で入手できます。

バックアップサーバーが起動すると、接続ログに登録されます。そのステータスは、以下のセクション [リポジリのステータス情報とログ](#) で説明されているように、Java API または HTTP 要求を使用して取得できます。

バックアップサーバーをアクティブ化し、リポジリの排他的所有権をそのサーバーに転送するには、HTTP 要求または Java API を使用して特定の要求を発行する必要があります。

- HTTP を使用する場合、リクエストにはパラメーター `activationKeyFromStandbyMode` を含める必要があります、このパラメーターの値は EBX® メイン構成ファイルのエントリ `ebx.repository.ownership.activationkey` に宣言された値と同じにする必要があります。 [フェールオーバーの構成](#) を参照してください。

リクエスト URL の形式は次のようになります。

```
http[s]://<host>[:<port>]/ebx?activationKeyFromStandbyMode={value}
```

- Java API を使用して、メソッド `RepositoryStatus.wakeFromStandby` を呼び出します。

メインサーバーがまだ稼働していてデータベースにアクセスしている場合は、次のようになります。バックアップサーバーはデータベース内の所有権テーブルにマークを付け、メインサーバーのクリーンシャットダウンを要求します (ただし、実行中のトランザクションは終了できます)。メインサーバーが所有権を返した後にのみ、バックアップサーバーはリポジリの使用を開始できます。

リポジリのステータス情報とログ

リポジリへのすべての試行された Java プロセス接続のログは、領域の [\[履歴とログ\]](#) > で確認できます。

リポジリのステータスは、`RepositoryStatus` API のメソッドを使用して取得できます。

次のいずれかの値を持つパラメーター `repositoryInformationRequest` を含む HTTP リクエストを使用して、リポジリのステータス情報を取得することもできます。

`state`

所有権登録の観点から見たリポジリの状態。

- `D`: Java プロセスが停止しています。

	• O : Java プロセスはデータベースの排他的所有権を持ちます。 • S : Java プロセスはフェールオーバースタンバイモードで開始されましたが、リポジリとの対話はまだ許可されていません。 • N : Java プロセスがデータベースの所有権を取得しようとしたが、別のプロセスが所有権を保持しているため失敗しました。
heart_beat_count	リポジリがデータベースに関連付けられてから接続した回数。
info	リポジリの登録ステータスに関するエンドユーザー向けの詳細情報。この情報の形式は、将来、明示的な警告なしに変更される可能性があります。

Auto-increments

EBX® ユーザーインターフェイスの 領域では、いくつかの技術テーブルにアクセスできます。これらのテーブルは内部使用のみを目的としており、古いデータや誤ったデータを削除する場合を除き、その内容を手動で編集しないでください。これらの技術テーブルには次のものがあります。

Auto-increments	リポジリ内のすべての自動インクリメントフィールドを一覧表示します。
-----------------	-----------------------------------

リポジリ管理

インストールとアップグレード

自動インストールとアップグレード

[データベースアクセスとユーザー権限のルール](#) に準拠することで、リポジリのインストールまたはアップグレードが自動的に実行されます。

データベース間の移行

EBX® は、リポジリの完全なコンテンツを別のデータベースにエクスポートする方法を提供します。エクスポートには、すべてのデータスペース、構成データセット、およびマップされたテーブルが含まれます。この移行を実行するには、次のガイドラインに従う必要があります。

- ソースリポジリを **シャットダウンする必要があります**。EBX® サーバープロセスがアクセスしてはなりません。この要件に厳密に従わないと、ターゲットリポジリが破損する可能性があります。
- 新しい EBX® サーバープロセスは、ターゲットリポジリで起動する必要があります。ターゲットリポジリは空にする必要があります。従来の Java システム プロパティ `-Debx.properties` に加えて、このプロセスでは、ソースリポジリを指定する EBX® プロパティファイルの場所である `ebx.migration.source.properties` も指定する必要があります (ターゲットとソースの間に異なるテーブルプレフィックスを指定できます)。
- その後、移行プロセスが自動的に実行されます。ただし、このプロセスはトランザクションではないことに注意してください。途中で失敗した場合は、最初からやり直す前に、ターゲットデータベースに作成されたオブジェクトを削除する必要があります。
- 移行が完了すると、例外がスローされ、ターゲットリポジリにアクセスする EBX® サーバープロセスを強制的に再起動します。

制限事項

- マップされたテーブル (履歴、レプリケーション) を表すデータベースオブジェクトの名前は、データベースエンジンの制限 (最大長、予約語など) に準拠するために、ターゲットデータベースに移行するときに変更する必要がある場合があります。このような変更は、移行プロセス中にログに記録されます。
- その結果、データモデル内の複製されたテーブルに指定された名前は、データベース内の適応された名前と一致なくなります。このデータモデルの最初の再コンパイルにより、この不一致が強制的に修正されます。
- 数値型の表現方法が異なるため、ターゲットデータベースエンジンの精度がソースより低い場合、`xs:decimal` 型の値は丸められる可能性があります。たとえば、Oracle の `100000000.1234567890123456789` の値は、SQL Server では `100000000.123456789012345679` に丸められます。

リポジリのバックアップ

EBX® リポジリのグローバルバックアップは、基盤となる RDBMS に委任する必要があります。データベース管理者は、基盤となるデータベースの標準バックアップ手順を使用する必要があります。

アーカイブディレクトリ

アーカイブは、`ebx.repository.directory` 内の `archives` というサブディレクトリに保存されます ([configuration](#) を参照)。このディレクトリは、EBX® からの最初のエクスポート時に自動的に作成されます。

注意:

セキュリティのベストプラクティスで指定されているように、このディレクトリへのアクセスは慎重に保護する必要があります。また、このディレクトリを手動で作成する場合は、EBX® プロセスに読み取り/書き込みアクセス権があることを確認してください。さらに、EBX® はこのディレクトリを管理しないため、管理者がこのディレクトリをクリーンアップする責任があります。

注意:

2 つの EBX® 環境間でのファイル転送は、FTP などのツールや、ネットワーク共有による単純なファイルコピーを使用して実行する必要があります。

リポジトリ属性

リポジトリには次の属性があります。

repositoryId	企業の範囲内でリポジトリを一意に識別します。これは 48 ビット (6 バイト) で、通常は 12 桁の 16 進数で表されます。この情報は、リポジトリで作成されたエンティティ、および履歴テーブルまたは XML 監査証跡に記録されたトランザクションの UUID (Universally Unique Identifiers) を生成するために使用されます。この識別子は、RFC 4122 で指定されている「UUID ノード」部分として機能します。
--------------	--

repository label	リポジリの目的とコンテキストを識別するユーザーフレンドリなラベルを提供します。 例：「Production environment」。
store format	構造の現在のバージョンを含む、基礎となる永続性システムを識別します。

レコードの重複排除

同じテーブル内のレコードが重複すると、インデックスの問題が発生する可能性があります。EBX® には、この問題を解決するためのビルトインサービスが用意されています。サービスを実行する手順については、以下を参照してください。

注意:

サービスの実行中は、UI、REST、または SOAP リクエストを通じて EBX® リポジリにアクセスすることはできません。完了すると、サービスはリポジリを自動的にシャットダウンします。

重複レコードを削除するには、次の手順を実行します。

1. `ebx.properties` 構成ファイルで、`ebx.persistence.boot.checkRecordDuplicates` プロパティを `true` に設定します。
2. リポジリを起動し、サービスがリポジリを自動的にシャットダウンするまで待機します。
3. `ebx.persistence.boot.checkRecordDuplicates` プロパティを `false` に設定します。
4. リポジリを起動します。

監視管理

リレーショナルデータベースの監視とクリーンアップ

EBX® の実行中にいくつかのエンティティが蓄積されます。

注意:

これらのエンティティを監視してクリーンアップするのは **管理者の責任** です。

データベースの監視と整理

リポジリの永続データソースは、RDBMS 監視を通じて監視する必要があります。

EBX® の次のテーブルにより、データベース内でのデータの永続化が可能になります。

- テーブル `{ebx.persistence.table.prefix}G_DSP`。このテーブル内の各レコードはデータスペースまたはスナップショットを表します (プロパティ `ebx.persistence.table.prefix` が設定されていない場合、名前は `EBXG_DSP` になります)。
- テーブル `${ebx.persistence.table.prefix}G_DST` にはデータセット参照が保持されます。
- テーブル `{ebx.persistence.table.prefix}G_BLK`。EBX® テーブルのデータは、最大 256 個の EBX® レコードのサイズのブロックに分割されます。
- テーブル `{ebx.persistence.table.prefix}G_DTR`、`{ebx.persistence.table.prefix}G_TRV`、および `{ebx.persistence.table.prefix}G_SHR`。特定のデータスペースまたはスナップショット内の特定の EBX® テーブルに属するブロックを定義します。
- テーブル `{ebx.persistence.table.prefix}G_TLR` および `{ebx.persistence.table.prefix}G_SCH` には、スキーマとそれらが定義するテーブルに関する情報が保持されます。

データベース統計

EBX® によって実行されるリクエストのパフォーマンスには、データベースがテーブルに関する最新の統計を計算していることが必要です。データベースエンジンは定期的に統計の更新をスケジュールするため、これは通常問題にはなりません。ただし、テーブルが短期間で大幅に変更された場合 (たとえば、多数のレコードを作成するインポートなど) は、統計を明示的に更新する必要がある可能性があります。

履歴テーブル：UI への影響

履歴テーブルの場合、一部の UI コンポーネントは、ユーザーが不本意にコストのかかるリクエストを実行するのを防ぐために、統計を使用して動作を調整します。

たとえば、テーブルに大量のレコードが含まれている場合、コンボボックスはユーザー入力を自動的に検索しません。この動作は、データベースの統計が最新でない場合にも発生する可能性があります。これは、実際にはそうでなくても、テーブルに大量のレコードが含まれていると見なされる可能性があるためです。

データスペース、スナップショット、履歴のクリーンアップ

リポジトリからデータスペース、スナップショット、履歴を完全にクリーンアップするには、いくつかの段階があります。

1. 未使用のデータスペースとスナップショットを閉じて、キャッシュのサイズを最小限に抑えます。
2. データスペース、スナップショット、履歴を削除します。
3. 削除されたデータスペース、スナップショット、履歴に関連付けられている残りのエンティティをリポジトリから削除します。

未使用のデータスペースとスナップショットを閉じる

キャッシュとリポジトリを適切なサイズに保つために、不要になったデータスペースとスナップショットを閉じることをお勧めします。これは、次の方法で実行できます。

- ユーザーインターフェイスの「データスペース」領域から。
- 領域の の下にある テーブルから、ワークスペースの メニューを使用します。アクションは、テーブルのフィルターされたビューで使用できます。
- Java API 経由で、メソッド `Repository.closeHome` を使用します。
- データサービスの および 操作を使用します。詳細については、[データスペースまたはスナップショットを閉じる](#) を参照してください。

データスペースとスナップショットが閉じられると、リポジトリからデータを安全に削除できます。

注意:

閉じられたデータスペースとスナップショットは、「データスペース」の下の 領域で再度開くことができます。

データスペース、スナップショット、履歴の削除

データスペース、関連する履歴、およびスナップショットは、リポジトリから完全に削除できます。ただし、データスペースを削除しても、必ずしも履歴が削除されるわけではありません。この 2 つの操作は独立しており、異なるタイミングで実行できます。

注意:

データスペース、スナップショット、またはそれらに関連付けられた履歴の削除は再帰的です。Delete 操作は、選択したデータスペースのすべての子孫に対して実行されます。

データスペースまたはスナップショットの削除後、古いデータのリポジトリ全体のページが実行されるまで、一部のエンティティが残ります。

特に、データスペースの完全な履歴は、リポジトリ全体のページが実行されるまで表示されたままになります。データと履歴を完全に削除するには、削除とリポジトリ全体のページの両方の手順を完了する必要があります。このプロセスは、パフォーマンスの問題のために 2 つのステップに分割されています。リポジトリの全体的なクリーンアップには時間がかかる可能性があるため、これにより、サーバーのオフピーク時にページの実行を開始できます。

データスペースの履歴を削除するプロセスでは、削除が送信されるまで、またはユーザーが指定した日付までに記録されたすべての履歴トランザクションが考慮されます。その後の履歴化された操作は、ページ操作の実行時には含まれません。新しいトランザクションを削除するには、データスペースの履歴を再度削除する必要があります。

注意:

将来の削除日を設定することはできません。指定された日付は無視され、代わりに現在の日付が使用されます。

データスペース、スナップショット、履歴の削除は、さまざまな方法で実行できます。

- 領域の下にある テーブルから、ワークスペースの メニューボタンを使用します。アクションは、テーブルのフィルターされたビューで使用できます。
- Java API、具体的にはメソッド `Repository.deleteHome` と `RepositoryPurge.markHomeForHistoryPurge` を使用します。
- データ サービスの「データスペースを閉じる」操作の終了時に、パラメーター `deleteDataOnClose` と `deleteHistoryOnClose` を使用するか、「データスペースのマージ」操作の終了時に、パラメーター `deleteDataOnMerge` と `deleteHistoryOnMerge` を使用します。

データスペース、スナップショット、または履歴の削除後に残りのエンティティを削除

アイテムが削除されたら、その時点までに実行された すべての削除から残ったデータをクリーンアップするためにパージを実行できます。パージは次の方法で開始できます。

- ユーザーインターフェイスから、ナビゲーションペインの 領域で > を選択します。
- Java API、具体的にはメソッド `RepositoryPurge.purgeAll` を使用します。
- タスクスケジューラの使用。詳細については、[タスクスケジューラ](#) を参照してください。

パージプロセスは、ディレクトリ `${ebx.repository.directory}/db.purge/` に記録されます。

他のリポジリエンティティのクリーンアップ

以下のエンティティを監視し、定期的にクリーンアップするのは *管理者の責任* です。

削除

削除を実行して、すべての削除からの残りのデータ、つまり、その時点までに実行されたデータスペース、スナップショット、および履歴を削除できます。これには、廃止された永続的な検証レポート用に作成されたデータスペースとスナップショットが含まれます。削除は、ナビゲーションペインの エリアで > を選択することで開始できます。

タスクスケジューラ実行レポート

タスク スケジューラ実行レポートは、領域の セクションにある テーブルに保存されます。スケジュールされたタスクは、実行されるたびにこのテーブルに継続的に追加されます。実行が正常に終了した場合でも、レコードは自動的に削除されません。したがって、古いレコードを定期的に削除することをお勧めします。

ユーザーインタラクション

ユーザーインタラクションは、アプリケーションがサービス実行を開始してその結果を取得するための信頼できる手段として、EBX® コンポーネントによって使用されます。これらは、*ebx-interactions* 管理セクションに保存されます。ユーザーインタラクションテーブルを定期的に監視し、必要に応じてクリーンアップすることをお勧めします。

ワークフロー履歴

ワークフローイベントは、エリアの セクションにあるワークフロー履歴テーブルに保持されます。データワークフローは、実行されるたびにこのテーブルに追加されます。実行が正常に終了しても、レコードは自動的に削除されません。したがって、古いレコードを定期的に削除することをお勧めします。

履歴を消去する手順は次のとおりです。

- プロセス実行が削除されていることを確認します (ワークフローの 領域で、ナビゲーションペインの > または > を選択することで実行できます)。
- 履歴内のメイン プロセスを消去します (ワークフロー履歴の 領域で、ナビゲーションペインの > または > を選択することで実行できます)。
- 「標準 EBX® パージ」を使用してワークフロー履歴に残っているエンティティをパージします。

以下も参照してください。:

- [標準 EBX® パージ](#)

ファイルシステムの監視とクリーンアップ

注意:

EBX® の正しい動作を保証するために、EBX® は Lucene インデックスを除いてクリーンアップを実行しないため、管理者は次のディレクトリのディスク使用量とディスクの可用性を監視する必要があります。

- **Lucene インデックス:** `${ebx.repository.directory}/indexes-(...)/`

Lucene インデックス: インデックスは大量のディスク領域を必要とする場合があります。インデックスは EBX® が正しく機能するために不可欠です。通常の使用では、インデックスを直接削除したり変更したりしないでください。ただし、次のような場合には、これらのインデックスを削除する必要がある場合があります。

- 。リポジトリを最初から再作成する場合、ディレクトリ `${ebx.repository.directory}/` は保持されますが、データの一貫性を確保するために、インデックスのルートディレクトリを削除する必要があります。
- 。より一般的には、インデックスがリポジトリデータと一致しなくなった場合 (これはバグのまれなケースで発生する可能性があります)。

削除後、インデックスの内容はリポジトリの内容から派生してテーブルごとに遅延再計算されます。削除はインデックスのルートフォルダーで実行する必要があります。下位レベルで1つのディレクトリを削除すると、インデックスのグローバル構造に一貫性がなくなります。ただし、この操作にはコストがかかるため、通常は避ける必要があります。

- XML 監査証跡: `${ebx.repository.directory}/History/`
- アーカイブ: `${ebx.repository.directory}/archives/`
- ログ: `ebx.logs.directory`
- 一時ディレクトリ: `ebx.temp.directory`

注意:

XML 監査証跡 の場合、完全な更新詳細を有効にした状態で大規模なトランザクションを実行すると (デフォルト設定に反して)、必要なディスク容量が増加する可能性があります。

注意:

データ サービスの `getChanges` 操作でのページ分割には、**一時ディレクトリ** の永続ストアが使用されます。大きな変更には、大量のディスク領域が必要になる場合があります。

以下も参照してください。:

- XML 監査証跡
- EBX® リポジトリの調整

データスペース

一部のデータスペース管理タスクは、EBX® の 領域で を選択して実行できます。

データスペース/スナップショット

このテーブルには、開いているか閉じているかに関係なく、リポジリ内の既存のデータスペースとスナップショットがすべてリストされます。このテーブルに含まれるデータスペースの情報を表示および変更できます。

以下も参照してください。:

- [データスペース情報](#)

このセクションからは、開いているデータスペースを閉じたり、以前に閉じたデータスペースを再度開いたり、開いているデータスペースや閉じているデータスペース、関連する履歴、スナップショットを削除したり消去したりすることもできます。

以下も参照してください。:

- [データスペース、スナップショット、履歴のクリーンアップ](#)

データスペースの権限

このテーブルには、リポジリ内のすべてのデータスペースに定義されている既存の権限ルールがすべてリストされます。権限ルールを表示し、その情報を変更できます。

以下も参照してください。:

- [データスペースの権限](#)

リポジリの履歴

「削除されたデータスペース/スナップショット」テーブルには、リポジリからすでに削除されたすべてのデータスペースがリストされます。

このセクションから、削除されたデータスペースの履歴を削除することもできます。

UI 管理

TIBCO EBX® には、利用可能なすべての機能を含む詳細パースペクティブと呼ばれる完全なユーザーインターフェイスが付属しています。インターフェイスは完全にカスタマイズ可能 (カスタムロゴ、色、フィールドサイズ、デフォルト値など) であり、ビルトインの管理者が利用できます。インターフェイスは完全にカスタマイズ可能 (カスタムロゴ、色、フィールドサイズ、デフォルト値など) であり、ビルトインの管理者が利用できます。

グローバル許可を使用して、エンドユーザーエクスペリエンスを簡素化するために、詳細パースペクティブへのアクセスを制限して、機能カテゴリへのアクセスを許可または制限することができます。管理者は、エンドユーザー向けに推奨されるパースペクティブと呼ばれる簡略化されたパースペクティブを作成できます。これには、日常のタスクに必要な機能とメニューのみが含まれます。

- グローバル許可
- 詳細パースペクティブ
 - パースペクティブの作成
 - ユーザーインターフェイス
 - 子パースペクティブメニュー
- 推奨されるパースペクティブ
 - 推奨されるパースペクティブの管理
 - パースペクティブの決定
- カスタムビュー
 - ビュー
 - ビューの権限
- ユーザーセッション管理
- アクセシビリティモード
 - サポートされているスクリーンリーダー

グローバル許可

グローバル許可ルールは、EBX® で作成できます。

「表示エリア」プロパティを使用すると、ユーザーインターフェイスの領域へのアクセスを制限できます。アクセスルールを定義するには、領域で を選択します。

プロファイル	ルールが適用されるプロファイルを示します。
制限ポリシー	ここで定義された権限が、他のプロファイルに定義された権限を制限するかどうかを示します。詳細については、 制限ポリシーの概念 を参照してください。
データスペース	データスペース領域の権限を定義します。
データモデル	データモデル領域の権限を定義します。
ワークフローモデル	ワークフローモデル領域の権限を定義します。
データワークフロー	データワークフロー領域の権限を定義します。
データサービス	データサービス領域の権限を定義します。 独立して、次のことも可能です。 • REST ビルトインコネクタ HTTP(S) のアクセス権限を定義します。この設定は、REST Toolkit アプリケーションには影響しません。 • REST OpenAPI サービスのアクセス権限を定義します。 • SOAP コネクタ HTTP(S) および JMS のアクセス権限を定義します。 • WSDL コネクタ HTTP(S) のアクセス権限を定義します。
管理	管理領域の権限を定義します。

注意:

権限は、管理者とデータスペースまたはデータセットの所有者が定義できます。

詳細パースペクティブ

詳細パースペクティブとそのパラメーター化は独特です。これは、[新しいパースペクティブ](#)が継承する親パースペクティブです。

エンドユーザーにカスタマイズされた簡素化されたメニューを提供するために、そのメインパースペクティブから子パースペクティブを作成できます。データセットの継承が利用できるため、これらの単純化されたパースペクティブは、詳細パースペクティブ (ルートデータセット) からパラメーターを受け取ります。これらのパラメーターは、新しく作成された簡略化されたパースペクティブでオーバーライドできます。簡略化されたパースペクティブは、既存の簡略化されたパースペクティブの下に作成できるため、親のパラメーターから継承されます。

以下も参照してください。:

- [継承](#)

詳細パースペクティブは、デフォルトですべてのエンドユーザーが利用できますが、アクセスを制限することができます。

注意: 管理者は、非アクティブ化されている場合でも、いつでも詳細パースペクティブにアクセスできます。

ユーザーがログインするときにデフォルトで適用されるパースペクティブを構成できます。この「デフォルトのパースペクティブ」は、管理者が定義する「推奨されるパースペクティブ」とユーザーが定義する「お気に入りのパースペクティブ」の2つの基準に基づいています。

以下も参照してください。:

- [推奨されるパースペクティブ](#)
- [お気に入りのパースペクティブ](#)

パースペクティブの作成

パースペクティブを作成するには、ドロップダウンメニューを開き、+ 記号をクリックして子データセットを作成します。

以下も参照してください。:

- 継承する子データセットの作成

ユーザーインターフェイス

オプションは、セクションの Web インターフェイスを構成するための管理領域で使用できます。

注意:

URL ポリシー (非推奨) を構成する場合は、注意が必要です。Web インターフェイスの構成が無効な場合、EBX® が使用できなくなる可能性があります。このような場合は、**EBX® メイン構成ファイル** で `frontEnd.rescueMode.enable=true` を設定して「レスキューモード」を使用し、ブラウザでビルトインの管理者ユーザーとして `http://.../ebx/?onwbpID=iebx-manager-rescue` にアクセスしてください。

セッション構成

これらのパラメーターは、ユーザーセッションオプションを構成します。

ユーザーセッションのデフォルトロケール	デフォルトのセッションロケール
セッションタイムアウト (秒単位)	セッションが非アクティブであると見なされて終了するまでの、ユーザーの非アクティブの最大期間。負の値は、セッションがタイムアウトしないことを示します。

インターフェイス構成

エントリポリシー (非推奨)

注意:

この画面は廃止されました。これらのオプションを `ebx.properties` ファイルで構成することをお勧めします。

以下も参照してください。:

- [セキュリティ構成](#)

アプリケーションにアクセスするための URL について説明します。

ログイン URL

ユーザーが認証されていない場合、セッションはこの URL に転送されます。

エントリポリシーは、EBX® ログインページを定義し、デフォルトのログインページを置き換えます。

- 特定のユーザー `ディレクトリ` を使用して定義された可能性のある認証 URL を置き換えます。
- これは、ユーザーインターフェイスでパーマリンクを構築するために使用されます。
- 完全 URL の場合、つまり `http://` または `https://` で始まる場合、ワークフローのメール構成の URL が置き換えられます。

URL ポリシー (非推奨)

URL とプロキシポリシーについて説明します。動的 (サーブレット) URL と静的 (リソース) URL の両方を構成できます。

注意:

この設定方法は非推奨です。>>>> に移動して、既存の設定を元に戻します。URL コンピューティング設定を更新するには、`ebx.properties` ファイルを編集します。詳細については、[URL コンピューティング](#) を参照してください。

HTTP サーブレットポリシー	サーブレット HTTP リクエストのヘッダーコンテンツ - フィールドが設定されていない場合、環境構成のデフォルト値が使用されます。 - デフォルト値が設定されていない場合は、最初のリクエストの値が使用されます。
HTTPS サーブレットポリシー	サーブレット HTTPS リクエストのヘッダーコンテンツ - フィールドが設定されていない場合、デフォルト値が選択されます (環境構成の場合)。 - デフォルト値が設定されていない場合は、最初のリクエストの値が使用されます。
HTTP 外部リソースポリシー	HTTP の外部リソース URL のヘッダーコンテンツ - フィールドが設定されていない場合、環境構成のデフォルト値が使用されます。 - デフォルト値が設定されていない場合は、最初のリクエストの値が使用されます。
HTTPS 外部リソースポリシー	HTTPS での外部リソースURLのヘッダーコンテンツ

- フィールドが設定されていない場合、環境構成のデフォルト値が使用されます。
- デフォルト値が設定されていない場合は、最初のリクエストの値が使用されます。

終了ポリシー (非推奨)

注意:

この画面は廃止されました。これらのオプションを `ebx.properties` ファイルで構成することをお勧めします。

以下も参照してください。:

- [セキュリティ構成](#)

アプリケーションがどのように終了するかを説明します。

通常のリダイレクト	セッションを正常に終了するときに使用されるリダイレクト URL を指定します。
エラーリダイレクト	エラーのためにセッションを終了するときに使用されるリダイレクト URL を指定します。 この機能は非推奨になり、EBX® では無視される可能性があります。
リダイレクト制限	許可されたドメインのリストと、各ドメインに HTTPS が必須かどうかを指定します。

グラフィカルインターフェースの構成

有効化と許可されたプロファイル

ラジオボタンを使用すると、パースペクティブをアクティブ化または非アクティブ化できます。非アクティブ化すると、パースペクティブは管理者のみが使用できるようになります。

「許可されたプロファイル」機能は、特定のプロファイルへのパースペクティブへのアクセスを提供するために使用されます。番号付きリストの下にある + アイコンをクリックすると、承認済みプロファイルのリストに複数のプロファイルを追加できます。

使用可能なパースペクティブプロパティは次のとおりです。

有効化済み	パースペクティブが許可されたユーザーに表示されるかどうかを示します。
許可されたプロファイル	パースペクティブの許可されたユーザープロファイルのリスト。
許可されたデバイス	パースペクティブに許可されたデバイスのリスト。 指定しない場合、「EBX® Web アプリケーション」のみがこのパースペクティブを表示できます。
デフォルト選択	デフォルトで選択されているメニュー項目。 このプロパティは、詳細パースペクティブでは使用できません。

アプリケーションのロック

EBX® の可用性ステータス

利用可能ステータス	このアプリケーションは、メンテナンス中にユーザーに対して閉じることができます (ただし、管理者に対しては開いたままにします)。すぐに有効になります。
-----------	--

利用不可メッセージ	アクセスが管理者プロファイルに制限されている場合にユーザーに表示されるメッセージ。
-----------	---

セキュリティポリシー

EBX® アクセスセキュリティポリシー。これらのパラメーターは、新しい HTTP セッションにのみ適用されます。

IP アクセス制限	指定された IP アドレスへのアクセスを制限します (以下の IP パターンを参照)。
IP 制限パターン	EBX® へのアクセスを許可する IP アドレスを正規表現で表したもの。たとえば、 <code>((127.0.0***) (192.168.****))</code> は、ローカルマシンとネットワーク IP 範囲 <code>192.168.*.*</code> にアクセスを許可します。
一意のセッション制御	EBX® がユーザーセッションの一意性を制御する必要があるかどうかを指定します。 「はい」に設定されている場合、ユーザーがブラウザを閉じる前にログアウトしないと、前のセッションがタイムアウトするまで、そのユーザーは再度ログインできません。

人間工学とレイアウト

EBX® 人間工学パラメーター

表示する最大テーブル列	ネットワークとブラウザのパフォーマンスに応じて、テーブルに表示する列の最大数を調整します。このプロパティは、ビューがテーブルに適用される場合は使用されません。
テーブル列の最大自動幅	テーブルの初期化中にテーブルの列が自動サイズ設定できる最大幅を定義します。これ

	は、URL などの非常に長い値で発生する可能性がある列の幅が広すぎるのを防ぐためです。ユーザーは、この値を超えて列のサイズを手動で変更できます。
階層の最大拡張エレメント	アクション「すべて展開」を使用するときに階層内で展開できるエレメントの最大数を定義します。「0」以下の値は、このパラメーターを無効にします。
デフォルトテーブルフィルター	表形式ビューのフィルターリストに表示するデフォルトのテーブルフィルターを定義します。変更した場合、新しい値を有効にするには、ユーザーはログアウトしてログインする必要があります。
自動的にメッセージボックスを表示	メッセージポップアップを表示するためのメッセージ重大度のしきい値を定義します。
フォーム：ラベルの幅	フォーム内のラベルの幅。
フォーム：入力の幅	フォームのフォーム入力フィールドの幅。
フォーム：テキストエリアの高さ	フォームのテキスト入力フィールドの高さ。
フォーム：集約リスト	リストに新しいインスタンスを作成するために使用できる、生成される非表示の候補行の数。
フォーム：HTML エディターの最小幅	フォーム内の HTML エディターの最小幅。固定幅のエディターにするには、最大幅と同じ値を設定します。
フォーム：HTML エディターの最大幅	フォーム内の HTML エディターの最大幅。
フォーム：HTML エディターの最小高さ	フォーム内の HTML エディターの最小高さ。固定高さのエディターにするには、最大高さと同じ値を設定します。

フォーム：HTML エディターの最大高さ	フォーム内の HTML エディターの最大高さ。フォームでは、スクロールの問題を避けるため、この値はワークスペースの高さを超えません。
検索可能リスト選択ページサイズ	検索可能なリスト選択の各リクエストでダウンロードされる最大行数 (外部キー、列挙型などの選択に使用されます)。
レコードフォーム：ノードのレンダリングモード	非ターミナルノードをレコード形式で表示する方法を指定します。このプロパティを変更した場合、新しい値を有効にするには、ユーザーはログアウトしてログインする必要があります。
レコードフォーム：作成モードで、選択および関連ノードを表示	有効にすると、選択ノードと関連付けノードがレコード作成フォームに表示されます。
表示密度	すべてのユーザーのデフォルトの表示密度モードを定義します。ユーザーが密度をまだ選択していない場合は、この値が適用されます。逆に、ユーザーがすでに密度を選択している場合は、その選択が優先されます。
ヘッダーに表示されるアバター	このプロパティは、ヘッダー内のアバターの表示モードを定義します。たとえば、このプロパティを更新することで、ヘッダーでのアバターの使用を有効または無効にすることができます。値が定義されていない場合、デフォルト値は「アバターのみ」です。相対パスの場合は、プレフィックスとして「../」を付けて、アプリケーションのルート URL に戻ります。
履歴に表示されるアバター	このプロパティは、履歴内のアバターの表示モードを定義します。たとえば、このプロパティを更新することで、履歴内のアバターの使用を有効または無効にすることができます。

	す。値が定義されていない場合、デフォルト値は「アバターののみ」です。相対パスの場合は、プレフィックスとして「../」を付けて、アプリケーションのルート URL に戻ります。
ワークフローに表示されるアバター	このプロパティは、ワークフローでのアバターの表示モードを定義します。たとえば、このプロパティを更新することで、ワークフローでのアバターの使用を有効または無効にすることができます。値が定義されていない場合、デフォルト値は「アバターののみ」です。相対パスの場合は、プレフィックスとして「../」を付けて、アプリケーションのルート URL に戻ります。

デフォルトのオプション値

ユーザーインターフェイスのオプションのデフォルト値を定義します。

インポート/エクスポート

CSV ファイルエンコード	CSV ファイルのインポートとエクスポートに使用するデフォルトの文字エンコードを指定します。
CSV：フィールド区切り文字	CSV ファイルのインポートとエクスポートに使用するデフォルトの区切り文字を指定します。
CSV：リスト区切り文字	CSV ファイルのインポートとエクスポートに使用するデフォルトのリスト区切り文字を指定します。
インポートモード	デフォルトのインポートモードを指定します。
欠損する XML 値を「null」とする	「はい」の場合、既存のレコードを更新するときに、インポートされたファイルに

ノードがないか空の場合、値は「null」と見なされます。「いいえ」の場合、値は変更されません。

色とテーマ

EBX® の色とテーマをカスタマイズします。

Web サイトアイコン URL (ファビコン)	カスタムファビコンを設定します。
ロゴ URL (SVG)	互換性のあるブラウザに使用される SVG 画像を指定します。指定されている場合、PNG 画像を使用するには、このフィールドを空白のままにします。ブラウザが SVG と互換性がない場合、ユーザーインターフェイスは指定された PNG 画像を使用しようとします。PNG 画像が指定されていない場合は、GIF/JPG 画像が使用されます。ロゴの最大高さは 40px の必要があります。高さが 40px を超える場合、40px の高さに拡大縮小されるのではなく、トリミングされます。ロゴの幅によって、ヘッダー内のボタンの位置が決まります。相対パスの場合は、プレフィックスとして「../」を付けて、アプリケーションのルート URL に戻ります。
ロゴ URL (PNG)	互換性のあるブラウザで使用される PNG 画像を指定します。GIF/JPG 画像を使用するには、このフィールドと SVG 画像の URL フィールドの両方を空白のままにします。ブラウザが PNG 互換でない場合、ユーザーインターフェイスは GIF/JPG 画像を使用します。ロゴの最大高さは 40px の必要があります。高さが 40px を超える場合、40px の高さに拡大縮小されるのではなく、トリミングされます。ロゴの幅によって、ヘッダー内のボタンの位置が決まります。相対パスの場合は、プレフィックスとして「../」を付け

	て、アプリケーションのルート URL に戻ります。
ロゴ URL (GIF/JPG)	PNG も SVG も定義されていない場合に使用する GIF/JPG 画像を指定します。推奨される形式は GIF と JPG です。ロゴの最大高さは 40px の必要があります。高さが 40px を超える場合、40px の高さに拡大縮小されるのではなく、トリミングされます。ロゴの幅によって、ヘッダー内のボタンの位置が決まります。相対パスの場合は、プレフィックスとして「../」を付けて、アプリケーションのルート URL に戻ります。
メイン UI	選択とハイライトに使用されるメインユーザーインターフェイスのテーマの色。
ヘッダー	ユーザーインターフェイスヘッダーの背景色。デフォルトでは、メインカラーと同じ値に設定されています。
ワークフローバッジ	新しいワークフロータスクカウンターの背景色とテキスト/アウトラインの色。
主ボタン	デフォルトで選択されているボタンの色。デフォルトでは、メインカラーと同じ値に設定されています。
リンクスタイルボタンのテキスト	リンクスタイルを持つ一部のボタンのテキストの色 (テキストは暗くも明るくもありませんが、色が付いています)。デフォルトでは、メインカラーと同じ値に設定されています。
選択したタブの境界	選択したタブの境界線の色。デフォルトでは、メインカラーと同じ値に設定されています。

テーブル履歴ビュー：テクニカルデータ	テーブル履歴ビューの技術データセルの背景色。
テーブル履歴ビュー：作成	テーブル履歴ビューの状態が「作成」のセルの背景色。
テーブル履歴ビュー：削除	テーブル履歴ビューの状態が「削除」のセルの背景色。
テーブル履歴ビュー：更新	テーブル履歴ビューの状態が「更新」のセルの背景色。

子パースペクティブメニュー

子パースペクティブは無制限に作成できます。子パースペクティブは、「詳細パースペクティブ」のパラメーターから継承します。これらのパラメーターの一部は、以下で詳しく説明するようにオーバーライドできます。

有効化と許可されたプロファイル

詳細については、[詳細パースペクティブのアクティベーションと許可されたプロファイル](#)を参照してください。

注意:

このパースペクティブに設定された特定のパラメーターは、「詳細パースペクティブ」構成で設定されたデフォルトのパラメーターをオーバーライドします。

パースペクティブメニュー

このビューには、パースペクティブメニューが表示されます。階層的なテーブルビューです。

このビューから、ユーザーはメニュー項目レコードを作成、削除、または並べ替えることができます。

以下も参照してください。:

- 階層テーブルビュー

セクションメニュー項目	これはトップレベルのメニュー項目です。他のメニュー項目が含まれています。
メニューグループ	これは他のメニュー項目のコンテナです。
アクションメニュー項目	このメニュー項目は、ワークスペース領域にユーザーサービスを表示します。

メニュー項目のプロパティ

メニューでレコードを作成する場合、使用可能なパースペクティブプロパティは次のとおりです。

タイプ	メニュー項目の種類。 以下も参照してください。: メニュー項目の種類
親	メニュー項目の親。 このプロパティは、セクションメニュー項目では使用できません。
ラベル	メニュー項目のラベル。 ラベルは、アクションメニュー項目ではオプションです。指定しない場合、メニュー項目が表示されたときにラベルが EBX® によって動的に生成されます。
許可されたデバイス	このアイテムの許可されたデバイスのリスト。

	指定しない場合、すべてのデバイスでこのメニュー項目を表示できます。現在サポートされているデバイスは、「EBX® Web アプリケーション」と「EBX® GO」の 2 つだけです。
アイコン	メニュー項目のアイコン。 アイコンは、「標準」(EBX® が提供) または URL で指定された画像のいずれかで、任意の Web サーバーでホストできます。 アイコンのサイズは 16x16 ピクセルにする必要があります。 このプロパティは、セクションメニュー項目では使用できません。
先頭区切り	メニュー項目セクションに上部区切り記号があることを示します。 このプロパティは、セクションメニュー項目でのみ使用できます。
アクション	ユーザーがメニュー項目をクリックしたときに実行するユーザーサービス。 以下も参照してください。: • ユーザーインターフェイスサービス エンドユーザーがパースペクティブを表示することは許可されているが、ユーザーサービスを実行することは許可されていない場合、ユーザーがメニュー項目をクリックすると、「アクセスが拒否されました」というメッセージが表示されます。 このプロパティは、アクションメニュー項目でのみ使用できます。

閉じるときの選択

サービス終了時に選択されるメニュー項目。

ビルトインサービスは、ユーザーが [閉じる] ボタンをクリックすると、このプロパティを使用します。

このプロパティは、アクションメニュー項目でのみ使用できます。

人間工学とレイアウト

詳細については、[詳細パースペクティブの人間工学とレイアウト](#)を参照してください。

注意:

このパースペクティブに設定された特定のパラメーターは、「詳細パースペクティブ」構成で設定されたデフォルトのパラメーターをオーバーライドします。

色とテーマ

詳細については、[詳細パースペクティブの色とテーマ](#)を参照してください。

注意:

このパースペクティブに設定された特定のパラメーターは、「詳細パースペクティブ」構成で設定されたデフォルトのパラメーターをオーバーライドします。

推奨されるパースペクティブ

パースペクティブ管理者は、特定のオーディエンス専用の推奨パースペクティブを構成できます。これらの推奨されるパースペクティブは、ユーザーのロールに基づいて、ユーザーがログインするときにデフォルトで適用されるパースペクティブを選択する方法です。

ただし、ユーザーは常に、利用可能なさまざまなパースペクティブを切り替えて、1 つをお気に入りとして設定することができます。詳細については、[パースペクティブモード](#)を参照してください。

推奨されるパースペクティブを構成するには、>> に移動します。

推奨されるパースペクティブの管理

メイン画面には、プロファイルとパースペクティブを関連付けるレコードの順序付きリストが表示されます。ユーザーは複数のレコードに一致する可能性があるため、ここでの順序は重要であることに注意してください (詳細については、[パースペクティブの決定](#)を参照してください)。

- エントリを追加するには、アクションを使用します。
- エントリを編集するには、最初にエントリをクリックしてリストから選択し、次にアクションをクリックするか、単にダブルクリックします。
- エントリを削除するには、最初にリストからエントリを選択してから、アクションをクリックします。
- エントリを移動するには、最初にリストでエントリを選択してから、リストの右側にあるツールのアクションを使用します。

パースペクティブの決定

ユーザーがログインすると、次のアルゴリズムによって、デフォルトで選択されるパースペクティブが決定されます。

```
// 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
```

カスタムビュー

ユーザーは、テーブルのメニューから直接カスタムビューを作成および管理できます。この管理セクションは、これらのカスタムビューを管理するための中心的なポイントです。

ビュー

このテーブルには、任意のテーブルで定義されたすべてのカスタムビューが含まれています。フィールドのサブセットのみが編集可能です。

ドキュメント	ローカライズされたラベルと説明
--------	-----------------

所有者	このビュー定義を所有および作成しているユーザーを定義します。
ビューグループ	このビューが [ビュー] メニューに表示されるメニューグループを示します。
共有先	このビューを [ビュー] メニューから選択できるユーザーを定義します。

ビューの権限

このテーブルを使用すると、データモデルとプロファイルごとに、カスタムビューに関連する権限を管理できます。次の権限を構成できます (特定のユーザーに権限が設定されていない場合、デフォルト値が適用されます)。

権限	説明	デフォルト値
推奨されるビュー	ユーザーが推奨ビューを管理できるようにします。	ユーザーがデータセットの所有者である場合、デフォルト値は です。それ以外の場合は です。
ビューの管理	ユーザーが変更および削除できるビューを定義します。	ユーザーがビルトインの管理者である場合、デフォルト値は「所有+共有」です。それ以外の場合は「所有」です。
ビューの共有	ユーザーが「共有」フィールドを編集できるビューを定義します。	ユーザーがビルトインの管理者である場合、デフォルト値は「所有+共有」です。それ以外の場合、ユーザーがデータセットの所有者である場合は「所有」、それ以外の場合は「なし」です。
ビューの公開	ユーザーがビューを公開して、Web コンポーネント、ワークフローユーザータスク、またはデータサービスを使用するすべてのユーザーが	ユーザーがビルトインの管理者である場合、デフォルト値は です。それ以外の場合は です。

権限	説明	デフォルト値
	ビューを利用できるようにします。	

ユーザーセッション管理

このツールはすべてのユーザーセッションを一覧表示し、必要に応じてアクティブなセッションを終了できるようにします。

例：メンテナンスの目的で、現在開いているアクティブなセッションをすべて無効にして終了することができます。[アプリケーションのロック](#)を使用すると、ユーザーインターフェイスへのアクセスを一時的に閉じて、使用不可メッセージを表示できます。アクティブなセッションが終了すると、ユーザーは再接続できなくなり、使用不可メッセージが表示されます。その後、メンテナンス操作を実行できます。

アクセシビリティモード

アクセシビリティモードは、障害を持つユーザーを支援するために利用できます。アクティブな場合、表示を微調整して、可能な限りユーザーインターフェイスにアクセスできるようにします。

このオプションは、ログイン画面と、接続された後のユーザーペインに表示されます。

選択した値はユーザーごとに保持され、すべてのワークセッションで共有されます。

フォームのすべてのコントロールをキーボードインターフェイスを介して操作可能にし、WCAG Success Criterion 2.1.1 に準拠させることができます。

アクセシビリティモードがオンのときに適用される変更の概要

表形式ビュー	すべての行ツールバーに [データにアクセス] ボタンを追加します。テーブルヘッダーセルとそのドキュメントボタンは、キーボードでフォーカスできます。
HTML エディター	ツールバーを常に展開するようにします。
フォーム	展開/折りたたみボタンとドキュメントボタンにフォーカスできます。

サポートされているスクリーンリーダー

Jaws (バージョン 2022) は、TIBCO EBX®でサポートされている唯一のスクリーンリーダーです。他のテクノロジーでは、予期しない動作が発生する可能性があります。

UI - ワークフローランチャー

- はじめに
- データセクションのワークフローランチャー
- ランチャーの作成と設定
- ワークフローランチャーのアクティブ化
- ワークフローの起動
- カスタムツールバーへのワークフローランチャーの追加
 - ツールバーの他の場所へのワークフローランチャーの追加
- ワークフローモデル変更後のランチャーへのアクセス

はじめに

ワークフローランチャーは、データワークフローの受信トレイを通過せずにデータセクションから直接 **TIBCO EBX®** でワークフローを起動するためのユーザーサービスです。この機能はワークフローパブリケーションを作成しませんが、既存のパブリケーションを起動します。**データセクション** (テーブルビュー、階層ビュー、またはレコードフォームビュー) から直接ワークフローパブリケーションを起動できるなど、いくつかの利点があります。このように、ユーザーがデータセクションとデータワークフローセクション間での注意の移動を回避することにより、ユーザーエクスペリエンスが向上します。したがって、ユーザーはメインタスクに集中しながらワークフローを起動できます。

2 つ目の利点は、任意のデータ選択から同じワークフローパブリケーションを起動できることです。これは、現在のデータ選択による**ワークフローデータコンテキストの動的マッピング**によるものです。動的マッピングは、起動時にデータコンテキスト入力を初期化する可能性を提供します。したがって、n 個の異なるデータ選択から同じワークフローを起動するために、異なるデータ選択で同じワークフローモデルを n 回複製したり、選択するデータを構成するための初期ユーザーサービスを提供したりする必要はなくなりました。

最後の解決策は、前述の問題を解決するプログラムによる解決策ですが、コードゼロ行のコミットメントを満たさないため、理想的な解決策ではありません。

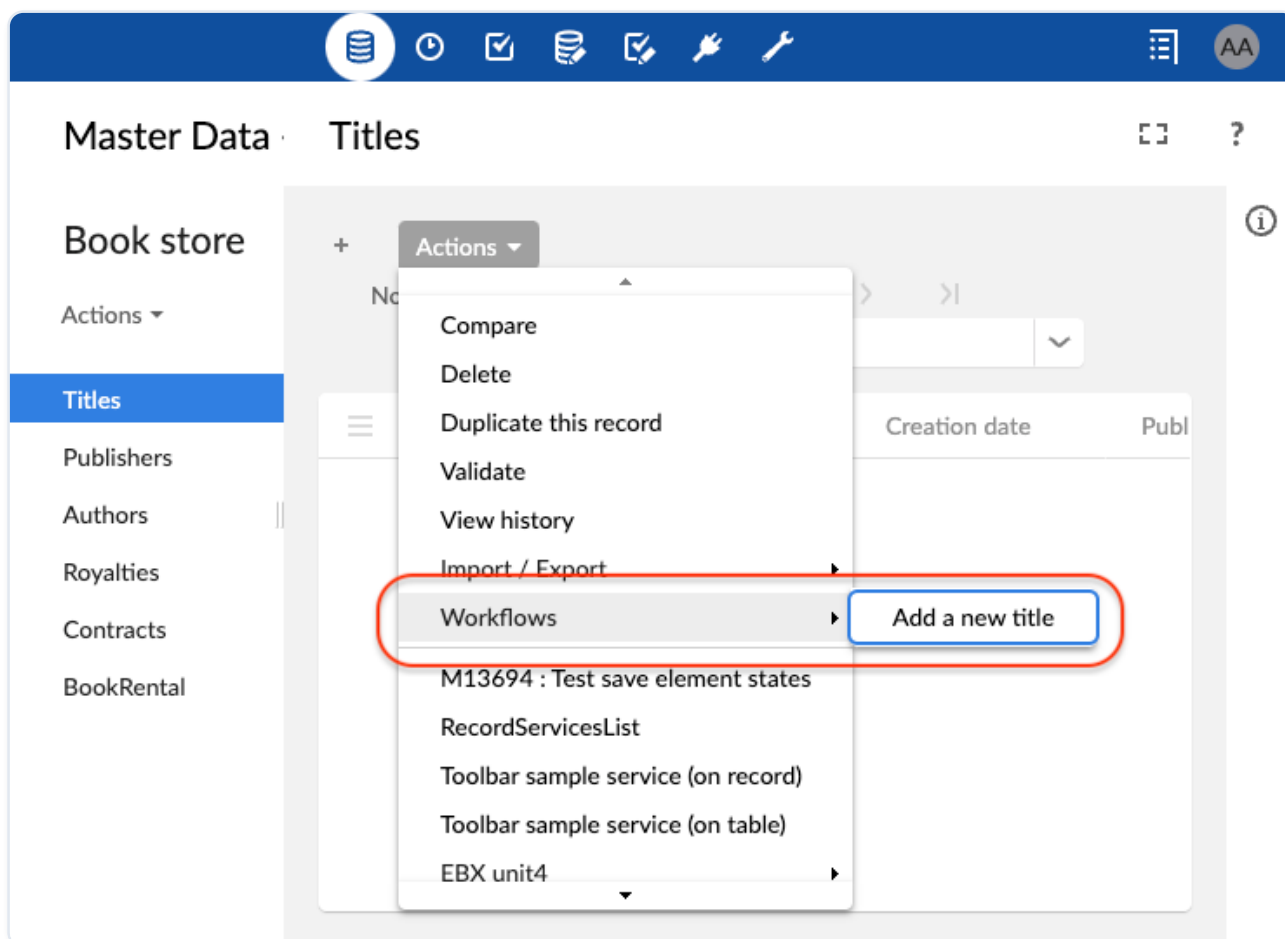
この機能を理解するには、次の概念に精通する必要があります。

概念	説明/リンク
ワークフローモデル	ドキュメントを参照
データコンテキスト	ドキュメントを参照
ワークフローパブリケーション	ドキュメントを参照
パブリケーション名	ワークフローパブリケーションの一意的識別子
データワークフロー	ドキュメントを参照
データセクション	EBX® のデータセットとテーブルを表示する データユーザーインターフェイス です。メインヘッダーからアクセスできます。
ワークフローランチャーデータセット	これには、 Launchers と Activations という 2 つのテーブルが含まれています。このデータセットを使用すると、ユーザーはワークフローパブリケーションのランチャーを構成し、特定のユーザーとテーブルに対してアクティブ化することができます。これは、というセクションの 領域 で利用できます。
Launcher (ランチャー)	ワークフローランチャーサービスが起動するワークフローパブリケーションとそのデータコンテキストを初期化する方法を識別するために使用するエンティティ。
Launchers	ワークフローランチャーデータセットのテーブル。各レコードは Launcher タイプです。データセクションからワークフローパブリケーションを起動できるようにするには、ワークフローパブリケーションを指すランチャーをこのテーブルに追加する必要があります。
アクティベーション	特定のユーザープロファイルおよび特定のテーブルのワークフローパブリケーションのランチャーをデータセクションから非表示または表示するエンティティ。

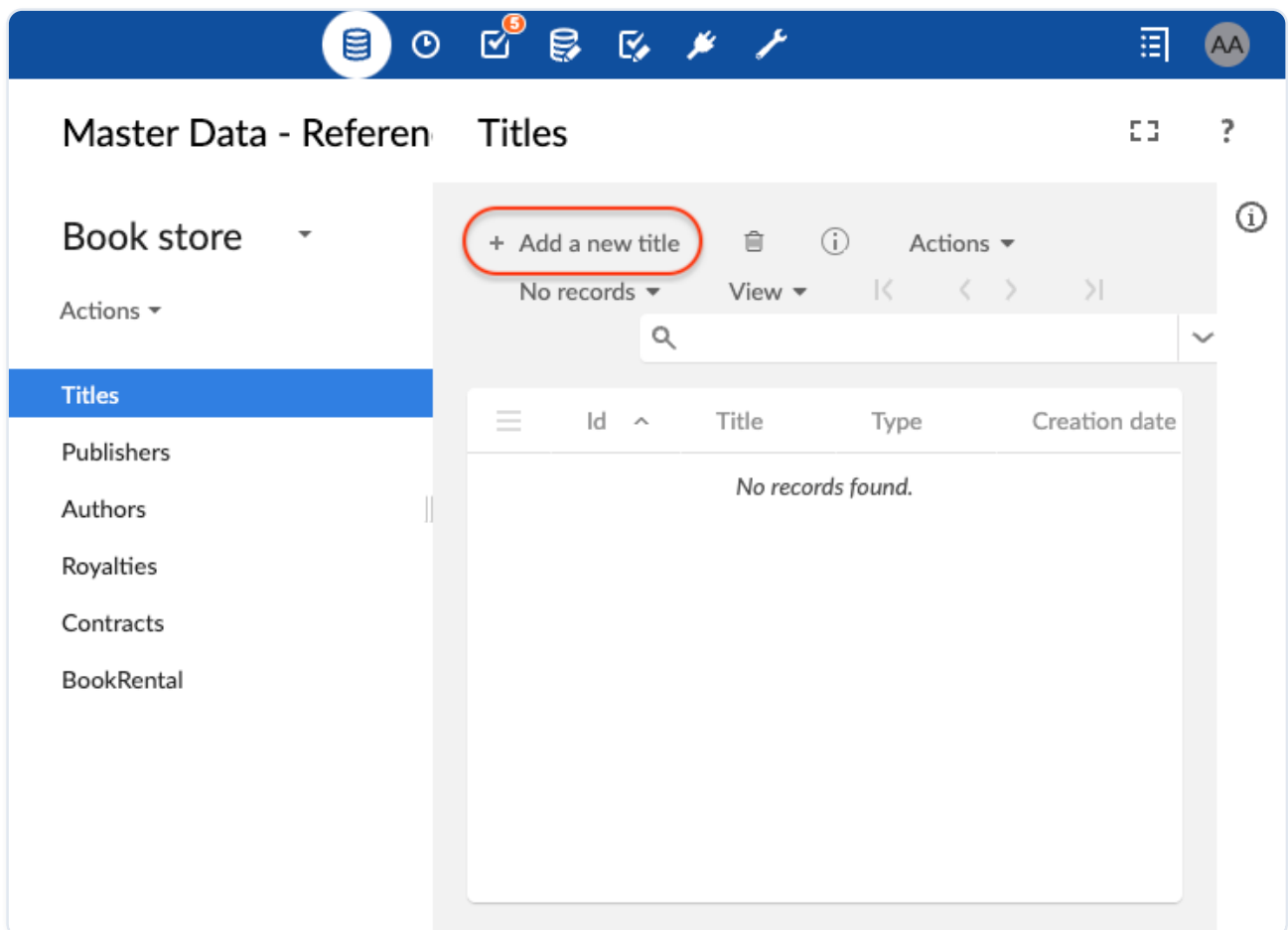
概念	説明/リンク
Activations	ワークフローランチャーデータセットのテーブル。各レコードは Activation タイプです。特定のテーブルのツールバーで特定のユーザーがワークフローを利用できるようにするには、対応するランチャーを含むレコードをこのテーブルに作成する必要があります。
データコンテキストの動的マッピング	ワークフローパブリケーションを起動するときに、データコンテキスト入力変数を初期化するプロセスです。以前は、データコンテキストの値はモデリング段階で静的に定義されるか、Java コーディングで動的に設定されていましたが、現在は、データコンテキストの値を、つまり起動時にゼロ行のコードで動的に設定することができます。
データコンテキスト用に予約された変数	予約変数のセット。これらの変数は、どのデータ選択をデータ コンテキスト変数にマップするかを定義します。可能な値は、 \$ {dataspace} 、 \$ {dataset} 、 \$ {xpath(table)} 、および \$ {xpath(record)} です。たとえば、値 \$ {xpath(table)} がデータコンテキスト変数に使用されている場合、これは、この変数が起動時に現在のアダプテーションテーブル参照にマップされることを意味します。

データセクションのワークフローランチャー

前述のように、ワークフローランチャーは、テーブル、テーブル行、レコード、および階層のツールバーで直接使用できるようになりました。データセクションのテーブルのツールバーにワークフローランチャーを表示するには、いくつかの方法があります。表示は、ツールバーのタイプ (デフォルト ツールバーまたは DMA のカスタムツールバー) と、**スマートフィルタリング** が適用されているかどうかによって異なります。デフォルトのツールバーの場合、アクションメニューはすべてのワークフローランチャーを という別のサブメニューに表示します (下のスクリーンショットを参照)。



カスタムツールバーの場合、特定のワークフローランチャーのアクションボタン (下のスクリーンショットを参照) またはアクションメニュー項目を定義することができます。最後に、スマートフィルタリングポリシーがアクティブになっている場合、アクションボタンまたはアクションメニュー項目を使用して表示されるすべてのワークフローランチャーは、アクションメニューに表示されません。



ツールバーにワークフローランチャーを表示するには、まずランチャーを作成し、構成する必要があります (ワークフローランチャーの作成と設定セクションを参照)。次に、このランチャーに対してアクティベーションを作成する必要があります (ワークフローランチャーのアクティブ化セクションを参照)。現在のユーザーにワークフローパブリケーションを起動する権限がない場合、ワークフローランチャーはツールバーで使用できないことに注意してください。また、ワークフローパブリケーションが削除された場合、またはランチャーの構成、アクティブ化にエラーや警告がある場合、対応するワークフローランチャーはツールバーで使用できなくなります。

ツールバーに表示されるボタンのタイトルとツールヒントは、次の優先度で計算されます。まず、ランチャーアクティベーションのカスタムドキュメントを使用し、それ以外の場合はランチャーのドキュメントを使用します。次に、説明が空のままの場合は「このユーザーサービスはデータワークフローを起動します」という説明を使用します。

注意:

レコードフォームとテーブルの行で利用できるワークフローランチャーには、特に注意を払う必要があります。レコードの選択が必要なランチャーのみが、レコードフォームとテーブル行のツールバーに表示されます。レコードの選択が必要なワークフローランチャーは、

データコンテキスト変数の 1 つがそのランチャーの構成で予約済み変数 `${xpath(record)}` にマップされているものです。

ランチャーの作成と設定

ランチャー は、サービスワークフローランチャーによって、起動するワークフローパブリケーションとそのデータコンテキストの初期化方法を識別するために使用されるエンティティです。特定のワークフローパブリケーションのランチャーを作成するには、領域から セクションに移動し、ワークフローランチャーデータセットとランチャーテーブルを選択して、新しいレコードを追加します (以下のフォームを参照)。

The screenshot displays the 'Administration' console with the 'CreateRecord' form for a workflow launcher. The left sidebar shows 'Workflows launchers' with 'Launchers' expanded and 'CreateRecord' selected. The main form contains the following fields and options:

- Launcher name:** CreateRecord
- Workflow publication name:** Create a new record
- Documentation:** English (United States) Create a new record
- Data context variables:**
 - Dataset: `${dataset}`
 - Dataspace: `${dataspace}`
 - Table: `${xpath(table)}`

At the bottom of the form are four buttons: 'Save', 'Save and close', 'Revert', and 'Close'.

2 番目のステップは、レコードの次のフィールドを設定することです。

フィールド名	説明
ランチャー名	一意の識別子。アクティブ化フェーズでランチャーを選択するために使用されます。
ワークフローパブリケーション名	ユーザーサービス「ワークフローランチャー」の実行時に起動されるワークフ

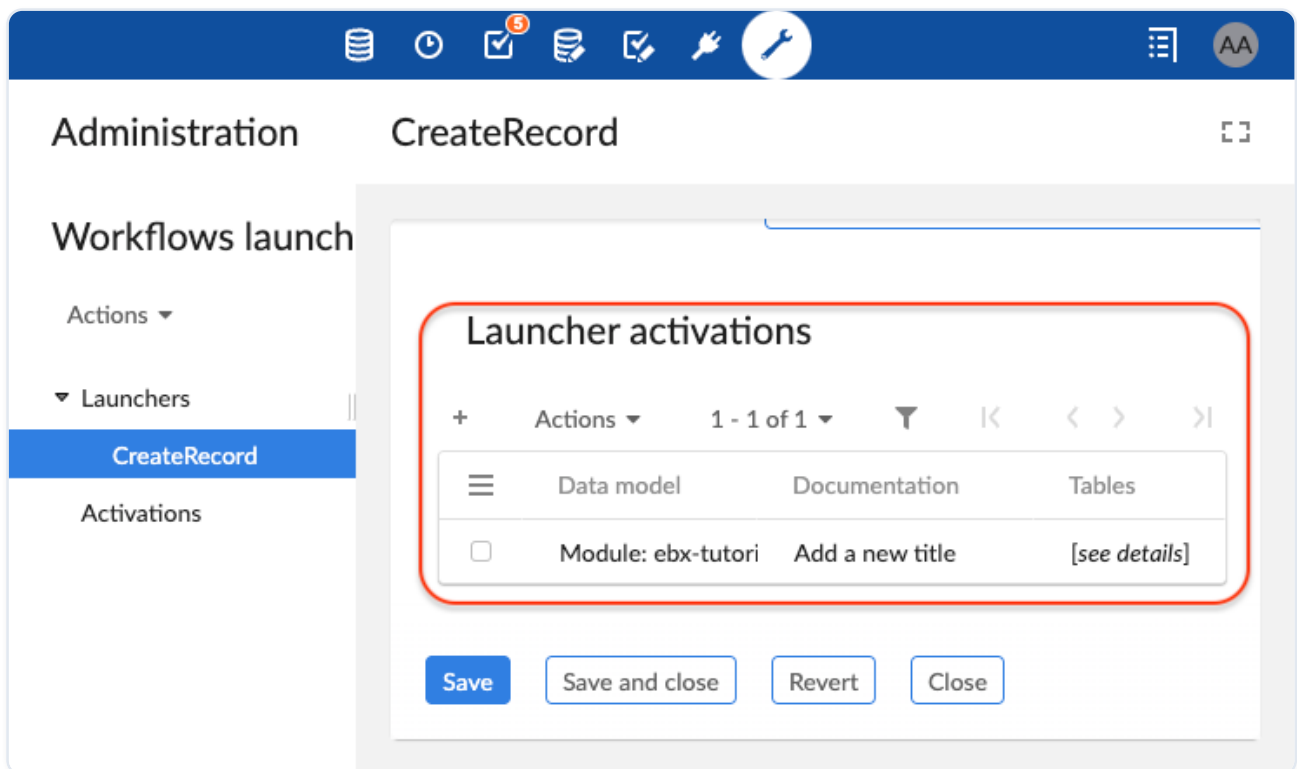
フィールド名	説明
	ローパブリケーションを定義します。リストで使えるようにするには、ワークフローモデルを最初に定義して公開する必要があります。
データコンテキスト変数	このフィールドには、ワークフローモデルのデータコンテキスト構成フェーズで定義された入力変数のリストが含まれます。各行は、ラベル、値、およびデフォルト値と上書き値を切り替えるためのトグルボタンで構成されます (ランチャーフォームの データコンテキストに関するスクリーンショット を参照)。ラベルは、ワークフローモデルのデータコンテキストでユーザーによって設定された変数の名前です。 デフォルトでは、各変数の値はデータコンテキストで設定されたデフォルトの値です。データコンテキストでデフォルト値フィールドが空のままの場合、変数の値は undefined に設定されます。トグルボタンが上書き値モードに設定されている場合、入力フィールドの右側でウィザードが使用可能になり、予約済み変数を選択できます。 データコンテキスト変数の値を上書きする際には、2つのオプションが考えられます。上書きする値は、定数または予約変数です。定数が使われた場合、データコンテキスト変数の値は、起動時のエンティティ選択に依存しません。ただし、<code>\${dataspace}</code> のような予約変数が使われた場合には、対応する変数の値は、現在のエンティティ選択にマッピングされます。たとえば、<code>\${dataspace}</code> に割り当てられたデータコンテキスト変数は、

フィールド名	説明
	起動時に現在のデータスペースで初期化されます。 注意: ワークフローモデルのデータコンテキストに入力変数がない場合は、このフィールドを非表示にする必要があります。ワークフローパブリケーション名フィールドが変更されると、この入力自動的に更新され、データコンテキストの新しいリストが表示されます。
ドキュメント	ドキュメントは、ラベルと説明から構成されます。ラベルのデフォルト値「ワークフローパブリケーション名」から継承されるパブリケーション名で、説明のデフォルト値は空白です。[パブリケーション名]のフィールドが変更された場合、ドキュメントは自動的に更新されます。必要に応じて、ドキュメントウィジェットの右側にある編集ボタンを使用して、ラベルと説明を上書きすることができます。ドキュメントの値は、アクティベーションで継承されます。なお、アクティベーションで定義されたドキュメントは、ランチャーで定義されたものよりも優先されます。

Data context variables

Dataset	\${dataset} Reserved variable	Wizard 	Toggle 
Dataspace	\${dataspace}		
Table	\${xpath(table)}		

ランチャーレコードの下部に、現在のランチャーのすべてのアクティベーションのテーブルが表示されます (下のスクリーンショットのランチャーのアクティブ化のセクションを参照)。



ワークフローランチャーのアクティブ化

アクティベーションは、テーブルまたはレコードフォームでのワークフローランチャーの可用性を制御するエンティティです。特定のテーブルのツールバーにワークフローランチャーを表示するには、最初に、[ランチャーを作成](#)してから、領域からセクションを選択し、ワークフローランチャーデータセットとアクティベーションテーブルを選択します。次に、新しいレコードを追加します。2 番目のステップは、レコードの次のフィールドを設定することです。

フィールド名	説明
Launcher (ランチャー)	ターゲットワークフローパブリケーションの公開名に関連付けられているランチャーの一意の識別子。このフィールドでは、データセクションに表示されるランチャーを選択できます。
データモデル	公開されたデータモデルのスキーマ参照。データセットで公開および使用されているスキーマのみが使用可能です。
テーブル	ワークフローランチャーを表示するテーブルの識別子。1 つまたは複数のテーブルを選択できます。このセクターは、 [データモデル]

フィールド名	説明
	フィールドに含まれるすべてのテーブルを表示します。
プロファイル	以前に選択したテーブルでワークフローランチャーを表示および起動できるプロファイルのリスト。
確認	このフィールドでは、データワークフローの起動を確認するダイアログボックスを表示するかどうかを指定できます (起動確認メッセージに関するスクリーンショット を参照)。デフォルトでは、この機能は無効になっています。ダイアログボックスを表示するには、値のチェックをオンにする必要があります (以下のスクリーンショットを参照)。
ドキュメント	ドキュメントはラベルと説明で構成されます。ラベルと説明のデフォルト値は、ランチャーのドキュメントフィールドから継承されます。このフィールドは、ツールバーのユーザーサービスのボタンのタイトルを表示するために使用されます。ユーザーがそのボタンの上にマウスポインターを移動すると、説明が表示されます。アクティベーションのドキュメントは、ランチャーのドキュメントよりも優先されます。なお、ドキュメントの説明が空のままの場合、「このユーザーサービスはデータワークフローを起動します」という説明が表示されることに注意してください。カスタムツールバーでは、 アクションボタン または アクションメニュー項目 のドキュメントフィールドが空のままの場合、このフィールドの値が アクションボタン と アクションメニュー項目 にも使用されます。

The screenshot shows the 'Administration' console with the 'Add new activation' form. The left sidebar is titled 'Workflows launchers' and includes a menu with 'Actions', 'Launchers', and 'Activations' (which is selected). Under 'Activations', the 'New record' button is highlighted. The main form area contains the following fields:

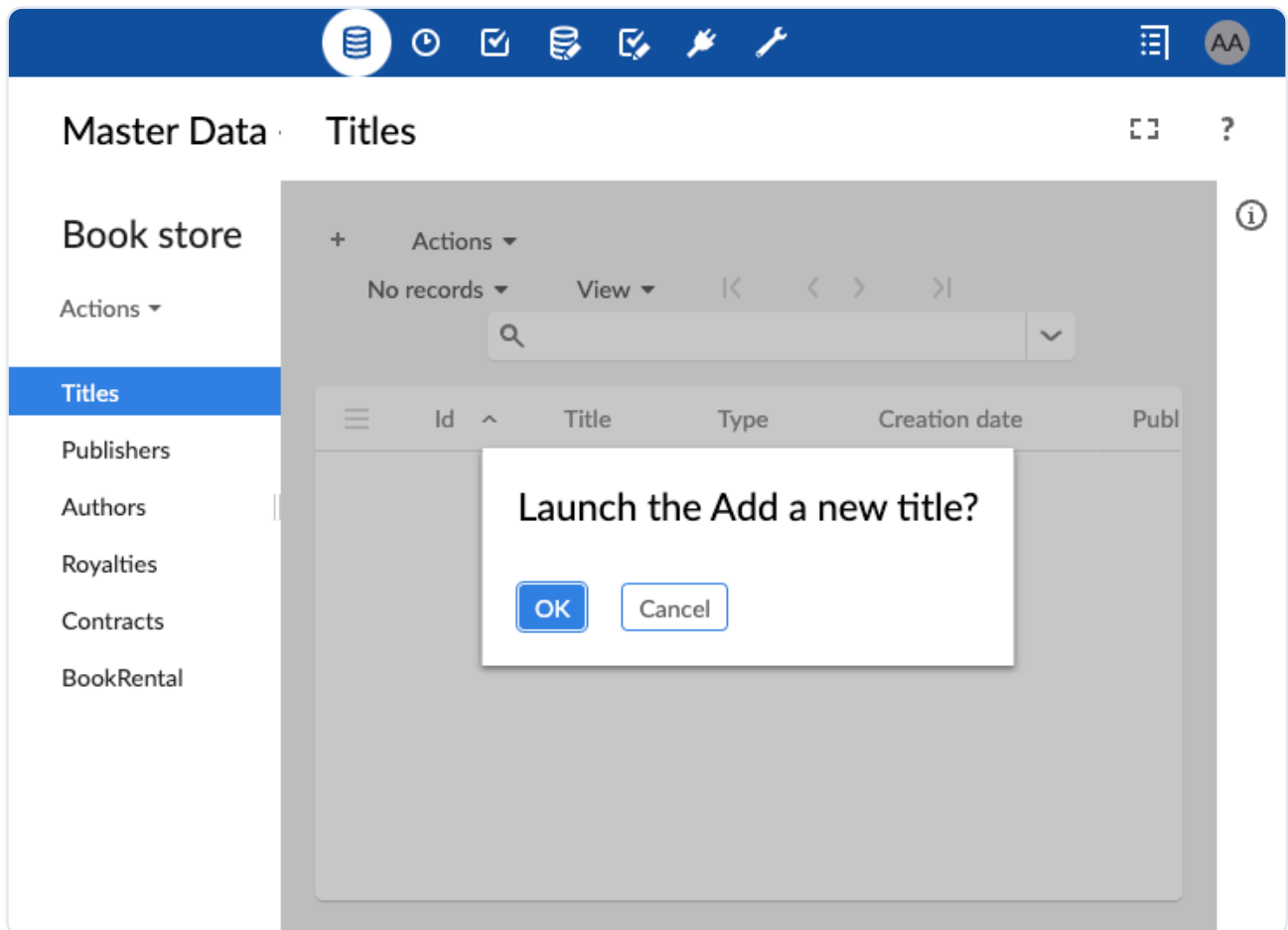
- Launcher**: CreateRecord
- Data model**: Module: ebx-tutorial, path: /WEB-INF/ebx/scher
- Tables**: 1. Titles
- Profiles**: 1. [all profiles]
- Confirmation**: ☒ Enable ☐ Disable
- Documentation**: English (United States)

At the bottom of the form are three buttons: 'Save', 'Save and close', and 'Close'.

ワークフローの起動

ランチャーが作成され、アクティブ化されると、対応するワークフローランチャーがツールバーのアクションメニューで使えるようになります。ワークフローを起動するには、ワークフローランチャーのボタンをクリックします (ワークフローランチャーのデフォルト表示に関するスクリーンショットを参照)。

[確認] オプションが有効になっている場合、選択したデータワークフローを起動する前に、ユーザーに確認を求めるダイアログボックスが表示されます (起動確認メッセージに関する以下のスクリーンショットを参照)。



それ以外の場合、ダイアログボックスは表示されず、異なる結果が生じる可能性があります。

- **[最初のステップを自動的に開く]** オプションが有効化されており、ユーザーがワークフローの最初の作業項目を実行する権限を持っている場合、ワークフローが起動し、最初の作業項目がデータセクションのワークスペースに直接表示されます。ワークスペースに作業項目を表示している間、元のデータ選択 (ソーステーブル) が階層リンクに表示されたままになるため、ユーザーは全体的なコンテキストの認識を維持できます (**データセクションでの作業項目の表示に関するスクリーンショット**を参照)。
- **[最初のステップを自動的に開く]** オプションが無効になっていて、ユーザーがワークフローを起動する権限を持っている場合、ワークフローが起動し、データワークフローの受信トレイに追加されます。情報メッセージによって、ワークフローが正常に起動されたこと、および最初の作業項目を表示して実行するためにデータワークフローセクションの受信ボックスに移動する必要があることが、ユーザーに通知されます。
- ワークフローでレコードの選択が必要な場合 (**データ領域のワークフローランチャーのセクションの「注意」**を参照)、レコードが選択されている場合にのみワークフローが起動および表示され、それ以外の場合は、エラーメッセージが表示されて、レコードを選択する必要があることがユーザーに通知されます。レコードは、テーブルビューから手動で選択することも、レコードフォームを表示するときに自動的に選択することもできます。

- 複数のレコードが選択されている場合は、1つのレコードのみを選択する必要があることが、警告メッセージによって通知されます。

The screenshot shows the 'Master Data' section with a sidebar menu containing 'Book store', 'Titles', 'Publishers', 'Authors', 'Royalties', 'Contracts', and 'BookRental'. The 'Titles' form is open, displaying a 'Source Table' and 'Work item' header. The form fields include:

- Id:** A000001
- Title:** (empty field)
- Type:** Unknown
- Creation date:** (date picker)
- Publisher:** [not defined]
- Author:** [not defined]
- Author age:** 0
- Price:** (empty field)

At the bottom, there are 'Save' and 'Save and close' buttons. The breadcrumb navigation at the top shows 'Master Data > Titles > Create a record > Create a new book'.

カスタムツールバーへのワークフローランチャーの追加

ワークフローランチャーは、デフォルトのツールバーだけでなく、[カスタムツールバー](#)でも使用できます。ワークフローランチャーは、カスタムツールバーの[アクションボタン](#)、または[アクションメニュー項目](#)を[カスタムメニューグループ](#)として追加できます。アクションボタンまたはアクションメニュー項目を追加してカスタムツールバーでワークフローを起動するには、一般的なユーザーサービスの場合と同様の手順に従います。まず、[ターゲットフィールド](#)を[現在のコンテキスト](#)に設定します。そうしないと、ユーザーサービスが使用できなくなるため、ユーザーサービスを選択します。サービスを選択すると、ユーザーサービス専用の新しいフィールドが、サービス入力フィールドの下に表示されます (以下の [カスタムツールバーへのワークフローランチャーの追加に関するスクリーンショット](#) を参照)。

フィールドには、「アクティベーション/ワークフロー管理/ワークフローランチャー/ランチャー」のテーブルにすでに作成されているランチャーのリストが表示されます ([ランチャーの作成と構成に関するセクション](#) を参照)。したがって、カスタムツールバーにワークフローランチャーを追加する前に、このワークフローランチャー用にランチャーを作成して構成する必要があります。

デフォルトのツールバーの場合、カスタムツールバーにワークフローランチャーを表示するには、>>>[アクティブ化]で、そのランチャーをアクティブにする必要があります。ランチャーのアクティブ化テーブルにアクセスするためのクイックリンクが、フィールドの右側に表示されます ([カスタムツールバーの \[ワークフローランチャー\] サービスに関する以下のスクリーンショット](#) を参照)。

フィールドには、ランチャーから継承されたラベルと説明が表示されます ([カスタムツールバー設定フォームの「ワークフローランチャー」サービスのラベルに関するスクリーンショット](#) を参照)。ランチャーフィールドが変更されると、ラベルと説明は自動的に更新されます。このフィールドをオーバーライドして、ツールバーのアクションボタンのラベルと説明をカスタマイズできま

す。ツールバーに表示されるラベルと説明は、次の優先度で計算されます。フィールドのカスタム値アクションボタンのラベルと説明。このフィールドが空のままであるか、アクションボタンのデフォルトのラベルが含まれている場合に使用されます。ランチャーを選択すると、ランチャーのアクティベーションのラベルと説明が表示されます。この最後の項目が空のままの場合、「このユーザーサービスはデータワークフローを起動します。」という説明が使用されます。

注意:

セクションのツールバーに表示されるラベルと説明、および DMA のツールバーツリーのアクションボタンのラベルは異なる場合があります。これは、ボタンのフィールドが空のままの場合です (デフォルトラベルもカスタムラベルも定義されていません)。DMA のツールバーツリーのボタンのラベルはランチャーのラベルから継承されますが、データセクションのツールバーのラベルと説明はランチャーのアクティブ化のラベルと説明に設定されます。

Label and description ⓘ The default value is inherited from the workflow launcher. The label of the button that will be displayed on the toolbar is computed as follows : the label of the activation is displayed. If the latter is not available, then the label of the launcher will be displayed.

▼ English (United States) ✎
Create a new record

ツールバーの他の場所へのワークフローランチャーの追加

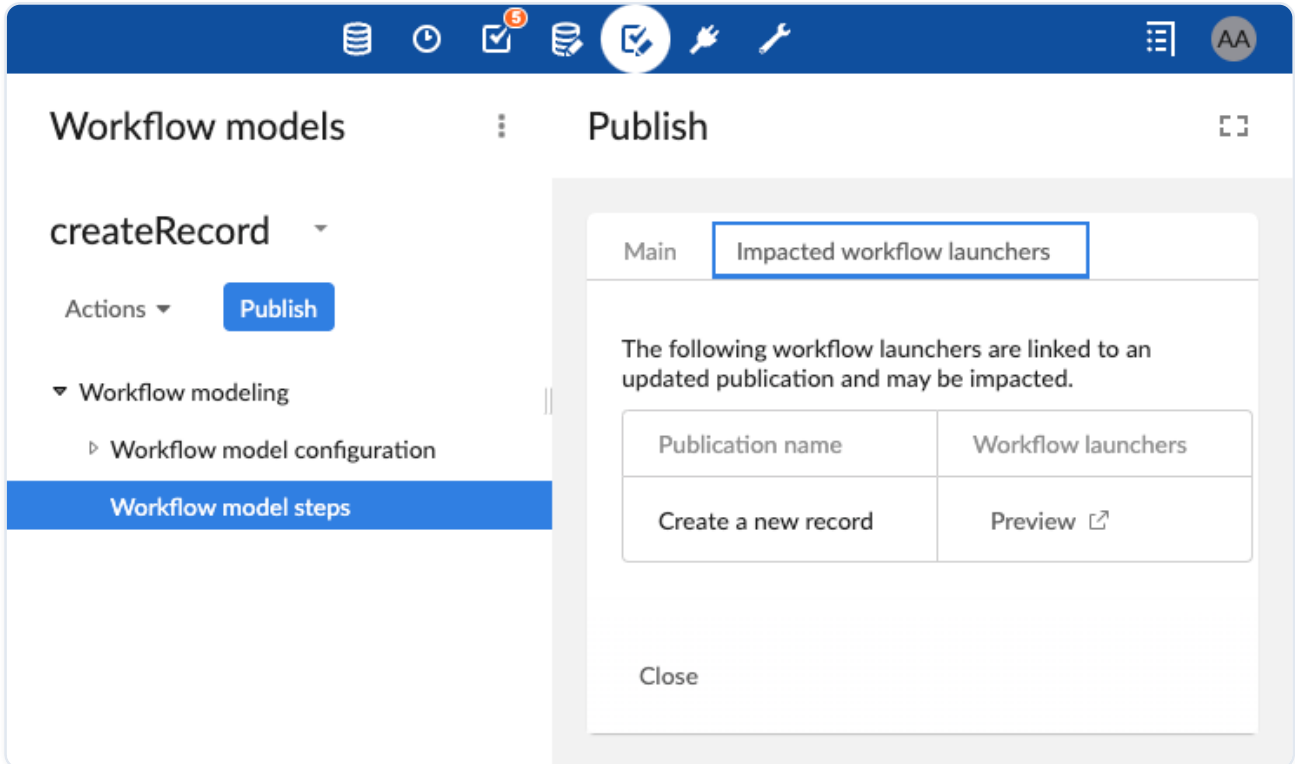
ワークフローランチャーサービスは、テーブルビュー上部、テーブル行、レコードフォームビュー上部、または階層ビュー上部に表示されるツールバーでのみ使用できます。現在まで、このサービスは、関連付けテーブルの上または関連付けテーブルの行に表示されるツールバーでは使用できません。

ワークフローモデル変更後のランチャーへのアクセス

ワークフローパブリケーションを指すランチャーを作成した後、このパブリケーションのワークフローモデルを変更して再公開できます。たとえば、ワークフローモデルのデータコンテキストの変更、1 つ以上の変数データコンテキストの追加、1 つ以上の可変データコンテキストの削除、変数名の変更が行えます。この特定のケースでは、これらのランチャーで使用されるデータコンテキストは無効になり、最新のワークフローパブリケーションのものと一致するように更新する必要があるため、このワークフローパブリケーションを指すランチャーは、ユーザーが確認および検証 (保存) する必要があります。

ワークフローモデルを再公開すると、ワークフローランチャーのいずれかが現在のワークフローパブリケーションを指しているかどうか、またそれを確認する必要があるかどうかユーザーに

通知されます。そのために、ワークフローモデルを公開した後、問題のランチャーにすばやくアクセスできるプレビューボタンが表示されます (以下のスクリーンショットを参照)。ユーザーがワークフローランチャーリストにアクセスする権限を持っていない場合、それらのランチャーの名前が表示されることに注意してください。



ユーザーがセクションの ボタンまたは セクションを介してこれらのランチャーの 1 つにアクセスすると、永続化されたデータコンテキストは、ワークフローパブリケーションの最新のデータコンテキストに置き換えられますが、新しいデータコンテキストは現在のランチャーではまだ永続化されていません。ランチャーのデータコンテキストを更新するには、ユーザーは最初に値を確認してから保存して、新しい変更を受け入れる必要があります (対応するワークフローパブリケーションのデータコンテキストの変更後のランチャーへのアクセスに関するスクリーンショットを参照)。

Data context variables

i The data context of the selected workflow publication has been modified. In the box below, the old data context is replaced by the new data context. Note that the default values are used for new data context values. Please review the default values and save the form to accept the new changes.

Dataset

Information message

✖

🗑

Dataspace

✖

🗑

Record

✎

The new input data context

Table

✖

🗑

ユーザーとロールのディレクトリ

- 概要
- ディレクトリの概念概要
 - ポリシー
- メインディレクトリ
 - デフォルトディレクトリ
 - LDAP ディレクトリ
 - カスタムディレクトリ
- 同期ディレクトリ
 - LDAP ディレクトリ
 - SCIM ディレクトリ
 - ログイン時ディレクトリ
 - ロールマッピング
 - 移行

概要

TIBCO EBX®は、ユーザー認証とロール定義を管理するためにディレクトリシステムを使用します。メインディレクトリは必須であり、認証と認可のデフォルトソースです。

シングルサインオン (SSO) が有効な場合、EBX®はメインディレクトリに加えて同期ディレクトリを使用できます。同期ディレクトリにより、ID プロバイダーを介して認証された外部ユーザーの管理が可能になります。この設定では、次のようになります。

- 技術ユーザーにはメインディレクトリを使用します
- 外部システムによって管理されるユーザーには同期ディレクトリを使用します

以下に、ユーザー認証と認可を処理するためのディレクトリ設定の一般的な手順を示します。

- メインディレクトリの実装を選択します: デフォルト、ビルトイン LDAP、またはカスタムディレクトリ
- 必要に応じて、OIDC または SAML2 のいずれかのプロトコルを使用して SSO を有効にします
- 必要に応じて、ビルトイン実装のいずれか (LDAP、SCIM、またはログイン時) を使用して同期ディレクトリを有効にします

- 選択したディレクトリとプロトコルを構成します

以下も参照してください。:

- TIBCO EBX® ディレクトリプロパティの構成

注意:

本番環境ではデフォルトディレクトリを使用しないでください。代わりに、より安全なパスワードシステムを備えた独自のカスタムディレクトリ、または機密情報を保護するためのビルトインディレクトリオプションを使用してください。

メインディレクトリと同期ディレクトリの両方を使用する場合、ユーザー名はディレクトリ間で一意である必要があります。ユーザー名が両方に存在する場合、システムは両方のディレクトリからロールをマージします。ただし、ユーザーの情報(名前、メールなど)は同期ディレクトリからのみ取得されます。これにより、不整合や意図しない上書きが発生する可能性があるため、ディレクトリ間で一意のユーザー名を強く推奨します。

ディレクトリの概念概要

TIBCO EBX®では、ユーザーは1つ以上のロールのメンバーになることができ、複数のユーザーが同じロールを共有できます。さらに、ロールは他のロールを含むことができます。ドキュメントでは、ユーザーまたはロールのいずれかを記述するために、一般的な用語 **プロファイル** を使用します。プロファイルに関連する情報は、主にディレクトリで定義されます。

ディレクトリで定義するロールに加えて、TIBCO EBX®は次の **ビルトインロール** を提供します。

ロール	定義
Profile.ADMINISTRATOR	管理タスクへのフルアクセスを許可します
Profile.READ_ONLY	ユーザーはコンテンツのみを表示できます。リポジトリの変更は許可されません
Profile.OWNER	ユーザーが現在のエレメントの所有者として定義されたプロファイルである (または含まれ

ロール	定義
	ている) 場合にのみアクティブになる動的ロール
Profile.EVERYONE	すべてのユーザーがこのロールに含まれます

注意:

ユーザーとビルトインロール *OWNER* および *EVERYONE* の間の関連付けは、TIBCO EBX®によって自動的に管理されるため、ディレクトリを介して変更してはなりません。

ユーザー権限はディレクトリとは別に管理されます。[権限](#) を参照してください。

以下も参照してください。:

- プロファイル
- ロール
- ユーザー
- 管理者
- ユーザーとロールのディレクトリ

ポリシー

ディレクトリ > ポリシー テーブルでは、ユーザーが自分のプロファイルの特定の側面を管理できるかどうかを構成できます。たとえば、ポリシーは以下を指定できます。

- ユーザーが自分のアカウント情報を更新できるかどうか
- パスワードリマインダーオプションも利用できるかどうか

メインディレクトリ

メインディレクトリは必須であり、ユーザー名とパスワードを使用してログイン認可を処理します。EBX®は、[デフォルトディレクトリ](#)、[LDAP ディレクトリ](#)、または[カスタムディレクトリ実装](#)の次のディレクトリタイプをサポートしています。

デフォルトディレクトリ

デフォルトディレクトリと関連テーブルは、**管理 > ユーザー、ロール、および設定 > ディレクトリ** データセットにあります。このデータセットには、ユーザーポリシー、メーリングリスト、ユーザー、およびロールを作成および保存するためのテーブルが含まれています。デフォルトディレクトリを削除または複製することはできません。

定義されたポリシーに応じて、ディレクトリデータセットに定義された権限に関係なく、ユーザーは自分のアカウントに関連する情報を変更できます。

注意:

ロール包含サイクルが検出された場合、権限解決中にロール包含は無視されます。このようなサイクルを検出するには、ディレクトリ検証レポートを更新して確認します。

注意:

ユーザーロール および **ロール包含と敬称** テーブルは、**デフォルトでは非表示** です。

パスワード回復手順

デフォルトディレクトリでは、パスワードは暗号化され (デフォルトでは SHA256 のようなアルゴリズムを使用)、この状態で保存されます。その結果、紛失したパスワードは取得できません。新しいパスワードを生成してユーザーに送信する必要があります。

この手順には 2 つのオプションがあります。

1. 通知メールが管理者に送信され、管理者は手動でパスワードを変更し、新しいパスワードをユーザーに送信します。これがデフォルトオプションです
2. プロシージャは自動的に新しいパスワードを生成し、それをユーザーに送信します。このオプションを有効にするには、**TIBCO EBX® メイン構成ファイル**で次のプロパティを設定します:

```
ebx.password.remind.auto=true
```

注意:

セキュリティ上の理由から、上記のパスワード回復手順は管理者プロファイルでは利用できません。代わりに管理者回復手順を使用してください。

管理者回復手順

すべての管理者資格情報が失われた場合は、次の特別な回復プロセスを使用して、ディレクトリクラスでログインとパスワードを `admin / admin` とする一時的な管理者アカウントを再定義します。

1. TIBCO EBX® [メイン構成ファイル](#)で次のプロパティを指定します。

```
ebx.directory.factory=
com.orchestranetworks.service.directory.DirectoryDefaultRecoverFactory
```

2. TIBCO EBX®を起動し、手順が完了するまで待ちます
3. 'ebx.directory.factory' プロパティをリセットします
4. TIBCO EBX®を再起動し、「admin」アカウントを使用して接続します

注意:

'ebx.directory.factory' プロパティが回復手順に設定されている間、他のすべてのユーザー認証は無効になります。

LDAP ディレクトリ

EBX®は、LDAP (Lightweight Directory Access Protocol) をメインディレクトリまたは同期ディレクトリのいずれかとしてサポートしています。[同期 LDAP ディレクトリ](#)で、同期ディレクトリとして使用する方法を参照してください。LDAP をメインディレクトリとして使用するには、[ユーザーとロールディレクトリの構成](#)でディレクトリファクトリをビルトイン LDAP ディレクトリファクトリに設定します。

カスタムディレクトリ

カスタムディレクトリは、EBX®のビルトインディレクトリを使用する代替手段です。リレーショナルデータベースに含まれる特定の企業ディレクトリ、またはTIBCO EBX®にインスタンス化された特定のディレクトリデータモデルを統合できます。デフォルトのログインページを特定の企業ページに設定することもできます。対応するドキュメントリソースについては、以下のリンクを参照してください。

以下も参照してください。:

- TIBCO EBX® Container Edition 環境変数の構成
- `DirectoryFactory`
- `Directory.getUserAuthenticationURI`

同期ディレクトリ

同期ディレクトリは、構成後に **管理** 領域で利用可能なデータセット **ebx-synchronized-directory** によって表されます。これは、外部ユーザーを認証および認可するために使用される統合ディレクトリです。EBX®は内部ロールのみを使用して権限を管理します。外部ロールは内部ロールにマッピングする必要があります。

同期ディレクトリは、EBX®で SSO と組み合わせて使用する必要があります。

- ディレクトリは、最新のアクセス制御を保証するために、自動プロビジョニングでユーザーとロールを管理するために使用されます
- SSO はユーザー認証を処理します

このアプローチは、一元化されたユーザー管理を提供し、システム全体で一貫したアクセス制御を保証します。

同期ディレクトリとそのロールマッパーは、メイン EBX® 構成ファイルを通じて構成されます。[同期ディレクトリの構成](#)で手順を参照してください。

LDAP ディレクトリ

TIBCO EBX®の LDAP (Lightweight Directory Access Protocol) のビルトインサポートは、メインディレクトリおよび同期ディレクトリの実装として設計されており、既存の LDAP ディレクトリを TIBCO EBX®と統合できます。これにより、組織の LDAP インフラストラクチャのニーズに合わせて調整された、安全で一貫性のあるアクセス制御を通じて、合理化された効率的なディレクトリ管理エクスペリエンスが保証されます。

ロールの同期は、タスク

`com.orchestranetworks.service.directory.scim.SchedulerLdapSynchronization` をスケジュールすることで行われます。

LDAP 構成

次の4種類の構成パラメーターが利用可能です。

- 接続パラメーターは、LDAP サーバーへの接続を確立および処理するために必要です。これらは、最適なパフォーマンスとセキュリティのために接続を調整するためにも使用されます

LDAPS を使用する場合、ルート証明書が JDK TrustStore がない場合は、ルート証明書をインポートする必要があります ([TrustStore へのルート証明書のインストール](#)を参照)

- マッピングパラメーターは、特定のビルトインロールとユーザーを LDAP ディレクトリ属性にマッピングするために使用されます
- LDAP 属性から派生した式を使用したユーザーとロールの表示
- 検索要求テンプレートは LDAP 要求を生成し、LDAP ディレクトリから正確かつ効率的に情報を取得できるようにします

検索リクエストテンプレート

LDAP 検索クエリは、次の 3 つの主要エレメントで構成されます。

- ベース DN (識別名): ディレクトリツリー内の開始点を設定します
- 検索フィルター: 検索する属性と値を定義します
- 検索範囲: ディレクトリツリー内の検索の深さ (ベースから数レベルの深さまで) を指定します

クエリは効率と管理性を高めるためにページ分割されることに注意することが重要です。さらに、各リクエストテンプレートはパフォーマンスを最適化するためにキャッシュを使用できます。利用可能なパラメーターを使用して、ページ取得とキャッシュ実装を微調整します。短期間でサイズが小さい場合でも、キャッシュを推奨します。キャッシュにより、冗長なデータ取得が削減され、システム全体のパフォーマンスが大幅に向上します。

検索テンプレートは、特定のクエリタイプごとに適応するプレースホルダーをサポートします。次の4つの必須テンプレートがあります。

- 個々のユーザーを検索します
- ユーザーのグループを識別します
- グループ内のユーザーを検索します
- すべてのグループを取得します

さらに、すべてのユーザーを検索するためのオプションのテンプレートがあります。必須ではありませんが、より広範なユーザーアクセスにとって重要です。権限を割り当てる際にこれが構成されていない場合、選択はグループに限定されます。個々のユーザーは選択できません。

以下も参照してください。:

- TIBCO EBX® メイン構成プロパティの構成
- TIBCO EBX® Container Edition 環境変数の構成

制限事項

EBX®で LDAP を同期ディレクトリとして使用する場合、次の制限が適用されます。

- キャッシュメカニズムは、ページネーションのためにクッキーに保存されたセッション識別子に依存する LDAP サーバーと互換性がありません。そのような場合、リクエストキャッシュシステムはキャッシュの最大サイズを 0 に設定することで無効にする必要があります
- 一度に構成できる LDAP サーバーは1つだけです。同じインスタンス内で複数の LDAP サーバーへの同時接続はサポートされていません
- EBX®側では、ロールの包含や特定のロールの追加はサポートされていません。したがって、LDAP 内のグループの構造とネストされた関係は、権限の計算では考慮されません。この制限は、LDAP の階層的なグループダイナミクスの複雑さなしに、直接的なグループベースのアクセス制御アプローチを強調しています
- 単純で匿名の認証方法のみが許可されます。LDAP 内で使用される、より複雑または安全な認証プロトコルはサポートされていません
- ロールとユーザー名の表示はロケールに依存せず、構成ファイルの表示式によって定義された1つの形式に制限されます。ロケールは、表示式を適用できない場合にのみ考慮されます

SCIM ディレクトリ

SCIM (System for Cross-domain Identity Management) は、IT システム間でユーザー ID 情報の交換を自動化するように設計された標準プロトコルです。共通のスキーマと RESTful API を使用することで、複数のプラットフォームにわたるユーザープロビジョニングと管理を簡素化します。

使用されるスキーマバージョンは `urn:ietf:params:scim:schemas:core:2.0` です。

ロールの同期は、スケジュールされたタスク

`com.orchestranetworks.service.directory.scim.SchedulerScimSynchronization` によって管理されます。

構成

SCIM サーバーは IdP (Identity Provider) 側でアクティブ化する必要があります。

以下も参照してください。:

- TIBCO EBX® メイン構成プロパティの構成
- TIBCO EBX® Container Edition 環境変数の構成

ログイン時ディレクトリ

ログイン時ディレクトリは、SSO プロトコルからの情報に基づいて、ログイン時にユーザーとロールを同期する永続的なディレクトリです。その結果、ユーザーのリストは完全ではありません。一度もログインしたことのないユーザーは含まれません。

ロールマッピング

外部ユーザーのロールと内部ロールをマッピングするために、メイン EBX® 構成ファイルでロールマッパーを構成できます。

以下も参照してください。:

- 同期ディレクトリの構成

移行

バージョン 6.2.2 より前の LDAP および SCIM ディレクトリの移行を確実にするには、各ディレクトリのスケジュールされたタスクオプションでロールのインポートを有効にします。これにより、すでに永続化されている権限と外部ロールとの一貫性が保証されます。

データモデルの管理

- パブリケーションとバージョンの管理
- リポジトリ内の以前のデータモデルの移行
- スキーマの進化
 - データの永続性への影響
 - サイド機能への影響

パブリケーションとバージョンの管理

データモデルのパブリケーションとバージョンに関連する技術データには、管理者がセクションからアクセスできます。

データモデリングには次の 2 つのテーブルが含まれています。

- **パブリケーション。** リポジトリで利用可能なパブリケーションを保存します。
- **バージョン。** リポジトリで使用可能なデータモデルのバージョンを保存します。

これらのテーブルは読み取り専用ですが、パブリケーションまたはバージョンを手動で削除することは可能です。

****重要：****パブリケーションまたはバージョンが削除されると、関連するデータセットのコンテンツは利用できなくなります。したがって、この技術データは慎重に削除する必要があります。

EBX® リポジトリからアーカイブをエクスポートし、別の EBX® リポジトリにインポートすることで、この技術データを他の TIBCO EBX® リポジトリに広めることができます。これは、データモデルの進化を他のリポジトリに継承させるのに役立つ場合があります。

リポジトリ内の以前のデータモデルの移行

5.2.0 より前のバージョンでは、モジュールに依存しない公開されたデータモデルは、ファイルシステムディレクトリ `${ebx.repository.directory}/schemas/` に、データモデルの名前 (たとえば、データモデルの名前が *Product* の場合は *product.xsd*) で生成されました。バージョン 5.2.0 以降、この種のデータモデルは、EBX® 内で **パブリケーション** を通じて完全に管理されるようになりました。つまり、既存のデータモデルを再公開すると、それが **パブリケーション** として移行され、リンクされたデータセットが新しい埋め込みデータモデルにリダイレクトされます。 `${ebx.repository.directory}/schemas/` にある以前の XML スキーマドキュメントの名前が変更され、末尾に *toDelete* が付加されます。これは、ドキュメントが使用されなくなり、安全に削除できることを意味します。

スキーマの進化

管理側でデータモデルの変更の影響を評価することは非常に重要です。次の点を考慮する必要があります。

データの永続性への影響

管理タスクは、モデルの変更後のデータベースのクリーンアップに関係している場合があります。**データベース内マスターテーブルの削除** では、データモデルの進化が永続レベルでどのように管理されるかについて説明します。

サイド機能への影響

一部のコンポーネントはデータモデルに大きく依存しており、その進化によって影響を受ける可能性があります。例としては、ユーザーインターフェイス、WSDL ドキュメント、既存のアーカイブなどがあります。

セクションでは、これらのコンポーネントの一部 (ビューなど) を管理できますが、アーカイブ、WSDL ファイルなどの他のコンポーネントは管理者の範囲外となります。

データベースマッピング管理

- 概要
- データベースの列名の変更
- データベースの列の削除
- データベースのマスターテーブル名の変更
- データベースの補助テーブル名の変更
- データベースのマスターテーブルの削除

概要

データベースマッピングに関連する情報とサービスは、領域にあります。

以下も参照してください。:

- マップモード
- DatabaseMapping

データベースの列名の変更

この機能は、メニューの テーブルレコードで使用できます。これにより、データベースの列名を変更できます。

管理者は、マップモードのデータベース内のデータモデルの各列の名前を指定できます。

レコード上でサービスが選択されると、概要画面に選択した列に関する情報が表示され、管理者はデータベース内の列の新しい名前を入力するよう求められます。

注意:

新しい識別子は文字で始まる必要があります。

さらに、新しい名前は有効な列識別子にする必要があります、これは基礎となる RDBMS の命名規則によって異なります。

以下も参照してください。:

- `DatabaseMapping`

データベース内の列の削除

この機能は、メニューの テーブルレコードで使用できます。マップされた構造内の列を削除できます。

マップされたモードで無効になっている列は削除できます。

次の場合、マップモードで列が無効になります。

- 対応するフィールドがデータモデルから削除されている、または
- 対応するフィールドがデータモデル内で互換性のない方法で変更された (例: データ型が変更された)、または
- 定義されたマップモードは、エレメント `osd:history` および `osd:replication` を使用して、対応するフィールドでローカルに無効化されています。

以下も参照してください。:

- 特定のフィールドまたはグループの履歴の無効化
- 特定のフィールドまたはグループでのレプリケーションの無効化

集約されたリストではこの動作が変わることに注意してください。

- 複雑な集約リストを非アクティブ化すると、その内部フィールドは `LIVING` 状態のままになりますが、リストノードは無効になります。リストはマッピングシステムでは補助テーブルと見なされるため、この情報は「テーブル」テーブルで確認できます。

- 一方、リストの内部ノードのみを非アクティブ化する場合、リストは **LIVING** のままになりますが、その子は **DISABLED IN MODEL** になります。

列は、その列自体の状態が **DISABLED IN MODEL** の場合、または **DISABLED IN MODEL** リストの内部フィールドである場合にのみ削除できます。

データベースのマスターテーブル名の変更

この機能を使用すると、データベース内の履歴テーブルのマスターテーブルの名前を変更できます。複製されたテーブルの名前はデータモデルで指定されるため、この機能は使用できません。

両方の機能は、メニューの テーブルレコードで利用できます。

マスターテーブルは、データモデルのテーブルを永続化するために使用されるデータベーステーブルです。

管理者は、データモデルのテーブルに対応する各マスターテーブルの名前をデータベースで指定できます。

レコード上でサービスが選択されると、概要画面に選択したマスターテーブルに関する情報が表示され、管理者はデータベース内のマスターテーブルの新しい名前を入力するよう求められます。

注意:

新しい識別子は文字とリポジトリプレフィックスで始まる必要があります。

履歴テーブルの場合、リポジトリプレフィックスの後に履歴テーブルプレフィックスが続くことも必要です。

さらに、新しい名前は有効なテーブル識別子にする必要があります。これは、基盤となる RDBMS の命名規則によって異なります。

データベース補助テーブル名の変更

この機能により、データベース内の履歴補助テーブルの名前を変更できます。名前がデータモデルで指定されているため、この機能はレプリケートされたテーブルでは使用できません。

この機能は、メニューの テーブルレコードで使用できます。

補助テーブルは、集約リストを永続化するために使用されるデータベーステーブルです。

管理者は、データモデルの集約リストに対応する各補助テーブルの名前をデータベースで指定できます。

レコードでサービスが選択されると、要約画面に選択された補助テーブルに関する情報が表示され、管理者はデータベース内の補助テーブルの新しい名前を入力するように求められます。

注意:

新しい識別子は文字で始まる必要があります。

新しい識別子はリポジトリプレフィックスで始まる必要があります。

また、リポジトリプレフィックスの後に履歴テーブルプレフィックスが続く必要があります。

さらに、新しい名前は有効なテーブル識別子にする必要があります。これは、基盤となる RDBMS の命名規則によって異なります。

データベース内マスターテーブルの削除

この機能を使用すると、データベースが使用されなくなった場合にデータベース内の履歴を削除できます。

これは、メニューの テーブルレコード で使用でき、マスターテーブルでのみ使用できます。この機能は、マスターテーブルにのみ適用されます。マスターテーブルがパージされると、そのすべての補助テーブルもパージされます。

マップされたテーブルは、対応するマップされたモードで無効にされている場合にのみ、データベースでパージできます。

テーブルのマップモードを無効にするには、以下の手順に従います。

- データモデル内のテーブルの履歴化を無効にする、または
- データモデルからテーブルを削除する。

外部サービス設定

- [設定テーブル](#)

- 認証タイプ
- キーストアファイル
 - はじめに
 - キーストア機密プロパティファイル
 - 自動インストール
 - ステージング

設定テーブル

管理 > 統合 > 外部サービス > 設定テーブルで外部サービスを設定します。次のプロパティを入力します。

プロパティ	定義
識別子	サービス設定の一意の識別子
URL	呼び出すサーバーのベースURLを指定します。クライアントとサーバー間の通信を暗号化するにはHTTPSを使用します。 URLの形式は次のとおりです。 <pre>http[s]://<host>[:<port>]/<path>{?<queryParameters>}</pre> 各要素は次のとおりです。 • <code>host</code> と <code>port</code>: APIが存在するサーバーアドレスを定義します • <code>path</code>: API内の特定のリソースのパス • <code>queryParameters</code>: リソースをフィルタリングまたは絞り込むために使用されるオプションのパラメーター
使用タイプ	外部サービスの使用タイプ (Assistant AI および/または REST サービス) を定義します。

プロパティ	定義
認証タイプ	APIリクエストを行う際のユーザー認証を定義します。APIとそのリソースにアクセスするには、basic、custom、EBX、noneのタイプを使用します。

認証タイプ

認証により、許可されたユーザーのみがサービスとそのリソースにアクセスできるようになります。次の認証タイプがサポートされています。

タイプ	説明
Basic	有効なユーザー名とパスワードを必要とする、シンプルで高速なHTTP認証方法です。資格情報はBase64でエンコードされ、HTTPリクエストヘッダーに含まれます。パスワードは、強力な暗号化によりEBX®リポジトリに安全に保存されます。詳細については、「 キーストアファイル 」を参照してください。
EBX	EBX組み込みRESTful サービスへのアクセスを有効にします。有効なユーザー名とパスワードを提供する必要があります。パスワードは、強力な暗号化によりEBX®に安全に保存されます。詳細については、「 キーストアファイル 」を参照してください。
Custom	有効なキー名とキー値を提供することにより、APIキー、トークン、OAuthなどのカスタム認証を有効にします。キーの値は機密である場合とそうでない場合があります。すべてのキーは、機密であるかどうかにかかわらず、異なる名前を持つ必要があります。そうでない場合、検証エラーが表示されます。秘密鍵は、強力な暗号化によりEBX®リポジトリに安全に保存されます。詳細については、「 キーストアファイル 」を参照してください。
None	認証は不要です。

キーストアファイル

はじめに

EBX®は、データの暗号化または署名に使用される秘密鍵を保存するためにキーストアファイルを使用します。このファイルの場所と関連する詳細は、*キーストア機密プロパティファイル*によって指定されます。

キーストア機密プロパティファイル

機密プロパティは、キーストアファイルの場所、および秘密鍵のエイリアスやパスワードなどのその他の関連情報を指定します。

機密プロパティは、`ebx.properties` ファイルとは別のプロパティファイルに保存します。デフォルトでは、このファイルは `ebx-keystore-sensitive.properties` という名前で、EBX® リポジトリと同じディレクトリに保存されます。`ebx.properties` ファイルでプロパティ `ebx.keystore.sensitive.properties.path` を設定することにより、場所を変更できます。

このプロパティファイルの形式は次のとおりです。

```
## 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/u1HdPqaeC6/
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/
u1HdPqaeC6/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/u1HdPqaeC6/
MMMHLyA11XvQZ6/rFqVbmjDZgDL5GPRcDc1+OQQpw6Fg==

```

自動インストール

初回起動時に、EBX®は `ebx-keystore.p12` ファイルと `ebx-keystore-sensitive.properties` ファイルをEBX®リポジトリフォルダーに作成します。秘密鍵は自動的に生成され、インストールに固有のものです。

ステージング

デフォルトでは、EBX®ステージングサーバーは異なる秘密鍵を持っています。あるステージングサーバーによって暗号化されたデータは、別のサーバーでは復号化できません。この問題を回避し、すべてのEBX®ステージングサーバー間で鍵を整合させるには、次の手順を実行します。

- 同じキーストアファイル (デフォルトでは `ebx-keystore.p12`) をコピーします
- キーストア機密プロパティファイル (デフォルトでは `ebx-keystore-sensitive.properties`) をすべてのEBX®ステージングサーバーインストールにコピーします

ワークフロー管理

- ワークフロー
 - ワークフローの実行
 - ワークフローの公開
 - 構成
- インタラクション
- ワークフロー履歴
 - 履歴のクリーンアップ

ワークフロー

データワークフローの実行、ワークフロー公開の管理、または進行中のデータワークフローを監視するための一般的なパラメーターを定義するには、領域に移動します。ナビゲーションペインの下矢印をクリックして、`* > *`を選択します。

注意:

ワークフロー実行テクニカルテーブルで予期しない不整合が発生した場合、データワークフローでエラーが発生する可能性があります。次に、ナビゲーションペインの`* > *`の下にあるメニューから 操作を実行する必要がある場合があります。

ワークフローの実行

さまざまなテーブルを使用して、現在進行中のデータワークフローを管理できます。これらのテーブルには、ナビゲーションペインの `>` からアクセスできます。

以下も参照してください。:

- データワークフローの管理

ワークフローテーブル

テーブルには、サブワークフローとして呼び出されたものを含む、リポジトリ内のすべてのデータワークフローのインスタンスが含まれています。データワークフローは、ワークフローモデル公開の特定の実行インスタンスです。このテーブルは、すべてのデータワークフローのデータコンテキスト変数へのアクセスを提供します。これを使用して、現在の変数値に関するデータワークフローの進行状況にアクセスし、データワークフローの一時停止の場合は、変数値を変更できます。

テーブルのメニューから、サービスを選択することにより、指定された日付より古い完了したデータワークフローをクリアすることができます。このサービスは、アクティブなデータワークフローを自動的に無視します。

トークンテーブル

テーブルを使用すると、データワークフローの進行状況を管理できます。各トークンは、実行中のデータワークフローで実行されている現在のステップと、データワークフローの現在の状態をマークします。

以下も参照してください。:

- トークン

作業項目テーブル

テーブルには、現在存在するユーザータスクに関連付けられているすべての作業項目が含まれています。必要に応じて、データワークフローがブロックされた場合に、このテーブルからユーザーに作業項目を手動で割り当てることができます。ただし、可能な場合は常に、領域のワークスペースにあるボタンを使用して、作業項目の割り当て、再割り当て、および割り当て解除を行うことをお勧めします。

以下も参照してください。:

- 作業項目

待機中のワークフローテーブル

テーブルには、イベントを待機しているすべてのワークフローが含まれています。必要に応じて、このテーブルをクリーンアップするサービスを利用できます。このサービスは、削除されたワークフローに関連付けられているすべての行を削除します。

以下も参照してください。:

- 待機タスク

コメントテーブル

テーブルには、メインワークフローとそのサブワークフローに対するユーザーのコメントが含まれています。

ワークフローの公開

テーブルは、リポジトリのすべてのワークフローモデルパブリケーションを含む技術的なテーブルです。このテーブルは、公開されたワークフローモデルとそのスナップショットに関連付けます。このテーブルを直接変更することは推奨されません。むしろ、パブリケーションに変更を加えるためにワークフローモデリングエリアで利用可能なアクションを使用することをお勧めします。

構成

メール構成

データワークフローの実行中に電子メール通知を送信するには、で次の設定を構成する必要があります。

- フィールドは、ワークフローでリンクを構築し、メール変数を評価するために使用されます。
- フィールドには、メール通知の送信に使用されるメールアドレスを入力する必要があります。

インターフェイスのカスタマイズ

モデリングのデフォルト値

一部のプロパティのデフォルト値は、このセクションでカスタマイズできます。

管理者は、セクションで、新しいワークフローモデルまたはワークフローステップが作成されるときに使用されるデフォルト値を定義することができます。

作業項目ビュー

特定の列は、受信トレイと監視作業項目テーブルのセクションで使用できます。

10 個の特定の列が利用可能です。特定の列ごとに、カスタマイズされたラベルを定義できます。

優先度の構成

プロパティ「優先度レベル」は、優先度レベルがない場合に、リポジトリ全体のデータワークフローとその作業項目をどのように表示するかを定義します。たとえば、このプロパティが値に設定されている場合、優先度のないワークフローと作業項目はすべて「標準」の優先度を持っているように見えます。

テーブルは、リポジトリ内のデータワークフローで使用可能なすべての優先度レベルを定義します。ユーザーが作業項目テーブルの優先度アイコンにカーソルを合わせると表示されるラベルとともに、必要な数の整数の優先度レベルを追加できます。TIBCO EBX® が提供するセットから、またはアイコン画像ファイルへの URL を指定することにより、各優先度レベルに対応するアイコンを選択することもできます。

一時タスク

で、ワークフロー内の時間依存タスクのポーリング間隔 (期限やリマインダーなど) を設定できます。間隔値が設定されていない場合、「進行中」のステップが 1 時間ごとにチェックされます。

ワークフローの受信トレイカウンター構成

エンドユーザーがアクションを起動しなくても、ワークフローの受信トレイカウンターは非同期で更新されます。調整するには、次の 2 つのパラメーターを設定する必要があります。

キャッシュの有効期限 (秒)

受信トレイキャッシュが新しく更新されるまでの有効期限 (秒単位)。多くの作業項目があるリポジトリでは計算時間が長くなる可能性があるため、このパラメーターは CPU

	の負荷とパフォーマンスに影響を与える可能性があることに注意してください。値が定義されていない場合、デフォルト値は 600 です。
ユーザーインターフェイスの更新周期 (秒)	ユーザーインターフェイスの受信トレイカウンター の 2 つの更新間の更新時間 (秒単位)。この更新は、ユーザーインターフェイスのすべての受信トレイカウンター (カスタムパースペクティブの受信トレイカウンター、ヘッダー受信トレイカウンター、および詳細パースペクティブのデータワークフロー受信トレイカウンター) に関係することに注意してください。値が定義されていない場合、デフォルト値は 5 です。値がゼロ (または負) の場合、更新は無効になります。また、変更はユーザーからのログアウト/ログイン後にのみ有効になります。

また、一部のアクションにより、受信トレイカウンターが強制的に更新される可能性があることに注意してください。

- データワークフロー へのアクセス
- データワークフローセクション の任意のサブディビジョンへのアクセス
- 作業項目の承諾、または拒否
- ワークフローの起動

これらのパラメーターには、ナビゲーションペインの >>> からアクセスできます。

インタラクション

ワークフローの相互作用を管理するには、領域に移動します。ナビゲーションペインの下矢印をクリックして、エントリ > を選択します。

相互作用 は、作成されるすべての作業項目に対して自動的に生成されます。これは作業項目のローカルデータコンテキストであり、EBX® セッションからアクセスできます。作業項目が実行されると、ユーザーはワークフローエンジンとは関係なく、その相互作用に基づいて割り当てられ

たアクションを実行します。ユーザータスクは、入力パラメーターと出力パラメーターのマッピングを定義して、データワークフローの全体的なデータコンテキストとの相互作用をリンクします。

インタラクションは、作業項目の現在のパラメーターを監視するのに役立ちます。

たとえば、インタラクションは、トリガーまたはユーザーサービスによって手動で更新できます。

ワークフロー履歴

履歴データワークフローの実行を表示するには、領域を参照します。ナビゲーションペインの下矢印をクリックして、> を選択します。

「ワークフロー」テーブルには、ワークフローの実行中に実行されたすべてのアクションが含まれています。

このデータは、グラフィカルまたはテキストで表示できます。特定の時点でのワークフローに関連するさまざまなオブジェクトの状態を表示すると特に便利です。これには、作業項目、データコンテキストの変数、およびトークンに対するアクションが含まれます。エラーが発生した場合は、テクニカルログを利用できます。

履歴のクリーンアップ

「ワークフロー」テーブルのメニューから、サービスを選択することにより、特定の日付より古い完了したデータワークフローの履歴をクリアできます。

以前にクリーンアップされた (実行データが削除されたなど) ワークフローの履歴のみがクリアされます。このサービスは、既存のワークフローに関連付けられている履歴を自動的に無視します。

「ワークフロー」テーブルの専用サービスを使用して、関連する履歴をクリアする前にデータワークフローをクリアする必要があります。また、スケジュールされた は、ビルトインのスケジュールされたタスク `SchedulerPurgeWorkflowMainHistory` で使用できます。

メインプロセスのみがクリーンアップされることに注意してください。サブプロセスとすべての関連データを削除するには、「標準の EBX® パージ」を実行する必要があります。

以下も参照してください。:

- [ワークフロー履歴のクリーンアップ方法](#)

注意:

ワークフローの履歴を取得するための API を利用できます。基になるワークフロー履歴 SQL テーブルへの直接アクセスはサポートされていません。

[WorkflowEngine.getProcessInstanceHistory](#) を参照してください。

タスクスケジューラ

- 概要
- [EBX® からの構成](#)
- cron 式
 - 形式
 - 特殊文字
 - 例
- タスク定義
- タスク構成

概要

TIBCO EBX® は、プログラムによるタスクスケジューリング機能を提供します。

注意:

競合やデッドロックを回避するために、タスクは単一のキューでスケジューリングされます。

EBX® からの構成

スケジューリングとタスクの宣言は、領域で を選択することによって行われます。

- **スケジューリング**：「cron 式」を使用してスケジューリングを定義します。
- **タスク**：タスクインスタンスとその実行用のユーザープロファイルのパラメーター化など、タスクを構成します。
- **スケジューリングされたタスク**：タスクスケジューリングのアクティブ化/非アクティブ化を含む現在のスケジューリング。
- **実行レポート**：タスクがトリガーされた直後に表示される、スケジューリングされた各タスク実行のレポート。レポートには、タスク定義で使用可能になったときに、実行中のタスクを中断、一時停止、または再開するアクションが含まれます。

cron 式

(Quartz スケジューラドキュメントの抜粋)

タスクスケジューラは「cron 式」を使用して、「毎週月曜日から金曜日の午前 8 時」や「毎月最終金曜日の午前 1 時 30 分」などの起動スケジュールを作成できます。

形式

cron 式は、空白で区切られた 6 つまたは 7 つのフィールドで構成される文字列です。フィールドには、そのフィールドに許可されている特殊文字のさまざまな組み合わせとともに、許可されている任意の値を含めることができます。フィールドは次のとおりです。

フィールド名	必須	許容値	許可されている特殊文字
Seconds	はい	0-59	, - * /
Minutes	はい	0-59	, - * /
Hours	はい	0-23	, - * /
Day of month	はい	0-31	, - * ? / L W
Month	はい	1-12 または JAN-DEC	, - * /
Day of week	はい	1-7 または SUN-SAT	, - * ? / L #
Year	いいえ	空、1970-2099	, - * /

cron 式の単純なフォーマット: **"0 * * * * ?"**

または、より複雑なフォーマット: **"0/5 14,18,3-39,52* ? JAN、MAR、SEP MON-FRI 2002-2010"**

注意:

有効な文字と月と曜日の名前は大文字と小文字を区別しません。MON は mon と同じです。

特殊文字

cron 式は、空白で区切られた 6 つまたは 7 つのフィールドで構成される文字列です。フィールドには、そのフィールドに許可されている特殊文字のさまざまな組み合わせとともに、許可されている任意の値を含めることができます。フィールドは次のとおりです。

- * ("すべての値") - フィールド内のすべての値を選択するために使用されます。たとえば、フィールドの「*」は「毎分」を意味します。
- ? ("特定の値なし") - 文字が許可されている 2 つのフィールドの一方に何かを指定する必要があるが、もう一方には指定する必要がない場合に役立ちます。たとえば、トリガーを特定の日付 (たとえば、10 日) に起動したいが、曜日が何であるかは気にしない場合は、日付フィールドに「10」、曜日フィールドに「?」を入力します。説明については、以下の例を参照してください。
- -- 範囲を指定するために使用されます。たとえば、時間フィールドの「10-12」は「時間 10、11、および 12」を意味します。
- , - 追加の値を指定するために使用されます。たとえば、曜日フィールドの「MON,WED,FRI」は「月曜日、水曜日、金曜日」を意味します。
- / - 増分を指定するために使用されます。たとえば、秒フィールドの「0/15」は「秒 0、15、30、および 45」を意味します。また、秒フィールドの「5/15」は「秒 5、20、35、および 50」を意味します。「*」文字の後に「/」を指定することもできます。この場合、「*」は、「/」の前に「0」を付けることと同じです。日フィールドの「1/3」は、「月の初日から 3 日ごとに起動する」ことを意味します。
- L ("last") - 許可されている 2 つのフィールドのそれぞれで異なる意味を持ちます。たとえば、day-of-month フィールドの値「L」は「月の最後の日」を意味します。うるう年以外の場合は 1 月の 31 日目、2 月の 28 日目です。曜日フィールドで単独で使用される場合、それは単に「7」または「SAT」を意味します。ただし、曜日フィールドで別の値の後に使用する場合は、「その月の最後の xxx 日」を意味します。たとえば、「6L」は「その月の最後の金曜日」を意味します。「L」オプションを使用する場合、混乱する結果が得られるため、リストまたは値の範囲を指定しないことが重要です。
- W ("weekday") - 指定された日に最も近い平日 (月曜日から金曜日) を指定するために使用されます。たとえば、day-of-month フィールドの値として「15W」を指定した場合、その意味は「その月の 15 日に最も近い平日」です。したがって、15 日が土曜日の場合、トリガーは 14 日金曜日に起動します。15 日が日曜日の場合、トリガーは 16 日月曜日に起動します。15 日が火曜日の場合、15 日火曜日に起動します。ただし、曜日の値として「1W」を指定し、1 日が土曜日の場合、トリガーは 3 日月曜日に起動します。これは、月の日の境界を超えて

「ジャンプ」しないためです。「W」文字は、曜日が 1 日であり、範囲や日のリストではない場合にのみ指定できます。

注意:

「L」文字と「W」文字を日フィールドで組み合わせて「LW」を生成することもできます。これは「月の最後の平日」に変換されます。

- # - 月の「n 番目」の曜日を指定するために使用されます。たとえば、曜日フィールドの「6#3」の値は、「その月の第 3 金曜日」を意味します (6 日目=金曜日、「#3」=月の第3金曜日)。その他の例: 「2#1」=その月の第1月曜日、「4#5」=その月の第 5 水曜日。「#5」を指定し、その月に指定された曜日が 5 つない場合、その月にはトリガーが起動しないことに注意してください。

例

式	定義
0 0 12** ?	毎日午後 12 時 (正午) に起動します。
0 15 10 ? **	毎日午前 10 時 15 分に起動します。
0 15 10** ?	毎日午前 10 時 15 分に起動します。
0 15 10** ? *	毎日午前 10 時 15 分に起動します。
0 15 10 * * ? 2005	2005 年中は毎日午前 10 時 15 分に起動します。
0*14** ?	毎日午後 2 時から午後 2 時 59 分まで毎分起動します。
0 0/5 14** ?	毎日午後 2 時から午後 2 時 55 分まで 5 分ごとに起動します。
0 0/5 14,18** ?	毎日午後 2 時から午後 2 時 55 分まで 5 分ごとに起動し、午後 6 時から午後 6 時 55 分まで 5 分ごとに起動します。
0 0-5 14** ?	毎日午後 2 時から午後 2 時 5 分まで毎分起動します。

式	定義
0 10,44 14 ? 3 WED	3 月の毎週水曜日の午後 2 時 10 分と午後 2 時 44 分に起動します。
0 15 10 ? * MON-FRI	毎週月曜日、火曜日、水曜日、木曜日、金曜日の午前 10 時 15 分に起動します。
0 15 10 15* ?	毎月 15 日の午前 10 時 15 分に起動します。
0 15 10 L* ?	毎月最終日の午前 10 時 15 分に起動します。
0 15 10 ? *6L	毎月最終金曜日の午前 10 時 15 分に起動します。
0 15 10 ? *6L 2002-2005	2002 年、2003 年、2004 年、および 2005 年の間、毎月最終金曜日の午前 10 時 15 分に起動します。
0 15 10 ? *6#3	毎月第 3 金曜日の午前 10 時 15 分に起動します。
0 0 12 1/5* ?	毎月 1 日から 5 日おきの午後 12 時 (正午) に起動します。
0 11 11 11 11 ?	毎年 11 月 11 日午前 11 時 11 分に起動します。

注意:

曜日および日付フィールドの「?」および「*」の影響に注意してください。

曜日と日付の両方の値を指定するためのサポートは完全ではありません (現在、これらのフィールドの 1 つで「?」文字を使用する必要があります)。

居住する地域で「夏時間」の変更が発生する朝の時間帯に起動時間を設定する場合は注意が必要です (米国の地域では、通常、午前 2 時の前後の時間になります。時間のずれによりスキップまたは時間が戻るか進むかに応じて繰り返します)。

タスク定義

EBX® スケジューラには、次の事前定義されたタスクが含まれています。

- 期限切れの一時ステージングフォルダーの削除
- リポジトリのクリーンアップ (パージ)
- 古いデータスペースの削除
- データスペース履歴の削除
- スケジュールされたタスクの実行レポートのパージ
- ワークフロープロセス履歴のパージ
- ワークフローの起動

カスタムのスケジュールされたタスクは、[スケジューラ](#) Java API を使用して追加できます。

スケジュールとタスクは、ユーザーインターフェースの >> で宣言および構成します。

タスク構成

ユーザーはタスク定義に関連付けられている必要があります。このユーザーは、タスクを実行する [セッション](#) を生成するために使用されます。

注意:

ユーザーは認証されず、パスワードは必要ありません。結果として、ディレクトリにパスワードが設定されていないユーザーは、スケジュールされたタスクの実行にのみ使用できます。

カスタムタスクは、JavaBean 仕様 (ゲッターとセッター) を使用してパラメーター化できます。

サポートされているパラメータータイプは次のとおりです。

- 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

パラメーター値は XML 形式で設定されます。

バックグラウンドタスク

- [概要](#)
- [バックグラウンドタスクの設定](#)
- [バックグラウンドタスクの概要](#)
 - [タスクのプロパティ](#)
 - [サイドバービュー](#)

概要

EBX® には、リポジトリに非常に大量のデータを保存する機能があります。高性能システムであっても、大量のデータのインポート、エクスポート、または転送などの一部のアクションは、完了までに長い時間がかかる場合があります。ユーザーが作業を再開する前にこれらのタスクが完了するのを待つ必要がある場合、生産性に悪影響を与える可能性があります。これらのタスクをバックグラウンドで実行するように設定することで、ユーザーは EBX® で他のアクティビティを実行できるようになります。

バックグラウンドタスクは、ユーザーインターフェースのサイドバーに一元化されています。このインターフェースから、ユーザーは次のことができます。

- 実行中のタスクの進行状況を表示
- 実行中のタスクを中断
- 終了したタスクのレポートを表示。タスクは、ユーザーがビューから削除するまで表示されたままになります

バックグラウンドタスクの設定

現在、TIBCO EBX® Data Exchange (New) アドオンでは、そのインポート、エクスポート、および転送サービスをバックグラウンドで実行するように設定できます。設定手順については、このアドオンのドキュメントを参照してください。

バックグラウンドタスクの概要

以下のセクションでは、バックグラウンドタスクの概要を説明します。

- [タスクのプロパティ](#)
- [サイドバービュー](#)

タスクのプロパティ

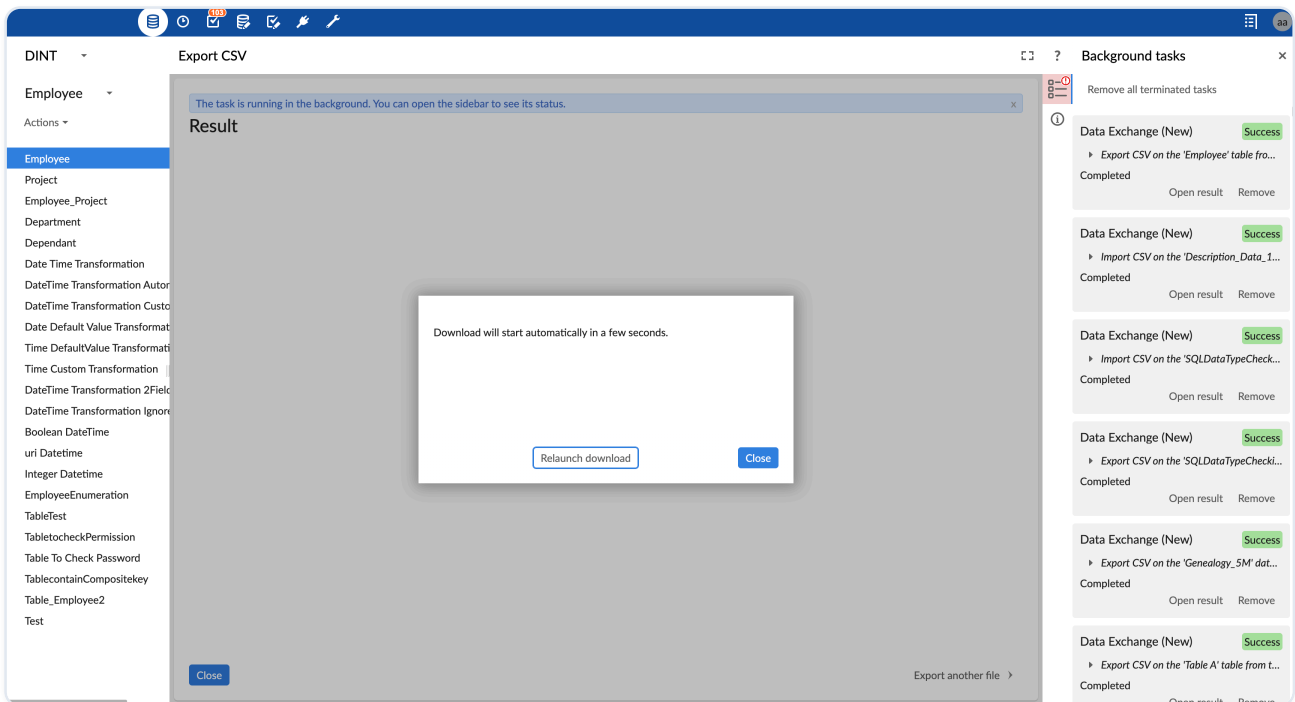
タスクには、次のプロパティが含まれます。

- 一意のID番号
- 提出日
- 終了日 (タスクが終了した場合)
- ローカライズされた説明
- 所有者 (タスクを提出したユーザー)
- 状態 (以下のいずれか)
 - 保留中: タスクは提出されましたが、実行はまだ開始されていません
 - 実行中: タスクは現在実行中です。進行状況情報が利用できる場合があります
 - 中断中: タスクは中断要求を受け取りましたが、まだ実行中です
 - 中断済み: タスクは中断されたため停止しました
 - 成功: タスクはエラーなしで停止しました
 - 警告: タスクは警告とともに停止しました。ローカライズされた警告メッセージが利用できます
 - 失敗: タスクは失敗により停止しました。ローカライズされたエラーメッセージが利用できます

サイドバービュー

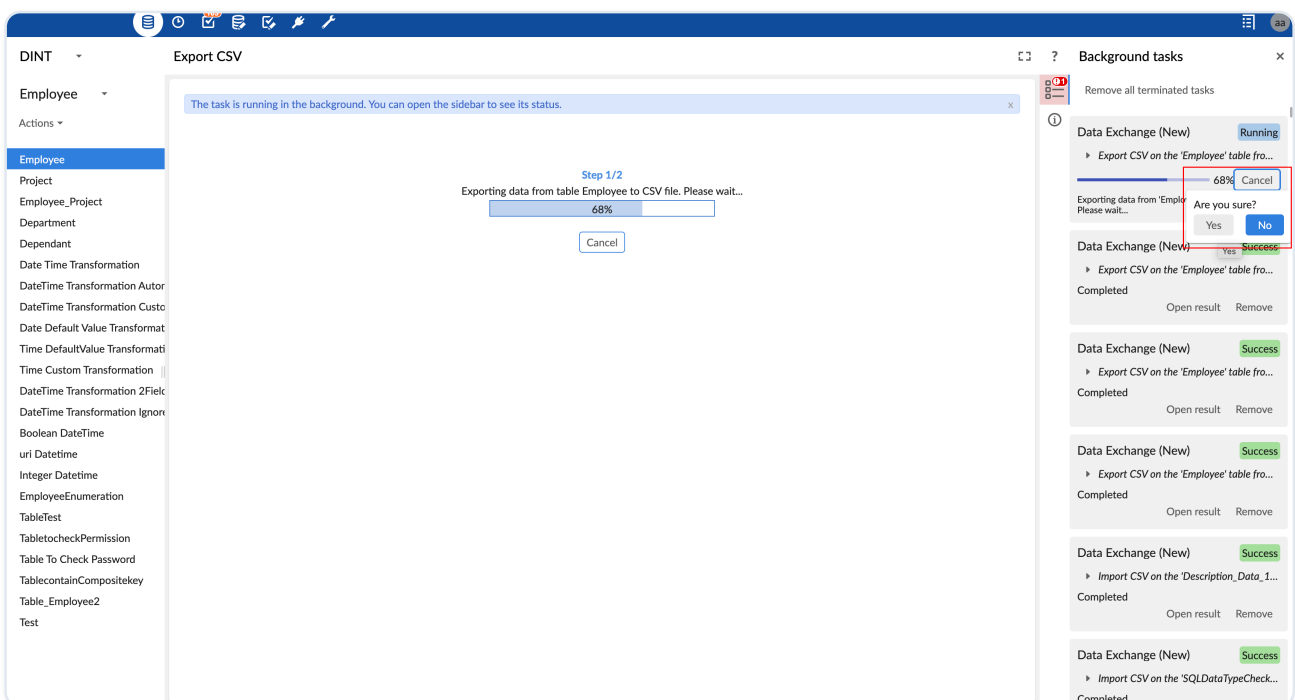
サイドバービューには、ユーザーが表示を許可されているすべてのタスクが表示されます。タスクプロバイダーは、エンドユーザーに表示する進行状況情報の2つの形式を実装できます。

- 進捗率 (0から100までの数値) がプログレスバーとして表示されます
- 短いローカライズされたメッセージ (「ステップ1」、「ステップ2」など)



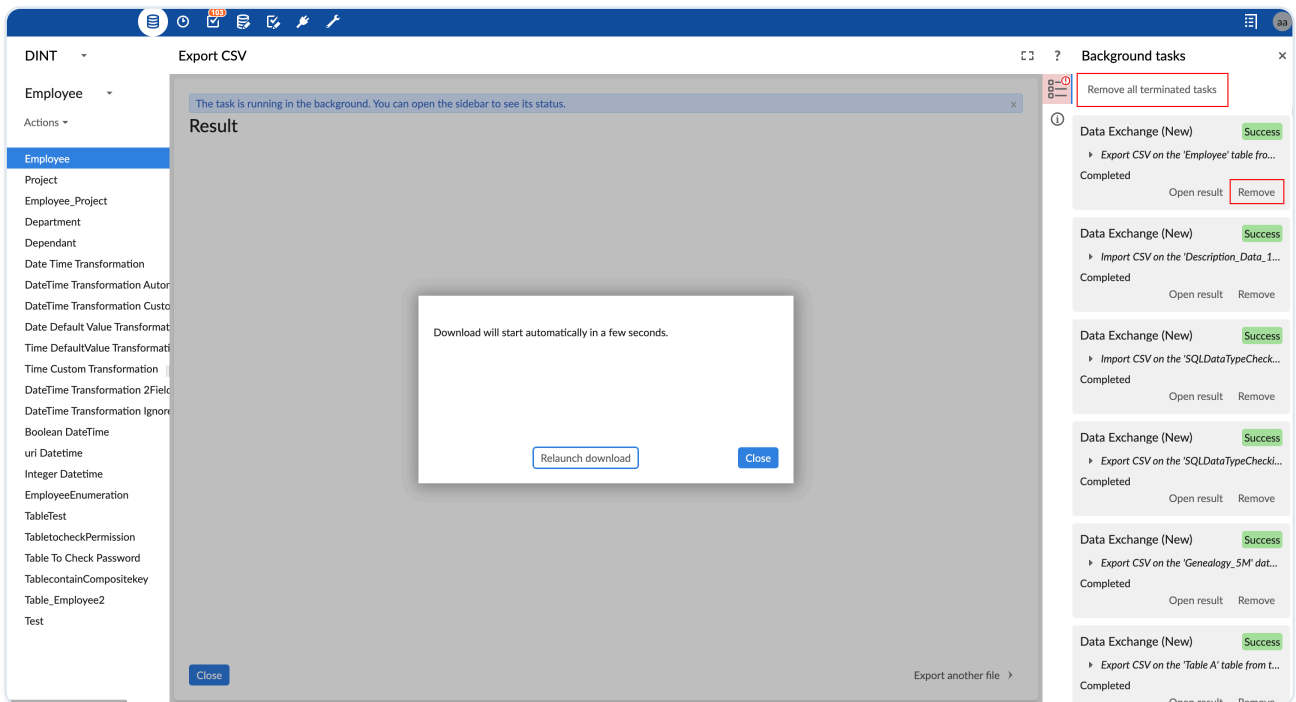
実行中のタスクのキャンセル

ユーザーは、タスクプロバイダーによって実装されている場合にのみタスクをキャンセルできます (一部のタスクは中断できない場合があります)。



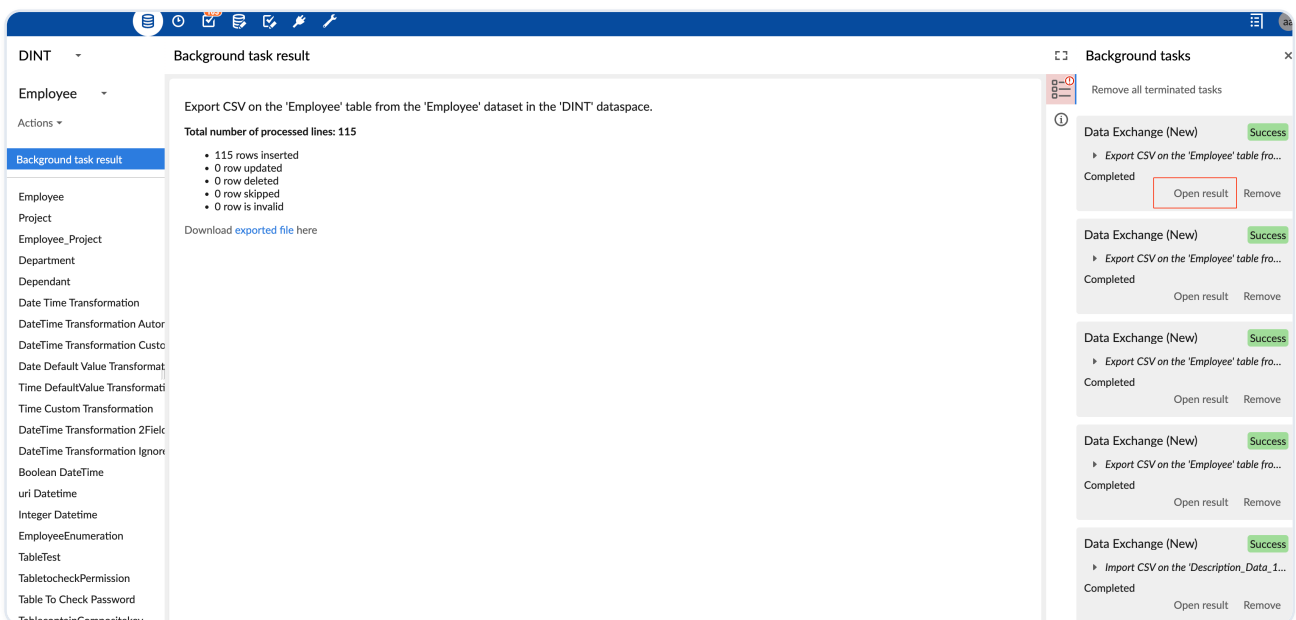
終了したタスクの削除

終了したタスクは、削除またはすべての終了したタスクを削除ボタンを使用して、いつでもサイドバーから削除できます。



タスクレポート

タスクレポートはオプションであり、タスクのタイプに固有です。「成功」、「警告」、「エラー」、または「中断済み」のステータスで利用できます。レポートはバックグラウンドタスクプロバイダーによって実装されます。レポートが利用可能な場合、ユーザーはリンクからレポートに移動できます。



監査証跡

- 概要
- ログコンテンツ

- 共通ヘッダー
- テーブルビューイベント
- ユーザーサービス
- レガシー XML 監査証跡 (非推奨)

概要

監査証跡機能を使用すると、TIBCO EBX® でのすべてのビューアクセスとユーザーサービスの起動をログに記録できます。

この機能はデフォルトで無効になっています。監査証跡ログを有効にするには、**INFO** メッセージを有効にするレベルで **audit** ログカテゴリを構成します。

以下も参照してください。:

- 監査証跡のアクティブ化

ログコンテンツ

監査証跡によってログに記録される情報は、次のセクションで説明する共通ヘッダーとイベント固有のコンテンツの 2 つの部分に分かれています。

共通ヘッダー

常にログに記録されるヘッダー属性は次のとおりです。

user-login	ユーザーのログイン
user-ip	ユーザーの IP アドレス (利用可能な場合)
event	次の値のうち、ユーザーが実行したイベントのタイプ。 • TABULAR_ACCESS : UI で表形式のビューにアクセスする場合

- HIERARCHY_EXPAND_NODE：ノードが階層ビューで展開されたとき、またはビューが表示された場合
- HIERARCHY_EXPAND_ALL：複数のノードが階層ビューで展開されている場合
- DATASPACE_PERMISSION_TABULAR_ACCESS：データスペースの権限テーブルにアクセスした場合
- DATASET_PERMISSION_TABULAR_ACCESS：データセットの権限テーブルにアクセスした場合
- USER_SERVICE：エクスポート、比較、検証のビルトインサービス、アドオンまたはカスタム ユーザーサービスが起動された場合

テーブルビューイベント

このセクションでは、テーブルビューに固有のイベントについて説明します。

- TABULAR_ACCESS
- HIERARCHY_EXPAND_NODE
- HIERARCHY_EXPAND_ALL
- DATASPACE_PERMISSION_TABULAR_ACCESS
- DATASET_PERMISSION_TABULAR_ACCESS

これらのイベントは、テーブルビューが開かれたとき、更新されたとき、検索が実行されたとき、ユーザーがページネーションを使用してナビゲートしたときにトリガーされます。

これらのイベントに固有のコンテンツ属性は次のとおりです。

dataspace	テーブルを含むデータスペースのキー
dataset	テーブルを含むデータセットのキー

table	アクセスされているテーブルのパス (権限イベントには適用されません)
view-id	現在適用されているカスタムビューの識別子 (権限イベントには適用されません)
view-name	現在適用されているカスタムビューの公開名 (権限イベントには適用されません)
pagination-info	表形式ビューまたは展開されたノードのページネーションに関する情報
hierarchy-expanded-node	展開されるノードの識別子 (階層イベントにのみ適用されます)。 複数のレベルが展開される (すべて検索または展開) 場合、上位のノードです。識別子は、プレフィックス「:」付きのノードラベルです。ルートノードは「:」として識別されます。 階層が最初に表示されるとき、ルートノードのみが展開されるため、イベントは HIERARCHY_EXPAND_NODE になります。
filters	ビューに適用され、JSON 形式で集約されたすべてのフィルター。フィルターには次の 3 種類があります。 <code>quick-search</code> : クイック検索フィールドへのユーザー入力 (存在する場合) <code>structured-search</code> : JSON 形式の高度な検索パネルを使用して構成されたユーザーの高度なフィルター (存在する場合) <code>view-filter</code> : XPath 形式のビューで定義されたフィルター (存在する場合)

例

```
filters={"structured-search":["starts-with","/fi
```

ユーザーサービス

このイベントに固有のコンテンツ属性は次のとおりです。

service-key	実行中のサービスのサービスキー
service-mode	サービスは、選択エンティティの任意の中間レベルで実行できます。たとえば、現在の選択エンティティがレコードの場合、データスペース、データセット、テーブル、またはレコードのレベルでサービスを実行できます。この属性により、サービスが実行される選択エンティティのレベルを定義できます。
selection	JSON 形式でサービスが実行されるエンティティの選択 (ブランチまたはバージョン、データセットなど)。100 を超えるレコードが選択された場合、最初の 100 個のレコードのみがログに記録されます。
service-parameters	サービス実行用に指定された追加の入力パラメーター - ビルトインサービスについては、パラメータのリストを参照してください - カスタムサービスの場合、パラメータは宣言で指定されます - アドオンサービスについては、アドオンのドキュメントを参照してください

レガシー XML 監査証跡 (非推奨)

XML 監査証跡は、データの更新を XML ファイルに記録します。現在非推奨となっているこのレガシー機能は、将来のバージョンで削除される予定です。このタイプの監査証跡は使用されなくなりましたが、[レガシー XML 監査証跡](#)は引き続き利用できます。

コラボレーション機能 (チームアップ)

- [はじめに](#)
- [チームアップの概要](#)
 - [チームアップへのテーブルの登録](#)
 - [ユーザーアクセスとチームアップ機能の管理](#)
- [タグとタグカタログの管理](#)

はじめに

チームアップを有効にして、ビジネスユーザー間のコラボレーションと連携を強化できます。チームアップを使用すると、レコードにコメントを残したり、レコードを評価したり、タグをレコードに割り当てたりできます。チームアップ機能へのアクセスと可視性は高度にカスタマイズ可能です。グローバル設定はデータモデルレベルで使用できますが、データセットレベルでオーバーライドできます。

チームアップ機能での管理者の役割の概要は、次のとおりです。

- **コメント：**レコードへのコメントを有効にします。他のユーザープロフィールからのコメントを管理できるテーブルにモデレーターを割り当てます。
- **タグ：**レコードのタグ付けを有効にします。タグは、タグカタログに分類されます。タグカタログを管理する管理者を割り当てます。
- **評価：**レコードの評価を有効にします。

チームアップの概要

EBX® データモデルへの管理アクセス権を持っている場合は、個々のテーブルを登録してチームアップを有効にします。デフォルトでは、テーブルが登録されると、すべてのチームアップ機能がアクティブで表示されるように設定されます。データモデルが公開された後、読み取り/書き込みまたは読み取り専用アクセス権を持つユーザーはチームアップを使用できます。チームアップ構成設定を使用して、ユーザーアクセスと個々の機能を制限します。さらに、各機能の可視性を構成できます。たとえば、機能はアクティブでも非表示にすることができます。ユーザーには表示されませんが、以前のコメント、評価、またはタグは引き続きレコードに関連付けられています。

チームアップへのテーブルの登録

チームアップデータモデル拡張機能を使用して、各テーブルを登録します。

1. >> に移動し、新しいレコードを作成します。

注意:

拡張機能が表示されない場合、この機能を有効にします。手順については、[データモデルの拡張機能](#)を参照してください。

2. メニューを使用して、チームアップ機能を有効にするテーブルを選択します。
3. 、 、 グループを使用して、ユーザー権限、機能の可用性、モデレーター、タグカタログを定義します。

詳細については、[ユーザーアクセスとチームアップ機能の管理](#)を参照してください。

4. 変更を保存後、データモデルを公開します。

ユーザーアクセスとチームアップ機能の管理

チームアップ機能へのユーザーアクセスを、データモデルおよびデータセットレベルで管理します。次の表は、各場所で使用できる設定を示しています。

	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: <i>Extensions > Team Up</i> 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.

タグとタグカタログの管理

タグとタグカタログを使用して、関連データを見つけて整理します。チームアップが有効になっている各テーブルは、デフォルトのタグカタログに自動的に関連付けられます。関連付けるカタログを変更できます。テーブルのレコードに割り当てることができるのは、関連付けられたカタログ内のタグのみです。次の表は、タグおよびタグカタログに関連するアクションを実行する方法についての説明です。

タグカタログと管理者の作成	- 管理アクセスを使用して、[管理] > [メタデータ管理] > [チームアップ管理] > [タグ] > [タグカタログ] からタグカタログを作成します。 [管理] > [メタデータ管理] > [チームアップ管理] > [タグ] > [タグカタログ管理者] を選択し、カタログに管理者を割り当てます。
タグの作成	- ビジネスユーザーがタグカタログに存在しないレコードにタグを割り当てると、新しいタグが自動的に作成されます。 - 管理アクセスを使用して、[管理] > [メタデータ管理] > [チームアップ管理] > [タグ] > [タグ] からタグを作成します。
カタログへのタグの割り当て	[管理] > [メタデータ管理] > [チームアップ管理] > [タグ] > [タグ] を選択し、タグが属するカタログを変更します。このタグは、割り当て済みのレコードでは使用できなくなることに注意してください。

AIアシスタント

- はじめに
- LLM構成
- アシスタンス機能
 - アシスタンス機能の作成

- 。アシスタンス機能フィールドの説明
- 。アシスタンス機能の例

はじめに

AIアシスタントは、生成AI、特に大規模言語モデル（LLM）を活用し、レコードデータに基づいて提案を生成することで、EBX®ユーザーがタスクを完了するのを支援します。一般的なユースケースには、テキストの生成、テキストの要約、値の翻訳、フォーマットの標準化などがあります。

注:

EBX®は外部LLMプロバイダーサービスへの接続を必要とします。EBX®自体にはLLMは含まれておらず、提供もしていません。詳細については以下を参照してください。

LLM構成

LLM構成は、使用するLLMプロバイダーサービスとモデルを定義します。クラウドベースのLLMプロバイダーサービス（例：OpenAI、Azure OpenAI）のアクティブなアカウント、またはローカル/オンプレミスのLLMサーバー（例：Ollama）が必要です。LLMを構成する際には、プロバイダーのAPIにアクセスするために必要な資格情報とネットワークアクセスが必要です。

LLM構成を作成するには：

1. **管理 > AIアシスタント > AIアシスタント管理** に移動します。
2. **LLMテーブル**を開き、レコードを作成して新しいLLM構成を追加します。
3. LLMプロバイダーとタイプを選択します。プロバイダーを選択すると、次のステップで**外部サービス構成**を作成する際に使用するキーに関する詳細情報メッセージが表示されます。

必要なLLMモデルが**** LLMタイプ****にリストされていない場合は、以下の手順に従って指定します。

- a. **LLMタイプ**ドロップダウンから、********を選択します。
 - b. **保存**をクリックして、**LLMカスタムタイプ**フィールドを表示します。
 - c. **LLMカスタムタイプ**フィールドに、必要なLLMモデルの名前を入力します。
4. **外部サービス構成**を選択するか、新しいものを作成します。
 - a. メニューを開き、**+ 作成**を選択します。
 - b. IDを追加した後、以下を提供します。

- **URL:** 特定のサービスに対して正しいベースURLを提供するだけで済みます。EBX®は、完全なリクエストURLを構築する際に、必要なAPIパス（例：`/chat/completions`）を自動的に追加します。以下にプロバイダー固有の例を示します。

Open AI	URLは： https://api.openai.com/v1
Azure Open AI	URLは、特定のAzureリソースエンドポイントに対応し、Azureポータルで見つけることができる以下の形式を使用します： https://YOUR_RESOURCE_NAME.openai.azure.com/ 。ここで、 <code>YOUR_RESOURCE_NAME</code> は、Microsoft AzureポータルでAzure Open AI サービスが作成されたときに割り当てられた一意の名前です。
Ollama	URLは、以下の形式でOllamaサービスのアドレスです： http://OLLAMA_SERVER_ADDRESS:PORT • <code>OLLAMA_SERVER_ADDRESS</code> は、Ollama サービスを実行しているサーバーのアドレスです。 • <code>PORT</code> は、Ollamaサービスが使用するように構成されているポート番号です。

- ****使用法**:** ****AIアシスタント****にチェックを入れたままにします

- ****認証**:** LLMプロバイダーへの接続にキーが必要な場合は****カスタム****を選択し、キーが不要な場合は****なし****を選択します。

- ****キー** / ****シークレットキー****:** 該当する場合、以前のステップで表示されたキーを、セキュリティ要件に応じて****キー****リストまたは****シークレットキー****リストに追加します。次に、APIキーを****値****フィールドに貼り付けます。

****シークレットキー****リストで定義されたキーは暗号化され、EBX®リポジトリに安全に保存

されます。詳細については、[キーストアファイル](external_services_configuration.md#キーストアファイル)を参照してください。

1. レコードを保存して閉じます。
2. 作成した外部構成が選択されていることを確認し、レコードを保存して閉じます。

LLM構成は、アシスタンス機能を作成する際に使用できるようになりました。

アシスタンス機能

アシスタンス機能は、特定のテーブルのコンテキストで実行され、AIアシスタントがLLMを使用してエンドユーザーに役立つ提案を生成する方法を定義します。具体的には、レコードデータ、ユーザー入力、および事前定義されたプロンプトを使用して、レコード内の1つ以上のフィールドの値を提案します。

アシスタンス機能の作成

アシスタンス機能を作成するには：

1. データモデルアシスタント（DMA）でデータモデルを開きます。
2. **AIアシスタント**データモデル拡張が有効になっていることを確認します。有効にする方法については、[データモデル拡張](#)を参照してください。
3. [拡張](#) > [AIアシスタント](#) > [アシスタンス機能](#) に移動し、新しいレコードを作成します。
4. 入力フィールドと出力フィールド、プロンプト、および適切な説明を定義します。各フィールドの詳細については、次のセクションを参照してください。
5. LLMが使用するプロンプトを記述します。
6. 機能を保存します。
7. **機能プレビュー**サービスを使用して、公開前に機能をテストします。
8. データモデルを公開して、アシスタンス機能をエンドユーザーが利用できるようにします。

注:

LLMが期待される結果を返すことを確認するために、常に現実的なデータで機能をテストしてください。

アシスタンス機能フィールドの説明

テーブルパス	アシスタンス機能が実行されるテーブル。
ラベルと説明	アシスタンス機能のローカライズされたラベルと説明。
有効	アシスタンス機能が有効または無効であることを示します。
入力フィールド	プロンプトに値を提供するテーブルフィールド。 各入力フィールドについて、以下の情報を提供します。 • フィールド: 選択したテーブル内のフィールドパス • エイリアス: プロンプトで使用する一意の識別子
仮想入力フィールド	テーブルには存在しないが、アシスタンス機能に追加の入力を提供するオプションフィールド。 各仮想入力フィールドについて、以下の情報を提供します。 • エイリアス: プロンプトで使用する一意の識別子 • データ型: 仮想入力フィールドのデータ型 • ラベルと説明: 仮想入力フィールドのローカライズされたラベルと説明 • 最大長: フィールドに許可される最大文字数（テキストフィールドにのみ適用）

出力フィールド	生成された結果が保存されるフィールド。 各出力フィールドについて、以下の情報を提供します。 • フィールド: 選択したテーブル内の出力フィールドへのパス 警告: 外部キーを出力として使用する場合、以下の結果に注意してください。 • 参照されるテーブルのすべてのラベルがLLMに送信され、機密データが公開される可能性があります。 • 大量のラベルリストを送信すると、パフォーマンスに影響を与える可能性があります。
仮想出力フィールド	テーブルには存在しないが、アシスタンス機能に追加の出力を提供するオプションフィールド。 各仮想出力フィールドについて、以下の情報を提供します。 • 識別子: 仮想出力フィールドの一意の識別子 • データ型: 仮想出力フィールドのデータ型 • ラベルと説明: 仮想出力フィールドのローカライズされたラベルと説明
プロンプト	LLMが生成する出力をガイドするための指示。 プロンプト内で入力フィールドと仮想入力フィールドを参照するには、プレースホルダーパターン<code>\${Alias}</code>を使用します。ここ

で、**Alias**は入力フィールドのエイリアスまたは仮想入力フィールドのエイリアスです。

アシスタンス機能の例

機能	実装
スペルチェックアシスタンス機能	ユーザーが文字列フィールドの値をスペルチェックするのを支援する機能。 プロンプト: <code>\$(string_field)</code> のスペルチェック ここで、 string_field はスペルチェックするフィールドのエイリアスであり、機能の入力および出力として構成されます。
テキスト要約アシスタンス機能	ユーザーが文字列フィールドの値を要約するのを支援する機能。 プロンプト: <code>\$(long_string_field)</code> を要約 ここで、 long_string_field は要約するフィールドのエイリアスであり、機能の入力として構成されます。
翻訳アシスタンス機能	ユーザーが文字列フィールドの値を翻訳するのを支援する機能。 プロンプト: <code>\$(input_string_field)</code> をフランス語に翻訳 ここで、 input_string_field は翻訳するフィールドのエイリアスであり、機能の入力として構成されます。
郵便住所標準化アシスタンス機能	ユーザーが郵便住所を標準化するのを支援する機能。 プロンプト: 郵便住所を標準化: <code>\$(string_field)</code> ここで、 string_field は標準化するフィールドのエイリアスであり、機能の入力および出力として構成されます。
説明生成アシスタンス機能	ユーザーがレコードのフィールドに基づいて説明を生成するのを支援する機能。 プロンプト: 以下の情報に基づいて説明を生成: ラベル: <code>\$(label_field)</code> 、カテゴリ: <code>\$(category_field)</code> 、機能: <code>\$(features_field)</code> ここで、 label_field 、 category_field 、および** <code>features_field</code> は、機能の入力として構成されたフィールドのエイリアスです。

その他

- リネージ
- イベントブローカー
 - 概要
 - 管理

リネージ

リネージを管理するには、次の 3 つのテーブルにアクセスできます。

- **承認済みプロファイル**：データリネージ WSDL 生成に使用するには、このテーブルにプロファイルを追加する必要があります。
- **履歴**：一般的なデータリネージ WSDL とその構成を一覧表示します。
- **JMS の場所**：JMS URL の場所を一覧表示します。

イベントブローカー

概要

TIBCO EBX® は、イベントブローカー機能を使用して、特定のイベントに関連する通知や情報を受信する機能を提供します。この機能は、サブスクライバーが選択したトピックに従って、EBX® コアイベントに関連する通知をサブスクライバーに送信するものです。

用語

イベントブローカー	疎結合イベント処理用の通知コンポーネント。EBX® コアから関係するサブスクライバーに発生したイベントをディスパッチすることから構成されます。イベントブローカーは主に監視と統計の目的で使用されます。
トピック	メッセージを含む EBX® イベントタイプに対応します。トピックに登録されるサブスクライバーの数に制限はありません。
サブスクライバー	サブスクライブされたトピックに関連するイベントを受信するモジュール内のクライアント実装。

トピックス

データスペースおよびスナップショット	作成、閉じる、再度開く、削除、アーカイブのエクスポート、アーカイブのインポート(データスペースのマージのみ) など、データスペースおよびスナップショット内の操作に対応します。
リポジトリ	起動や削除など、リポジトリ内の操作に対応します。
ユーザーセッション	ログインやログアウトなど、ユーザー認証に関連する操作に対応します。

管理

管理コンソールは、領域の の下にあります。、、 の 3 つのテーブルが含まれています。

次の操作を除き、すべてのコンテンツは読み取り専用です。

- トピックとサブスクライバーは、専用サービスを使用して手動でアクティブ化または非アクティブ化できます。
- ブローカーに登録されなくなったサブスクライバーは削除できます。

イベントブローカーはスレッドプールメカニズムに基づいています。スレッドの最大数は、次のようにプロパティ ファイルで定義できます。

```
# Defines the number of thread pool executors to

# guarantee the publication of asynchronous events.

# The default value is 2

ebx.eventBroker.threadPool.size=2
```

分散データ配信 (D3)

- D3 の概要
- D3 ブロードキャストおよび配信データスペース

- D3 JMS 構成
- D3 管理

D3 の概要

- 概要
 - D3 アーキテクチャ
 - サードパーティシステムの関与
 - プロトコル
- D3 の用語
- 既知の制限事項
 - 一般的な制限事項
 - ブロードキャストと配信のデータスペースの制限
 - 管理上の制限

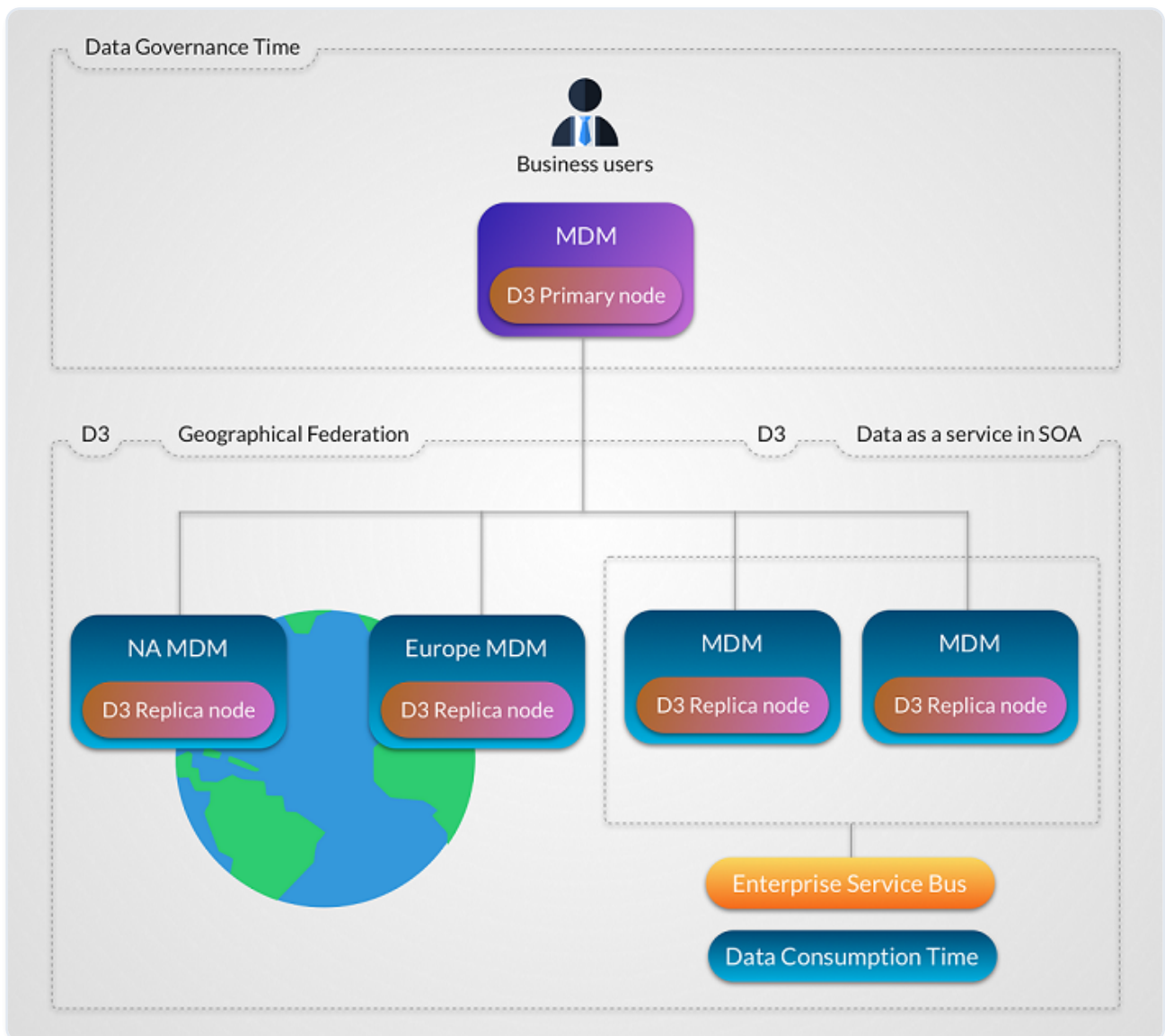
概要

TIBCO EBX® は、EBX® インスタンスから他のインスタンスにデータを送信する機能を提供します。ブロードキャストアクションを使用して、EBX® の他の機能にセキュリティと制御の追加レイヤーも提供します。これは、データガバナンスが最高レベルのデータ整合性、承認、およびロールバック機能を必要とする状況に特に適しています。

D3 アーキテクチャ

一般的な D3 インストールは、1 つのプライマリノードと複数のレプリカノードで構成されます。プライマリノードでは、データスチュワードは、どのデータスペースをブロードキャストする必要があるか、およびどのユーザープロファイルがそれらをレプリカノードにブロードキャストできるかを宣言します。データスチュワードは、1 つ以上のデータスペースのグループである配信プロファイルも定義します。

各レプリカノードは、ブロードキャストを受信する配信プロファイルを定義する必要があります。



サードパーティシステムの関与

D3 の機能により、サードパーティシステムはデータサービスを介して EBX® で管理されているデータにアクセスすることもできます。基本的に、システムが配信データスペースのデータをデータ消費すると、データは最後のブロードキャストスナップショットに透過的にリダイレクトされます。これにより、管理対象データのより制御された信頼性の高いビューが保証されます。

サードパーティシステムは、プライマリノードまたはレプリカノードを介してデータに直接アクセスできます。したがって、プライマリノードで構成され、レプリカノードがない物理アーキテクチャも可能です。

プロトコル

JMS がアクティブ化されている場合、プライマリノードとレプリカノード間の会話は SOAP over JMS に基づいており、アーカイブ転送は JMS バイナリメッセージに基づいています。

JMS がアクティブ化されていない場合、プライマリノードとレプリカノード間の会話は SOAP over HTTP(S) に基づいており、バイナリアーカイブ転送は TCP ソケットに基づいています。HTTPS を使用する場合は、信頼できる証明書で SSL を有効にして、ターゲットノードコネクタが正しく構成されていることを確認してください。

以下も参照してください。:

- 分散データ配信用の JMS (D3)

D3 の用語

ブロードキャスト	プライマリノードからレプリカノードにデータの公式スナップショットの公開を送信します。ブロードキャストは、データがレプリカノードに転送されることを透過的かつトランザクショナルに保証します。
配信データスペース	配信データスペースは、専用のアクションを使用して、認証および承認されたユーザーにブロードキャストできるデータスペースです。 デフォルトでは、データサービスが任意のノードの配信データスペースにアクセスすると、ブロードキャストされた最後のスナップショットにリダイレクトされます。データサービス を参照してください。
配信プロファイル	配信プロファイルは、1 つ以上の配信データスペースをグループ化する論理名です。レプリカノードは、1 つ以上の配信プロファイルにサブスクライブします。
クラスター配信モード	サブスクライブされたレプリカノードとの同期は、2 フェーズコミットトランザクションプロセスで実行されます。この配信モードは、負荷分散やフォールトトレランスを使

	用して大量のクエリに応答するように設計されています。これにより、レプリカノードとそのプライマリノード配信データスペース間のクラスター内のデータの整合性が保証されます。プライマリノードとレプリカノードは、同じ最後のブロードキャストスナップショットを使用します。
フェデレーション配信モード	同期は単一フェーズで実行され、登録されたレプリカノードごとに個別に実行されます。この配信モードは、応答時間とネットワークの可用性が保証されない、地理的に分散したアーキテクチャや異種のアーキテクチャで使用するよう設計されています。いつでも、レプリカノードは異なる最後のブロードキャストスナップショットにある可能性があります。したがって、同期プロセスは互いに独立しており、特定のブロードキャスト障害に対して個々のレプリカノードの再生が実行されます。
プライマリノード	1 つ以上の配信データスペースを定義でき、レプリカノードがサブスクライブできる EBX® のインスタンス。プライマリノードは、通常の EBX® サーバーとしても機能します。
レプリカノード	配信データスペースブロードキャストを受信するために、プライマリノードに接続された EBX® のインスタンス。配信データスペースの更新制限に加えて、レプリカノードは通常の EBX® サーバーとして機能します。
ハブノード	プライマリノードおよびレプリカノードとして動作する EBX® のインスタンス。プライマリノードの配信データスペースとレプリカノードの配信データスペースは分離している必要があります。

既知の制限事項

一般的な制限事項

- 各レプリカノードには、プライマリノードが 1 つだけ必要です。
- 埋め込みデータモデルは、D3 データスペースでは使用できません。したがって、D3 データスペースの公開に基づいてデータセットを作成することはできません。
- 少なくとも 1 つのレプリカノード製品バージョンがプライマリノードと異なる場合、互換性は保証されません。

ブロードキャストおよび配信データスペースの制限

- データスペースのアクセス権はブロードキャストされませんが、データセットのアクセス権はブロードキャストされます。
- データスペース情報はブロードキャストされません。
- データスペースとその親がブロードキャストされると、レプリカノードでそれらの親子関係が失われます。
- スナップショットがレプリカにブロードキャストされると、その後同名のどのスナップショットがブロードキャストされた場合でも、レプリカノードでは最初にブロードキャストされた同名のバージョンがリストアされることになります。つまり、プライマリノード上のオリジナルのスナップショットが削除され、同じ名前の新しいスナップショットが作成され、その後ブロードキャストされた場合、レプリカのコンテンツは、同じ名前の最新のスナップショットではなく、以前にブロードキャストされたスナップショットのものにリストアされることになります。
- D3 ノード間のデータスペースの一貫性を保証するには、ブロードキャストコンテンツの基になるデータモデル (モジュールに埋め込まれている、またはパッケージ化されている) が、プライマリノードとそのレプリカノード間で同じにする必要があります。
- レプリカ配信データスペースで、複数のレプリカノードが登録されていて、データモデルでレプリケーションが有効になっている場合、すべてのノードで有効になります。D3 ノードに従ってレプリケーションをアクティブ化/非アクティブ化するための設定はありません。
- レプリケーションは分散トランザクションに参加しません。コミット後に自動的にトリガーされます。

管理上の制限

テクニカルデータスペースはブロードキャストできないため、EBX® のデフォルトのユーザーディレクトリを D3 を使用して同期することはできません。

D3 ブロードキャストおよび配信データスペース

- ブロードキャスト
 - ブロードキャストの範囲と内容
 - ブロードキャストの実行
 - 条件
 - 永続性
 - 宛先
- レプリカノードの登録
 - 範囲と内容
 - 初期化の実行
 - 条件
- 配信データスペースへのアクセス
 - データサービス
 - アクセス制限
 - D3 ブロードキャスト Java API

ブロードキャスト

ブロードキャストの範囲と内容

D3 ブロードキャストは、データスペースレベルまたはスナップショットレベルで発生します。データスペースブロードキャストの場合、D3 はまずスナップショットを作成して現在の状態をキャプチャし、次にこの新しく作成されたスナップショットをブロードキャストします。

ブロードキャストでは、状況に応じて次のいずれかの手順が実行されます。

- 新しいブロードキャストスナップショットとレプリカノード上の現在の「コミット」スナップショットとの間で計算された差異の更新
- すべてのデータセット、テーブル、レコード、および権限を含む完全な同期。これは、前のレプリカノードのコミットがプライマリノードに認識されていない場合、特定のレプリカノードへの最初のブロードキャスト時に実行されるか、または「プライマリノード構成」のユーザーサービスを使用してオンデマンドで実行されます

以下も参照してください。:

- 。プライマリノード上のサービス

ブロードキャストの実行

ブロードキャストを実行するには、次の方法があります。

- エンドユーザーがデータスペースまたはスナップショットで利用可能な アクションを使用する (このアクションは、データスペースが配信データスペースとして登録されている場合にのみ使用できます)
- `D3NodeAsMaster` を使用するカスタム Java コードを使用する

条件

ブロードキャストを実行するには、以下の条件を満たす必要があります。

- 認証されたユーザープロファイルにはブロードキャストする権限があります
- ブロードキャストするデータスペースまたはスナップショットに検証エラーはありません

****注意：****推奨はされませんが、検証エラーを含む配信データスペースのブロードキャストを強制することは可能です。これを行うには、領域の [プライマリノード] 構成で、配信データスペース検証レポートで許可される最大重大度しきい値を設定します。

- D3 プライマリノード構成には、関係する配信データスペースの技術記録とそのすべての依存関係 (依存配信マッピング、配信プロファイル、登録済みレプリカ ノード) の範囲に関する検証エラーはありません
- 関連する配信プロファイルがあります
- データスペースをブロードキャストする場合、データスペースはロックされません
- スナップショットをブロードキャストする場合、スナップショットは配信データスペースとして宣言されたデータスペースに属しており、まだ現在のブロードキャストスナップショットではありません (ただし、以前にブロードキャストされたスナップショットへのロールバックは可能です)
- データスペースまたはスナップショットには、最後のブロードキャストスナップショットと比較した差異が含まれています

永続性

プライマリノードがシャットダウンすると、待機中または進行中のすべてのブロードキャストリクエストが中止され、一時ファイルに保持されます。起動時に、中止されたすべてのブロードキャストが再開されます。

以下も参照してください。:

- [一時ファイル](#)

宛先

ターゲットレプリカまたはハブノード側

- `ebx-d3-reference` データスペース識別子は、すべての配信データスペースの共通の親です
- 配信データスペースは、プライマリ、レプリカ、またはハブノードで同じ識別子を持ちます
- 配信データスペースが見つからない場合、最初の同期ブロードキャストまたは完全同期ブロードキャスト時に作成されます
- 最初のブロードキャストまたは完全同期時に配信データスペースがすでに存在する場合は、上書きされます
- 配信データスペースと同じ識別子を持つ既存のデータスペースが `ebx-d3-reference` の外部で検出された場合、エラーが発生します

以下も参照してください。:

- [既知の制限事項](#)

注意:

ブロードキャストは非同期で実行されます。したがって、ブロードキャストの成功または失敗に関する情報はユーザーインターフェイスに表示されません。ただし、[プライマリ ノード構成] 内でブロードキャスト操作を監視することは可能です。[監視](#) を参照してください。

レプリカノードの登録

範囲と内容

レプリカノードの TIBCO EBX® メイン構成ファイルに登録されている配信プロファイルに従って、レプリカノードレベルで初期化が行われます。プライマリノードは、その初期化要求を受信すると、レプリカノードエントリを作成または更新し、登録されているすべての配信データスペースの最新のブロードキャストスナップショットを送信します。

注意:

登録されたレプリカノードリポジトリ ID または通信レイヤーがすでに存在する場合は、「登録されたレプリカノード」技術テーブル内のレプリカノードエントリが更新され、存在しない場合は新しいエントリが作成されます。

初期化の実行

初期化は次のように実行できます。

- レプリカノードサーバーの起動時に自動的に実行されます
- レプリカノードサービス「レプリカノードの登録」を呼び出すときに手動で行います

条件

登録するには、以下の条件を満たす必要があります。

- D3 モードは「ハブ」または「スレーブ」にする必要があります
- プライマリノードとレプリカノードの認証パラメーターは、それぞれのディレクトリで定義されているプライマリノード管理者とレプリカノード管理者に対応している必要があります
- レプリカノードで定義された配信プロファイルは、プライマリノード構成に存在する必要があります
- 登録されたデータスペースに含まれるすべてのデータモデルは、レプリカノードに存在する必要があります。埋め込まれている場合、データモデル名は同じにする必要があります。パッケージ化されている場合、それらは同じモジュール名に配置され、モジュール内のスキーマパスはプライマリノードとレプリカノードの両方で同じにする必要があります
- D3 プライマリノード構成には、登録されたレプリカノードの技術記録とそのすべての依存関係 (依存配信プロファイル、配信マッピング、配信データスペース) の範囲に関する検証エラーはありません

注意:

パラメーターを設定するには、[プライマリ、ハブ、レプリカ ノードの構成](#) のレプリカまたはハブ EBX® プロパティを参照してください。

配信データスペースへのアクセス

データサービス

デフォルトでは、データ サービスが配信データスペースにアクセスすると、最新のブロードキャストスナップショットにリダイレクトされます。ただし、このデフォルトの動作は、リクエストレベルまたはグローバル構成で変更できます。

以下も参照してください。:

- 共通パラメーター '[disableRedirectionToLastBroadcast](#)'

アクセス制限

プライマリノードでは、配信データスペースをマージすることも閉じることもできません。権限に応じて、他の操作を実行できます。たとえば、配信データスペースを直接変更したり、ブロードキャストから独立したスナップショットを作成したり、子データスペースを作成してマージしたりできます。

レプリカノードでは、ブロードキャストプロセスを除き、エンドユーザー、データサービス、または Java プログラムによる配信データスペースへのいかなる変更も行うことはできません。さらに、マージ、クローズなどのデータスペース関連の操作は、レプリカノードでは禁止されています。

D3 ブロードキャスト Java API

最後のブロードキャスト スナップショットは、その間にブロードキャストが行われた場合、2つの呼び出し間で変更される可能性があります。複数の連続した呼び出しで完全に安定したビューが必要な場合は、これらの呼び出しで同じスナップショットを明示的に参照する必要があります。

最後のブロードキャスト スナップショットを取得するには、`D3Node.getBroadcastVersion` を参照してください。

D3 JMS 構成

- 分散データ配信用の JMS (D3)
 - プライマリノードとレプリカノードの共通宣言 (共有キューの場合)
 - プライマリノードでのオプションの宣言 (レプリカ固有のキュー用)
 - JMS 構成の例

分散データ配信用の JMS (D3)

デフォルトの HTTP および TCP プロトコルの代わりに JMS を使用するように D3 を構成するには、'ebx' Web アプリケーションの `WEB-INF/web.xml` 展開記述子で宣言されている [JMS 接続ファクトリ](#) と次のキューを構成する必要があります。

注意:

TIBCO EBX® メイン構成が、プロパティ `ebx.d3.mode`、`ebx.jms.activate`、および `ebx.jms.d3.activate` を通じて JMS および D3 (「スレーブ」、「ハブ」、または「マスター」ノード) をアクティブ化しない場合、以下の環境エントリは EBX® ランタイムによって無視されます。これらのプロパティの詳細については、EBX® メイン構成プロパティの [JMS](#) および [分散データ配信 \(D3\)](#) を参照してください。

プライマリノードとレプリカノードでの共通宣言 (共有キューの場合)

予約済みリソース名	デフォルトの JNDI 名	説明
<code>jms/EBX_D3MasterQueue</code>	JBoss: <code>java:/jms/EBX_D3MasterQueue</code>	D3 プライマリ JMS キュー (D3 モード「スレーブ」または「ハブ」のみ)。SOAP 要求を D3 プライマリノードに送信するために使用されるキュー名を指定します。メッセージプロデューサーは、プライマリノードリポジトリ ID をヘッダーフィールド <code>JMSType</code> の値として設定し

予約済みリソース名	デフォルトの JNDI 名	説明
		ます。Java タイプ: <code>jakarta.jms.Queue</code>
<code>jms/EBX_D3ReplyQueue</code>	JBoss: <code>java:/jms/EBX_D3ReplyQueue</code>	D3 応答 JMS キュー (「シングル」モードを除くすべての D3 モード用)。SOAP 応答を受信するための応答キューの名前を指定します。消費は、ヘッダーフィールド <code>JMSCorrelationID</code> を使用してフィルタリングされます。Java タイプ: <code>jakarta.jms.Queue</code>
<code>jms/EBX_D3ArchiveQueue</code>	JBoss: <code>java:/jms/EBX_D3ArchiveQueue</code>	D3 JMS アーカイブキュー (「シングル」モードを除くすべての D3 モード用)。D3 ノードが使用する転送アーカイブキューの名前を指定します。消費量は、ヘッダーフィールド <code>JMSCorrelationID</code> を使用してフィルタリングされます。アーカイブの重みがプロパティ <code>ebx.jms.d3.archiveMaxSizeInKB</code> で指定されたしきい値よりも高い場合、アーカイブは複数のシーケンスに分割されます。したがって、消費量は、代わりにヘッダーフィールド <code>JMSXGroupID</code> および <code>JMSXGroupSeq</code> を使用してフィルタリングされます。Java タイプ: <code>jakarta.jms.Queue</code>
<code>jms/EBX_D3CommunicationQueue</code>	JBoss: <code>java:/jms/EBX_D3CommunicationQueue</code>	D3 JMS 通信キュー (「シングル」モードを除くすべての D3 モード用)。リクエストを受信する通信キューの名前を

予約済みリソース名	デフォルトの JNDI 名	説明
		指定します。消費は、現在のリポジトリ ID に対応するヘッダーフィールド <code>JMSType</code> を使用してフィルタリングされます。Java タイプ: <code>jakarta.jms.Queue</code>

注意:

これらの JNDI 名はデフォルトで設定されていますが、`EBX.ear` に含まれる Web アプリケーションアーカイブ `ebx.war` 内で変更できます (JBoss、Websphere、またはその他のアプリケーションサーバーを使用している場合)。

プライマリノード上のオプションの宣言 (レプリカ固有のキュー用)

注意:

5.5.0 より前の互換性の昇順、または単方向キュートポロジに使用されます。

プライマリノードの展開記述子は、各レプリカノードの特定の通信キューとアーカイブキューを宣言して手動で変更する必要があります。これは、「ebx.war」内の「web.xml」にリソース名を追加することから成ります。レプリカ固有のノードキューは、1 つ以上のレプリカノードで使用できます。

リソースには自由に名前を付けることができますが、関連付けられているキューの物理名は、リソース `jms/EBX_D3ArchiveQueue` および `jms/EBX_D3CommunicationQueue` のレプリカノードの定義に対応している必要があります。

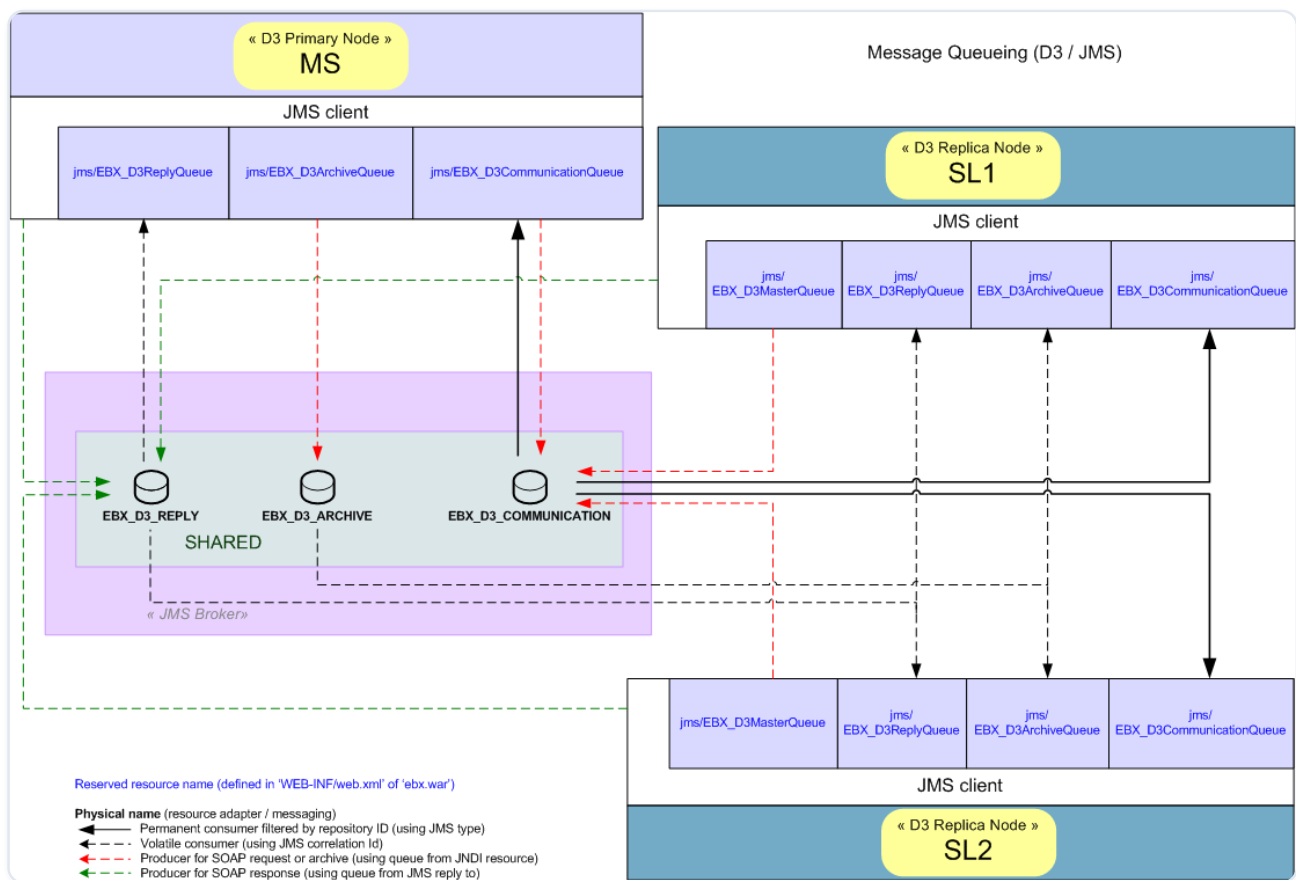
注意:

物理キュー名の一致：登録時に、レプリカノードは通信およびアーカイブの物理キュー名を送信します。これらのキューは、プライマリノードで宣言されたすべてのリソース間で物理キュー名によって一致します。一致しない場合は、登録は失敗します。

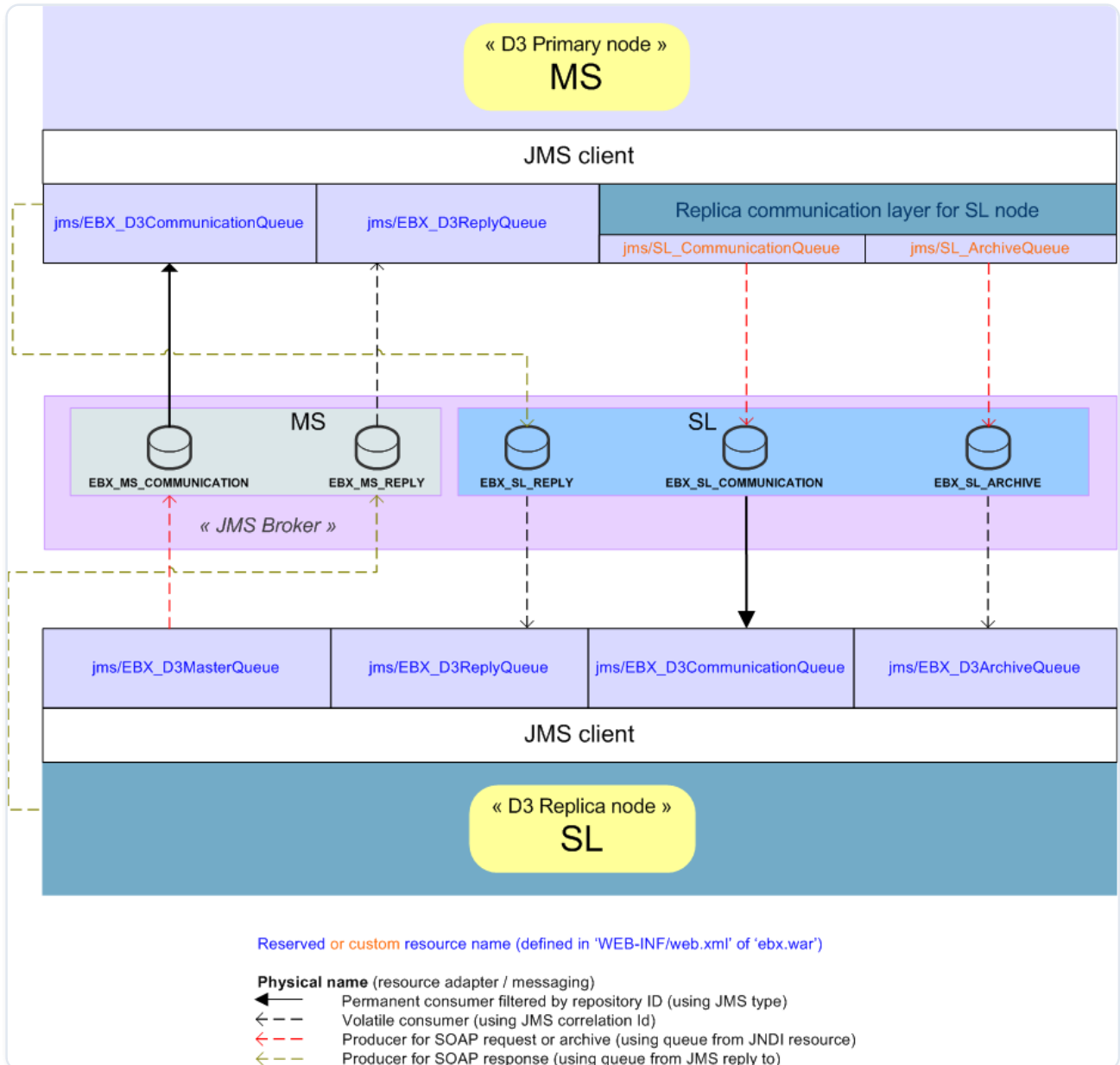
JMS 構成の例

	共有キュー	特定のキュー
プライマリレプリカノードのアーキテクチャ	プライマリノードと、共有キューを持つ2つのレプリカノード間	レプリカ固有のキューを持つプライマリノードとレプリカノードの間
Hub-Hub アーキテクチャ	共有キューを持つ2つのハブノード間	レプリカ固有のキューを持つ2つのハブノード間

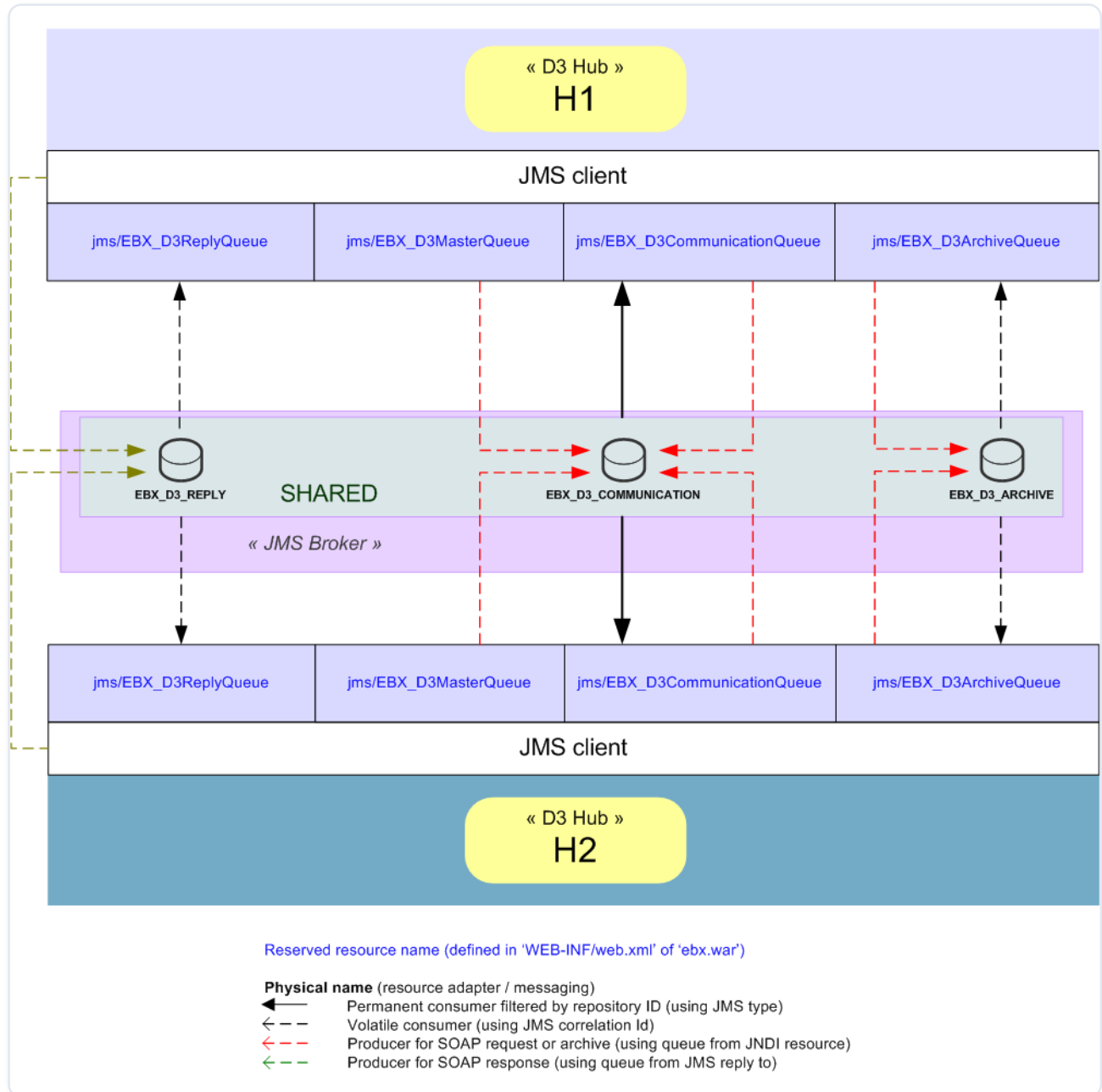
プライマリノードと、共有キューを持つ2つのレプリカノード間



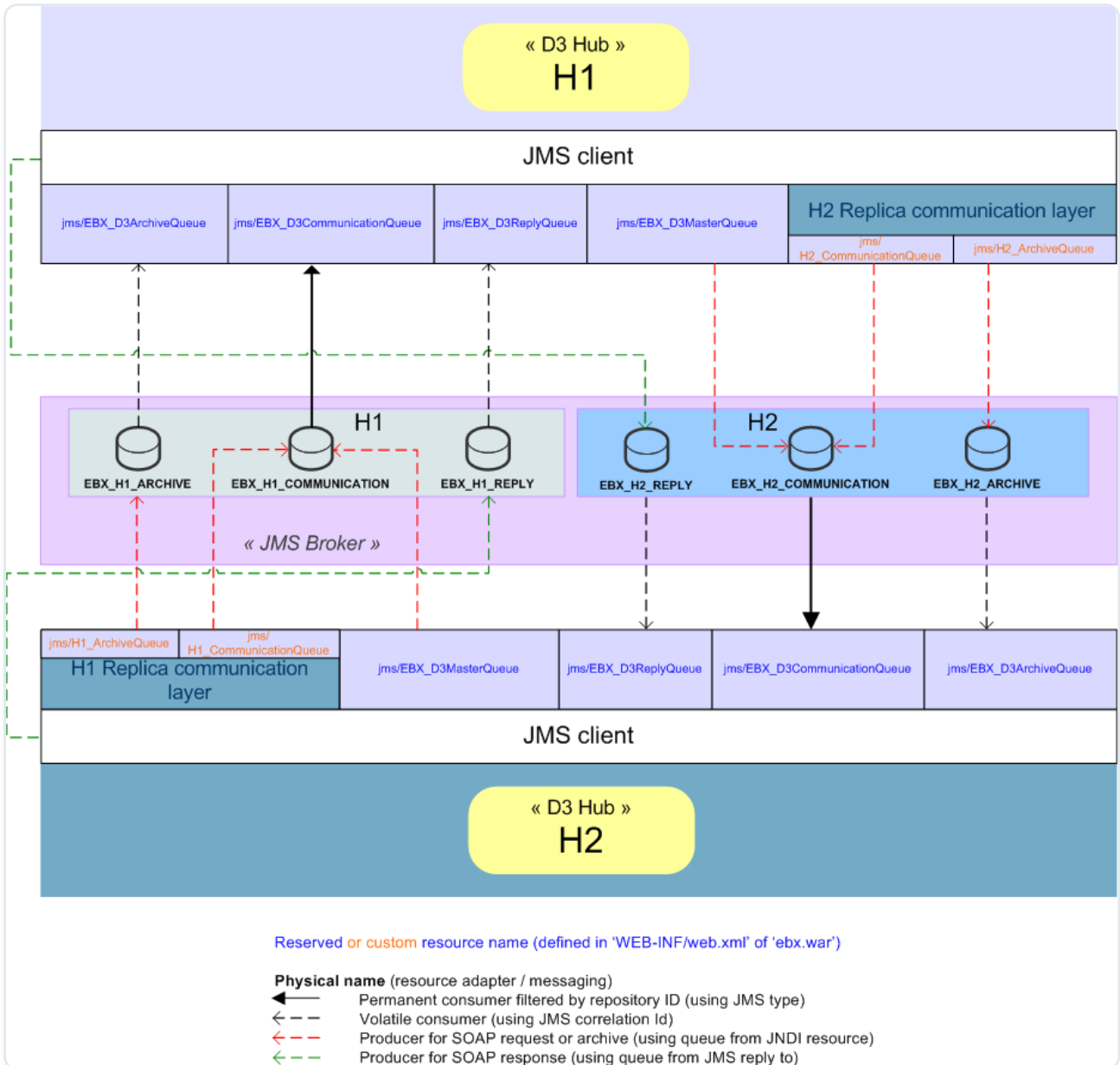
プライマリノードとレプリカ固有のキューを持つレプリカノード間



共有キューを持つ2つのハブノード間



レプリカ固有のキューを持つ2つのハブノード間



D3 管理

- クイックスタート
 - プライマリノードでの既存データスペースの宣言
 - 登録用レプリカノードの構成
- D3 ノードの構成
 - ユーザーインターフェイスを介したプライマリノードとハブノードのランタイム構成
 - プライマリノード、ハブノード、レプリカノードの構成
 - ノードのネットワークプロトコルの構成

- プライマリノード上のサービス
- レプリカノードのサービス
- 監視
 - プライマリノード管理コンソール
 - プライマリノード監視サービス
 - JavaAPI を介したレプリカノードの監視
 - プライマリノードの通知
 - ログ監視
 - 一時ファイル

クイックスタート

このセクションでは、2 つのTIBCO EBX® インスタンスを使用した基本的な D3 アーキテクチャの構成を紹介します。開始する前に、各インスタンスが独自のリポジトリで正しく機能することを確認してください。

注意:

EBX® を 2 つの異なる Web アプリケーションコンテナに展開します。両方のインスタンスが同じホストで実行されている場合は、すべての通信 TCP ポートが異なることを確認してください。

プライマリノードでの既存データスペースの宣言

目的は、プライマリ ノードから既存のデータスペースを構成してブロードキャストすることです。

この構成は、D3 インフラストラクチャー全体 (プライマリ ノードと レプリカ ノードを含む) で実行されます。

`ebx.properties` プライマリ ノード構成ファイルを次のように更新します。

1. キー `ebx.d3.mode` でD3 モードを `primary` として定義します。

注意:

プライマリ ノードは、構成後に開始できます。

ビルトインの管理者として認証した後、タブ内を移動します。

1. 前提条件：ノードが プライマリ ノードとして構成されていることを確認します (メニューで使用し、を確認します)。
2. [プライマリ構成] 管理機能を開きます。
3. ブロードキャストするデータスペースを「配信データスペース」テーブルに追加し、許可されたプロファイルを宣言します。
4. 配信プロファイル を「配信プロファイル」テーブルに追加し (論理名に対応している必要があります)、配信モードを宣言します。可能な値は、**クラスターモード** または **フェデレーションモード** です。
5. 配信プロファイルを使用して配信データスペースを「配信マッピング」テーブルにマッピングします。

注意:

これで、プライマリ ノードは、配信プロファイルでレプリカノードを登録する準備が整いました。

D3 ブロードキャストメニューがデータスペースまたはそのスナップショットの1つのメニューに表示されることを確認します。

登録用レプリカノードの構成

目的は、配信プロファイルと通信設定に基づいて レプリカ ノードを構成および登録することです。

`ebx.properties` レプリカノード構成ファイルを次のように更新します。

1. キー `ebx.d3.mode` で D3 モードを `replica` として定義します。
2. `ebx.d3.delivery.profiles` に プライマリ ノードで設定する `配信プロファイル` を定義します (配信プロファイルはカンマとスペースで区切る必要があります)。
3. ノード通信のための プライマリ ノードユーザー認証 (ビルトイン管理者プロファイルが必要) を `ebx.d3.master.username` および `ebx.d3.master.password` に定義します。

4. プロパティキー `ebx.d3.master.url` の値を設定して、プライマリノード通信用の `HTTP/TCP` プロトコル を定義します。

(例: `http://localhost:8080/ebx-dataservices/connector`)

5. ノード通信のための レプリカ ノードのユーザー認証 (ビルトインの管理者プロファイルが必要) を `ebx.d3.slave.username` および `ebx.d3.slave.password` に定義します。
6. プロパティキー `ebx.d3.slave.url` の値を設定して、レプリカノード通信用の `HTTP/TCP` プロトコル を定義します。

(例: `http://localhost:8090/ebx-dataservices/connector`)

注意:

レプリカノードは、構成後に開始できます。

ビルトインの管理者として認証した後、タブに移動します。

1. 前提条件: ノードが レプリカ ノードとして構成されていることを確認します (メニューで を 使用し、 を確認します)。
2. 管理機能を開きます。
3. 画面で情報を確認します。「N/A」の値を持つフィールドはありません。

注意:

ブロードキャストする前に、モデルが使用可能であることを確認します (データモデルアシスタントから、公開する必要があります)。

これで、レプリカノードをブロードキャストする準備が整いました。

D3 ノードの構成

ユーザーインターフェイスを介したプライマリノードとハブノードのランタイム構成

配信データスペースと配信プロファイルの宣言は、領域から [プライマリ構成] 機能を選択することによって行われます。次のテーブルが表示されます。

配信データスペース	ブロードキャストできるデータスペースの宣言。
配信プロファイル	レプリカノードがサブスクライブできるプロファイル。配信モードは、配信プロファイルごとに定義する必要があります。
配信マッピング	配信データスペースと配信プロファイルの間の関連付け。

注意:

上記の表は、一部のブロードキャストが保留中または進行中の間は読み取り専用です。

プライマリノード、ハブノード、レプリカノードの構成

このセクションでは、EBX® メイン構成ファイルでノードを構成する方法について詳しく説明します。

以下も参照してください。:

- [概要](#)

プライマリノード

プライマリノードとして機能するには、EBX® のインスタンスがメインの構成ファイルで次のプロパティを宣言する必要があります。

`ebx.d3.mode=master` ノードの構成例

```
#####

## 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
```

以下も参照してください。:

- プライマリノード

ハブノード

ハブノード (プライマリノード構成とレプリカノード構成の組み合わせ) として機能するには、EBX® のインスタンスがメイン構成ファイルで次のプロパティを宣言する必要があります。

`ebx.d3.mode=hub` ノードの構成例

```
#####

## 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=

```

以下も参照してください。:

- ハブノード

レプリカノード

レプリカノードとして機能するには、EBX® のインスタンスがメインの構成ファイルで次のプロパティを宣言する必要があります。

`ebx.d3.mode=slave` ノードの構成例

```
#####

## 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=
```

以下も参照してください。:

- [レプリカノード](#)

ノードのネットワークプロトコルの構成

このセクションでは、EBX® メイン構成ファイルでノードのネットワークプロトコルを構成する方法について詳しく説明します。

以下も参照してください。:

- [概要](#)

HTTP(S) およびソケット TCP プロトコル

HTTP (S) ネットワークプロトコルを使用した `ebx.d3.mode=hub` または `ebx.d3.mode=slave` ノードの構成例

```
#####

# 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 プロトコル

JMS がアクティブ化されている場合、D3 ノードの JMS 機能を有効にするために、次のプロパティを定義できます。

JMS ネットワークプロトコルを使用するすべての D3 ノードの構成例

```
#####

## 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=

以下も参照してください。:

- [分散データ配信用の JMS \(D3\)](#)

プライマリノード上のサービス

プライマリノードを管理するサービスは、レプリカノードの の下の 領域、および テーブルと テーブルで利用できます。サービスは次のとおりです。

リプレイの再起動	フェデレーションの配信を待機しているすべてのリプレイをすぐに再開します。
レプリカノード配信データスペースの削除	選択したレプリカノードの配信データスペースを削除するか、D3 プライマリノードの構成から登録を解除します。 サービスにアクセスするには、プライマリノードの [配信データスペース] テーブルから配信データスペースを選択し、ウィザードを起動します。
完全に再同期	最後にブロードキャストされたスナップショットの全コンテンツを、登録されたレプリカノードにブロードキャストします。
レプリカノードのサブスクライブ	選択したレプリカノードのセットをサブスクライブします。
レプリカノードの非アクティブ化	選択したレプリカノードをブロードキャストスコープから削除し、その状態を「利用不可」に切り替えます。 注意: 「進行中」のブロードキャストコンテキストはロールバックされます。
レプリカノードの登録解除	選択したレプリカノードをプライマリノードから切断します。 注意: 「進行中」のブロードキャストコンテキストはロールバックされます。

注意:

一部のブロードキャストが保留中または進行中の場合、上記のプライマリノードサービスは非表示になります。

レプリカノード上のサービス

プライマリノードへのサブスクリプションを管理したり、その他のアクションを実行したりするために、[レプリカの構成] ノードの領域でサービスが利用できます。

レプリカノードの登録	レプリカノードが登録解除されている場合は、プライマリノードに再サブスクライブします。
レプリカノードの登録解除	レプリカノードをプライマリノードから切断します。 注意: 「進行中」のブロードキャストコンテキストはロールバックされます。
スナップショットを閉じて削除	レプリカノード配信データスペースをクリーンアップします。 サービスにアクセスするには、レプリカノードの [配信データスペース] テーブルから配信データスペースを選択し、ウィザードに従って作成日に基づいてスナップショットを閉じて削除します。 注意： 最後のブロードキャストスナップショットは選択から自動的に除外されます。

監視

最後にブロードキャストされたスナップショットは、データスペースのスナップショットテーブルで強調表示され、最初の列に表示されるアイコンで表されます。

プライマリノード管理コンソール

プライマリノードの管理コンソールは、プライマリノードの領域の[プライマリノード構成]にあるいくつかのテーブルで構成されています。これらは次のとおりです。

登録されたレプリカノード	プライマリノードに登録されたレプリカノード。このテーブルから、各レコードでいくつかのサービスが利用可能になります。
ブロードキャスト履歴	これまで行われたブロードキャストの履歴。
レプリカノード登録ログ	実行された初期化操作の履歴。
詳細履歴	実行されたアーカイブ配信の履歴。関連する配信アーカイブのリストには、選択ノードを使用して、「ブロードキャスト履歴」および「初期化履歴」テーブルからアクセスできます。

プライマリノード監視サービス

プライマリノードの領域の [プライマリノード構成] で利用できます。サービスは次のとおりです。

レプリカノード情報の確認	レプリカノードと、レプリカノードの状態、関連付けられている配信プロファイル、配信されたスナップショットなどの関連情報を一覧表示します。
履歴コンテンツの消去	[ブロードキャスト履歴]、[レプリカノード登録ログ]、[詳細履歴] など、すべての履歴テーブル内のすべてのレコードを削除します。

Java API によるレプリカノードの監視

レプリカノードのステータスが「使用可能」または「使用不可」に切り替わったときにトリガーされるアクションを実装するために、レプリカノード監視クラスを作成できます。そのためには、`NodeMonitoring` インターフェイスを実装する必要があります。このクラスは、EBX® モジュールの外部にあり、「ebx.jar」のクラスローダーからアクセスできる必要があります、その完全なクラス名は [レプリカノード構成] で指定する必要があります。

以下も参照してください。:

- NodeMonitoring

プライマリノードの通知

D3 管理者は、ブロードキャストイベントを受信するためのメール通知を設定できます。

- ブロードキャスト失敗時
- フェデレーションブロードキャストで、リプレイが指定されたしきい値を超えた場合。

メールには、イベントの表と、オプションで詳細へのリンクが含まれています。

通知を有効にするには、領域から [プライマリノード構成] データスペースを開き、 の下のグループを構成します。

および オプションも、 リンクを使用して設定する必要があります。

ログ監視

技術的な監視は、EBX® メイン構成ファイルで宣言されたログカテゴリ「ebx.d3」を通じて行うことができます。例：

```
ebx.log4j.category.log.d3= INFO, Console, ebxFile:d3
```

以下も参照してください。：

- EBX® ログの構成

一時ファイル

交換されたアーカイブ、SOAP メッセージ、ブロードキャストキューなどの一部の一時ファイルが作成され、EBX® 一時ディレクトリに書き込まれます。この場所は、EBX® メイン構成ファイルで定義されています。

```
#####

## 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
```

ステージング

- [概要](#)
- [用語集](#)
 - [ステージングサーバー](#)
 - [ステージングエレメント](#)
 - [ステージングドメイン](#)
 - [ステージングアーカイブ](#)
 - [ステージングオペレーション](#)
- [ステージングの実行](#)
 - [インポート動作の指定](#)
- [ステージングドメインとベストプラクティス](#)

- ステージング権限と管理委任
- エラー処理
 - インポート操作中のエラー処理
 - エクスポート操作中のエラー処理
- サポートされているステージングエレメントのタイプ
 - データテーブル
 - プロファイル
 - データモデル
 - データセット
 - EBX® アドオン
- REST API
- 一時フォルダーの処理
- 起動時の自動インポート

概要

ステージング機能を使用すると、TIBCO EBX® リポジトリエレメントをある環境から別の環境に移動できます。たとえば、構成設定を開発環境からテスト環境にステージングできます。ステージング機能の一部として、多くのエレメントがサポートされています。ただし、その主な目的は、リポジトリに含まれる構成可能なエレメントを処理することであり、エンドユーザーによって更新されるデータではありません。サポートされているステージングエレメントのすべてのタイプのリストについては、[サポートされているステージングエレメントのタイプ](#)を参照してください。

注意:

グループにステージングするエレメントがない場合は表示されません。たとえば、**履歴プロファイル**は表示されません。

デフォルトでは、ステージング機能が有効になっています。`ebx.properties` ファイルに含まれる `ebx.staging.activated` パラメーターを使用して、さまざまな環境でステージングをアクティブ化または非アクティブ化します。

注意:

ステージング機能を使用するには、管理アクセスまたは委任されたステージングアクセスが必要です。詳細については、[ステージング権限](#)と[管理委任](#)を参照してください。

用語集

ステージングサーバー

開発、テスト、または実稼働専用の展開済みソフトウェア (EBX® を含む) で構成される環境。ステージングは、あるサーバーから別のサーバーに移動 (ステージング) される前に、更新されたソリューションまたは新しいソリューションを構築、テスト、および確認するために使用されます。

ステージングエレメント

ステージング可能な EBX® リポジトリ内の構成エレメント。さまざまなタイプのステージングエレメントには、データモデルパブリケーション、ワークフローモデルパブリケーション、データスペースなどがあります。

ステージングドメイン

EBX® サーバー間で転送できる一連のステージングエレメント。

ステージングアーカイブ

ドメインのエレメントをサーバーから別のサーバーにステージングするために使用されるファイル。

ステージングオペレーション

サーバーとの間でステージングアーカイブをインポートおよびエクスポートするアクション。

ステージングの実行

ステージングのプロセスは、エレメントを編成してステージングするためのドメインの作成から始まります。ドメインの準備が整うと、アーカイブファイルとしてエクスポートされます。最後に、アーカイブがターゲット環境にインポートされます。

この操作は、EBX® グラフィカルユーザーインターフェイスで実行するか、REST API を使用して実行できます。ステージングドメインのベストプラクティスについては、[ステージングドメインとベストプラクティス](#)を参照してください。

注意:

ステージングと互換性があるのは、マイナー番号が同じ製品バージョンのみです。

次の手順は、ユーザーインターフェイスを使用してステージングオペレーションを実行する方法を示しています。

1. ドメインを作成します。

以下も参照してください。:

- [ステージングドメインとベストプラクティス](#)
- a. ソース リポジトリで、> に移動し、 を選択します。
 - b. を選択し、 をクリックします。
 - c. 名前を付けて保存しますが、閉じないでください。
 - d. 目的のエLEMENTを選択し、必要に応じてカテゴリ内の特定のコンテンツを選択して、ステージングするELEMENTを定義します。選択リストは、変更時に自動的に保存されます。

注意:

エラー状態のエLEMENTは非表示になり、選択できません。

注意:

さらに、リポジトリ内の各ノードは、1000 個のエLEMENTの表示に制限されています。`ebx.properties` ファイルに含まれる `ebx.staging.maxElementPerList` プロパティを使用して、このデフォルト値を変更できます。

以下も参照してください。:

- [ステージングの有効化と構成](#)

- 画面上部またはドメインテーブルの各行にある を選択して、ステージングアーカイブをエクスポートおよびダウンロードします。
- ステージングアーカイブをインポートします。
 - ターゲットリポジトリで、[>](#) に移動し、 を選択します。
 - 画面上部の をクリックします。
 - ステージングアーカイブを参照して選択します。
 - インポートオプションのいずれかを選択します。詳細については、[インポート動作の指定](#)を参照してください。
 - をクリックします。

詳細については、[エラー処理](#)を参照してください。

インポート動作の指定

次の図と表は、ステージングアーカイブをインポートするときに使用できるオプションを示しています。

TIBCO® EBX®

Administration

Staging ▾

Actions ▾

Domains

▼ Reports

Import operations

Export operations

▸ Configuration

Import a staging archive

Upload and import a staging archive into the current repository.

Choose file

No file chosen

☐ **Force archive import.**

Imports the archive, even if it is identical to the most recently imported archive.

☐ **Force import of components.**

Import each component even if the identical component exists in the repository.

☐ **Save the staging domain definition.**

The staging domain definition will be saved in the staging management. (Should be used only when the target repository is for development).

☐ **Dry run (testing).**

Run a staging import for testing purposes without modifying the repository.

Import archive

アーカイブの強制インポート

同じステージングアーカイブがすでに正常にインポートされている場合、ステージングインポートはデフォルトで無視されます。このオプションを使用して、アーカイブのインポートの実行を強制します。最初のインポートが期待どおりに行われなかった場合は、このモードを使用し、ソースリポジトリに変更を加えた後に再試行します。

コンポーネントの強制インポート

同じステージングコンポーネントがすでに正常にインポートされている場合、デフォルトでは無視されます。コンポーネントのインポート実行を強制するには、このオプションを使用します。アーカイブを変更した後、このモードを使用し、実稼働に入る前に永続化プロセス全体を強制的に再生します。

	注意: 同じアーカイブがすでに正常にインポートされている場合は、[アーカイブの強制インポート] オプションもチェックして、コンポーネントのインポートを強制する必要があります。そうしないと、インポートプロセス全体がスキップされます。
ステージングドメイン定義の保存	デフォルトでは、インポート操作の実行時にステージングドメインレコードは Domains テーブルに作成されません。このオプションを使用して、ドメイン定義を手動でインポートします。
ドライラン (テスト)	このオプションは、リポジトリを変更せずにインポートを実行するためのテスト目的で使用されます。このオプションを使用して、宣言された依存関係が存在することを確認します。

ステージングドメインとベストプラクティス

ステージングドメインは、リポジトリの構成設定を整理するための効果的な手段です。ドメインを使用して、大きなリポジトリ構成を小さな部分に分割します。これにより、各パーツに異なるライフサイクルを持たせることができます。たとえば、人事および調達アプリケーションでは、さまざまな時期に更新が必要です。この場合、アプリケーションごとに個別のドメインを作成します。次に、複数の環境 (開発、テスト、実稼働など) でこれらのドメインを定義してステージングします。また、この方法により、構成リポジトリ全体を毎回テストする必要がなくなります。

ドメインには、ドット表記に準拠した一意の名前が必要です。サポートされている文字には、ASCII の英字と数字、アンダースコア、ダッシュ、およびドットが含まれます (名前をアンダースコアで始めることはできません)。この動作は、名前空間のように使用して階層を作成できます。

例: `com.acme.hr`、`com.acme.procurement`、`com.acme.procurement.catalog`、`com.acme.procurement.catalog.2022`。

ベストプラクティスには次のようなことがあります。

- ステージングライフサイクルでできるだけ早くドメインを明確に定義し始めます。

- 通常、開発環境の権限は実稼働環境の権限とは異なるため、権限には別のドメインを使用します。
- ステージングは、リポジトリ内の構成データのみを処理する必要があります。「データテーブル」と「フィルター処理されたデータ」のタイプはサポートされていますが、ステージングデータがそれらを更新し、ユーザーによる変更をオーバーライドする可能性があるため、ユーザー (地域、国など) によって更新されないデータに適用する必要があります。
- 長期間のトランザクションデータを含むデータスペースには、構成データではなく運用データが保持されます。それらは多数であり、ステージングコンポーネントのビューを乱雑にする可能性があります。>>>> で利用可能な設定を使用して、それらが表示されないようにします。
- 1 つ以上のステージングアーカイブをインポートする前に、データベースのバックアップを作成します。

ステージング権限と管理委任

デフォルトでは、EBX® の管理者はステージングにアクセスする権限を持っており、管理者以外のユーザーにアクセス権限を付与できます。アクセス権限は、個々のアクションまたはすべてのアクションの可用性を指定できます。ただし、ステージングには構成エレメントを作成および更新する機能があるため、ステージングへのアクセスを許可するときは注意が必要です。

管理委任を使用して、ステージング管理権限をユーザーに付与します。このプロセスは、ステージングドメインのコンテンツを選択するときに適用されるユーザー権限を使用して完了します。アクセス権限のないエレメントを選択してステージングすることはできません。

> [\[グローバル許可\]](#) の下の パネルで、管理委任を構成できます。

以下も参照してください。:

- [ユーザーとロールのディレクトリの構成](#)

次の表は、管理者とステージング管理委任を持つユーザーの権限の違いを示しています。

管理者	• ステージングアーカイブをエクスポートできます。
-----	---

	• ステージングアーカイブをインポートできます。
委任されたステージング管理者を持つユーザー	• ステージングアーカイブをエクスポートできます。 • ステージングアーカイブをインポートできません。 注意: このタイプのユーザーは、ステージングエレメントへのアクセス権限がなくても、他のユーザーまたは管理者によって作成されたドメインにアクセスでき、エクスポートできます。ドメイン定義の一貫性を確保する場合は、このオプションを選択します。書き込み権限を持つユーザーのみがドメインを作成できます。

エラー処理

インポート操作中のエラー処理

インポート操作により、エラーと警告が返される場合があります。インポートは成功するか、部分的に成功するか、失敗するか、無視されたエレメントまたは完全に無視されたエレメントを持つことができます。インポート操作は最初のエラーで停止せず、すべてのコンポーネントを試行します。エラーが発生すると、コンポーネントは正しくインポートされません。正常に完了するには、ソースまたはターゲットリポジトリの変更が必要になる場合があります。エラーメッセージは、問題を識別するのに役立つ十分な情報を提供する必要があります。

注意:

インポートは完全なトランザクションではなく、エラーによってリポジトリが不安定になる可能性があります。

依存関係のチェックが行われ、必要なエレメントが欠落している場合はエラーが返されます。これは、データセットを含むドメインをインポートし、そのデータセットに関連付けられたデータスペースがドメインまたは宛先リポジトリにない場合に発生する可能性があります。

警告が発生した場合、コンポーネントは正しくインポートされている可能性があります。機能が低下しているか、アクセスできない可能性があります。依存関係のチェック中に、参照されたオプションのエレメントが欠落している場合、警告が返されます。これは、データスペースの所有者プロファイルの場合です。

注意:

ステージングは、デフォルトで可能な限り書き込みを回避しようとします。この動作をオーバーライドして、ステージングエレメントを強制的に作成または更新できます。詳細については、[インポート動作の指定](#)を参照してください。

レポートのインポート

UI または REST を介して実行されるインポート操作ごとに、インポートレポートが生成されます。これには、ドメインとそのコンポーネントのインポートステータスが含まれます。レポートには、問題のトラブルシューティングに役立つ重大度 (`info` 、 `warning` 、 `error`) のメッセージも含まれています。

ドメインステータス値をインポートします。

- ****成功****: すべてのコンポーネントが正常にインポートされました。一部のコンポーネントがすでに存在し、インポート先のリポジトリに期待どおりに存在する場合、インポートはそれらのコンポーネントを無視する可能性があります。
- ****一部失敗****: 一部のコンポーネントのインポートでエラーが発生しました。他のコンポーネントは正常にインポートされました。[各コンポーネントのステータスインポート](#)を確認する必要があります。
- ****失敗****: アーカイブの読み取り中に例外が発生したか、コンポーネントのインポートがすべてエラーになっています。「一部失敗」ステータスとは異なり、リポジトリへの変更は実行されません。
- ****無視****: アーカイブはすでにインポートされているか、すべてのコンポーネントのインポートが無視されたため、無視されます。リポジトリへの変更は行われません。

- ****待機中：****非同期モードで使用されます。インポート操作はキューに入れられ、処理されるのを待っています。
- ****開始済み：****インポートプロセスが開始されましたが、終了していません。

注意:

システムの起動時に、ジョブが中断されたため、開始または待機中のすべてのインポートは自動的に失敗に渡されます。

コンポーネントのステータス値をインポートします。

- ****成功：****ステージングエレメントが正常に作成または更新されました。
- ****失敗：****ステージングエレメントのインポート中にエラーが発生しました。
- ****無視：****エレメントはリポジトリで期待どおりであるため、インポートは無視されます。
- ****開始済み：****インポートが開始され、終了していません。

エクスポート操作中のエラー処理

インポート操作とは異なり、エクスポート操作は完全なアーカイブエクスポートでのみ成功します。エラーが発生すると失敗します。ステージングエクスポートの部分的な状態はありません。

ドメイン内のコンポーネントが欠落している場合、エラーが返され、アーカイブは生成されません。

レポートのエクスポート

エクスポートレポートは、UI または REST を介して実行されるエクスポート操作ごとに生成されるレポートです。これには、アーカイブのエクスポートステータスが含まれます。

ドメインのステータス値をエクスポートします。

- ****エクスポート済み：****アーカイブは正常にエクスポートされました。
- ****要求済み：****非同期モードで使用されます。エクスポートが要求され、現在処理中です。
- ****中止：****エクスポートエラーまたは処理の中断により、アーカイブエクスポートが中止されました。
- ****期限切れ：****REST サービスの非同期モードを使用して、アーカイブが生成されましたが、有効期限が切れています。利用できなくなりました。

サポートされているステージングエレメントのタイプ

ステージング機能では、次のタイプのリポジトリエレメントがサポートされています。

- データスペース
 - データスペースの権限
 - データセット (詳細については、[データセット](#)を参照)
 - データセット権限
 - データテーブル
 - フィルター済みデータ
- データモデル (詳細については、[データモデル](#)を参照)
 - データモデル権限
- データモデルパブリケーション (詳細については、[データモデル](#)を参照)
- ワークフローパブリケーション
- パースペクティブ
 - パースペクティブ権限
- プロファイル (ロール) (詳細については、[プロファイル](#)を参照)
- ビューパブリケーション
 - パブリケーション権限の表示
- 管理
 - 推奨されるパースペクティブ
 - 推奨パースペクティブ権限
 - 履歴化プロファイル
 - 履歴とログの権限
 - タスクスケジューラ権限
 - ワークフロー実行権限
 - ワークフロー履歴権限
 - ワークフローインタラクション権限
 - ワークフローランチャー権限

- ワークフローの技術構成
 - データスペース管理権限
 - データモデリング権限
 - 自動インクリメント権限
- グローバル許可
 - スケジュールされたタスク
 - ワークフローモデル
 - スクリプトパブリケーション
 - スクリプトソース

データテーブル

データテーブルとフィルター処理されたデータは、デフォルトの構成でインポートされます。

- デフォルトのインポートモードは Update または Insert です
- ブロッキング制約が無効になっています
- by delta モードが有効になっています
- テーブルトリガーはインポート時に有効化されます

プロファイル

ステージングできるのはロールのみです。ステージングアーカイブをインポートするとき、ロールを除き、ディレクトリがデフォルトディレクトリである場合、プロファイルは無視されます。ステージングドメインをエクスポートすると、1 つまたは複数のプロファイル (ユーザーまたはロール) にリンクされたステージングエレメントが、ユーザーまたはロールがディレクトリにあるかどうかに関係なく、リンクとともにステージングされます。

注意:

ロール関係のサイクルの場合、サイクルを構成するすべてのロールは無視されます

データモデル

モジュール内にあるデータモデルパブリケーションおよびデータモデルは、ステージングに使用できません。ただし、対応するパブリケーションがモジュール内にある場合でも、データモデルをステージングできます (パブリケーションはステージングされません)。

データセット

子データセットをステージングすることはできません。ステージングエレメントをドメインに追加するときは表示されません。

EBX® アドオン

EBX® アドオンには、ステージングできる特定のコンポーネントがあります。以下のセクションでは、ステージングに使用できるアドオンコンポーネントと、ユーザーインターフェイス内のコンポーネントの場所について説明します。参考のために、以下の各表のステージングの場所の列は、>>>><<SelectedDomain>>> を指します。

ステージング可能なコンポーネントを含むアドオンは次のとおりです。

- TIBCO EBX® Data Model and Data Visualization Add-on
- TIBCO EBX® Data Exchange Add-on (New)
- TIBCO EBX® Insight Add-on (New)
- TIBCO EBX® Digital Asset Manager (New) Add-on
- TIBCO EBX® Match and Merge Add-on

TIBCO EBX® Data Model and Data Visualization Add-on

以下の表は、ステージングに使用できるアドオンのコンポーネントとその場所を示しています。

アドオンコンポーネント	ステージング場所 (カテゴリから開始)
DMA による保存済みグラフ	データモデル > DMA による保存済みグラフ :
値と関係のグラフの構成	データスペース > データセット > 値と関係のグラフの構成 :
DMA によるデータモデルグラフの構成	データモデル > DMA によるデータモデルグラフの構成 :

アドオンコンポーネント	ステージング場所 (カテゴリから開始)
データモデルのグラフ構成	データスペース > データセット > データモデルのグラフ構成：
保存済みグラフ	データスペース > データセット > 保存済みグラフ：
カスタムグラフ	カスタムグラフ：

TIBCO EBX® Data Exchange Add-on (New)

以下の表は、ステージングに使用できるアドオンのコンポーネントとその場所を示しています。

アドオンコンポーネント	ステージング場所 (カテゴリから開始)
SQL 構成	データスペース > データセット > データベースとデータモデルの接続 注意: データベース構成は、選択した [データベースとデータモデルの接続] 構成に合わせて自動的に移行されます。
ユーザーテンプレート	データスペース > データセット > テンプレート情報 注意: SQL サービスにテンプレートが使用されている場合、対応する SQL サービスが自動的に組み込まれます。

TIBCO EBX® Insight Add-on (New)

以下の表は、ステージングに使用できるアドオンのコンポーネントとその場所を示しています。

アドオンコンポーネント	ステージング場所 (カテゴリから開始)
ダッシュボード	ダッシュボード 注意: 各ダッシュボードには、データセット、テーブル、フィールドなど、ダッシュボードに含まれる特定の資産への依存関係があります。空のダッシュボードと無効なインジケータは自動的に除外されます。

TIBCO EBX® Digital Asset Manager (New) Add-on

以下の表は、ステージングに使用できるアドオンのコンポーネントとその場所を示しています。

アドオンコンポーネント	ステージング場所 (カテゴリから開始)
デジタル資産コンポーネント	データスペース > データセット > デジタル資産コンポーネント
デジタル資産	データスペース > データセット > ドライブ内のデジタル資産 注意: ステージングではデジタル資産のみを移行できます。物理ファイルは個別に移行する必要があります。

TIBCO EBX® Match and Merge Add-on

以下の表は、ステージングに使用できるアドオンのコンポーネントとその場所を示しています。

アドオンコンポーネント	ステージング場所 (カテゴリから開始)
照合の構成	データモデルパブリケーション > 照合の構成 > テーブル > 照合の構成

アドオンコンポーネント	ステージング場所 (カテゴリから開始)
	注意: 特定の照合の構成に属するすべてのエレメント (メイン情報、照合ポリシー、マージポリシー、およびレプリケーション) も移行されます。
照合ポリシー	データモデルパブリケーション > 照合の構成 > テーブル > 照合ポリシー 注意: 選択した照合ポリシーにリンクされている場合は、マージポリシーを含めて、特定の照合ポリシーに関連するすべてのコンポーネントが移行されます。例外は、照合の構成の主要な情報が含まれていないことです。
マージポリシー	データモデルパブリケーション > 照合の構成 > テーブル > マージポリシー 注意: 特定のマージポリシーに関連するすべてのコンポーネントも移行されます。

REST API

詳細については、[ステージングオペレーション](#)を参照してください。

一時フォルダーの処理

一時フォルダーは、ステージングオペレーションと非同期プロセスをサポートするために作成されます。作成された各フォルダーは、構成された遅延の後に有効期限が切れます。スケジュールされたタスク *DeleteExpiredStagingTemporaryFolders* により、期限切れのフォルダーが定期的に削除されます。

注意:

ステージング用に作成された一時フォルダーのディスク上の専用サイズは、プロパティ `ebx.staging.maxTemporaryFolderSizeInBytes` で構成できます。

起動時の自動インポート

起動時に特定のアーカイブを自動的にインポートするようにステージングを構成できます。これにより、インポートプロセスを手動で開始する必要がなくなり、環境内で必要なすべてのデータをタイムリーに利用できるようになります。

自動ステージングアーカイブインポートを設定するには、次の手順を実行します。

1. `ebx.properties` ファイルで、`ebx.staging.importOnStartup` プロパティを構成します。
2. インポートするアーカイブを含むフォルダーを宣言します。
 - a. フォルダーは、構成された `リポジトリディレクトリ` の下の `/startup` ディレクトリに物理的に配置される必要があります (例: `${ebx.home}/ebxRepository/startup/folder1`)。
 - b. 各フォルダーには、`instructions.json` という JSON ファイルが含まれている必要があります。このファイルは、そのフォルダー内のアーカイブのインポートの順序とオプションを指定します。

`instructions.json` ファイルの形式は次のとおりです。

```
{  
  
  "imports": [  
  
    {  
  
      "archiveFilePath": "myArchive1.zip"  
  
    },  
  
    {  
  
      "archiveFilePath": "myArchive2.zip",  
  
      "advanced_options": {
```

```

        "forceUpdateOnIdenticalComponent": false,

        "importDefinition": true

    }

}

]

}

```

インポートが失敗した場合、`errors.json` ファイルがフォルダー内に作成され、インポートプロセス中に発生したエラーの詳細なレポートが含まれます。インポートが成功すると、フォルダー内に `output.json` ファイルが生成され、正常に処理されたインポートのレポートが提供されます。

注意:

詳細オプションは必要に応じて使用し、UI のインポートオプション (コンポーネントのインポートを強制し、ステージングドメイン定義を保存) と同様の動作になります。

注意:

ファイル `output.json` を含むフォルダーは、すでに正常にインポートされたものと見なされ、起動中に無視されます。

注意:

宣言されたフォルダーと指示ファイルで指定されたアーカイブは、構成されたリポジトリの起動フォルダーの下に厳密に配置される必要があります。親フォルダーへの移動やシンボリックリンクの使用は許可されません。

新しいユーザーインターフェースのプレビュー

- 概要
 - 現在の制限事項
 - フィードバックの送信
- 新しいUIの有効化
- 新しいUIへのアクセス
- 一般的なナビゲーションの原則
- 用語集

概要

EBX® UIは大幅な改訂が行われています。目的は、直感的で効率的なナビゲーション体験を提供する最新のインターフェースを提供することです。このリリースから、新しいUIのプレビューが含まれています。新しいナビゲーションモデルは、現在のUIと共存できます。

注:

新しいUIの使用は任意です。既存のUIと並行して評価できますが、既存のUIは本番環境での使用が推奨されるインターフェースです。

現在の制限事項

UIプレビューで利用可能なコンテンツと機能範囲は意図的に制限されています。追加機能は今後のアップデートで導入されます。

注意:

これはプレビューであるため、すべての機能、UIコンポーネント、およびラベルは将来のリリースで変更される可能性があります。

フィードバックの送信

EBX®のあらゆる側面に関するフィードバックは常に歓迎されます。これはUIプレビューに関して特に当てはまります。[EBX® Ideas Portal](#)の特別なUI > UIプレビューカテゴリを使用して、フィードバックを送信してください。

新しいUIの有効化

新しいUIのプレビューは、管理者が有効にする必要があります。新しいUIを有効にすると、次のようになります。

- すべてのユーザーが利用できます
- ホームページを除き、パースペクティブなどの既存のユーザーインターフェースには影響しません
- プレビューは現在のUIと並行して動作し、置き換えるものではありません

新しいUIプレビューを有効にするには、次の手順を実行します。

- `ebx.jar` とともに `ebx-hub.war` ファイルをデプロイします
- `ebx.properties` ファイルで、プロパティ `ebx.internal.hub.unrestricted=true` を設定します。新しいUIモジュールがデプロイされていてもこのプロパティが設定されていない場合、ホームのみが利用可能であり、他の新しいUIコンポーネントや機能は利用できません。

新しいUIへのアクセス

新しいUIをデプロイして有効にした後、次の方法でアクセスできます。

- パースペクティブメニューから、ホームページオプションを使用します。新しいUIのアーティファクトがデプロイされていない場合、ホームページオプションはデフォルトのホームページに移動します。
- EBX® URLに `/ebx-hub` を追加します。

新しいインターフェースの操作に関する詳細については、[エンドユーザー向けの手順](#)を参照してください。

一般的なナビゲーションの原則

新しいUIは、ドック、ワークスペース、カードの3つの主要コンポーネントで構成されています。これらのコンポーネントは、ユーザーが機能にアクセスし、データをナビゲートする方法を定義

します。管理者の視点からは、ユーザー向けの機能はサービスを通じて公開され、ワークスペースにカードとして表示されます。

ドックは、画面の左側にある永続的な垂直領域です。その主な目的は、ユーザーのホーム、利用可能なパースペクティブ、進行中のデータ操作、および開いているテーブルとレコード間のナビゲーションを可能にすることです。

ワークスペースは、ドックの右側に表示される主要な領域です。新しいUIにアクセスすると、ワークスペースはデフォルトでユーザーのホームになります。ドックを使用してナビゲートすると、ワークスペースのフォーカスは選択されたビューまたはコンポーネントに自動的に更新されます。

サービスはユーザーにカードとして表示され、複数の列に整理されます。各列は、データスペース、データセット、テーブル、またはレコードなどのナビゲーションレベルを表し、表示または作業中の内容のコンテキストを提供します。深くナビゲートするにつれて、新しい列が右側に開き、以前の列はコンテキストを保持するために表示されたままになります。

ユーザーのホームに表示される一部のカードのコンテンツは構成可能です。ホームから移動した後、カードに表示されるコンテンツはパースペクティブの構成によって決定されます。

用語集

このセクションでは、このドキュメントで使用されている用語の定義を提供します。

ドック	画面の左側にある永続的な垂直バーで、EBX®ホーム、構成されたパースペクティブ、現在のデータ操作、開いているテーブルとレコードへのナビゲーションを提供します。
ワークスペース	ドックの右側に表示される主要な領域です。ユーザーがタスクを完了するためにデータを表示、操作、管理する場所です。
サービス	サービスは、構成してユーザーに提供できる機能を表します。サービスは、ユーザーがアクセスできるデータとアクションを定義します。パースペクティブでサービスが有効になっている場合、ワークスペース内の1つ以

	上のカードを通じてエンドユーザーに提示されます。
カード	カードは、情報、アクション、またはツールを提示するワークスペース内の構造化されたコンテナです。カードは、製品全体でデータを表示および操作する一貫した方法を提供します。権限に応じて、カードは編集可能、構成可能、または読み取り専用場合があります。
ホーム	新しいUIのパーソナライズされたランディングエリアです。ホームには、ショートカット、ドラフト、最近のアクティビティなどのユーザーが構成可能なカードが表示され、ナビゲーションの開始点として機能します。
プレビューUI	評価および早期導入のために提供される、EBX®ユーザーインターフェースの一般提供前のバージョンです。プレビュー機能はリリース間で変更される可能性があります。
パースペクティブ	役割とコンテキストに基づいて、ユーザーが利用できるサービス、データ、およびアクションを定義するEBX®の構成済みビューです。

EBX® メタデータ管理アプリケーション

- [メタデータ管理のパースペクティブ](#)
- [ユーザーロール](#)
- [権限](#)
- [ユーザーインターフェースの構成](#)
- [ワークフロー](#)
- [Web サービスと API](#)
- [Metadata Agent](#)

- TIBCO® メタデータエージェントのインストール
- TIBCO TDV メタデータアダプタのインストール
- TIBCO® EBX® メタデータアダプタのインストール
- TIBCO® メタデータエージェントフレームワーク
- TIBCO® メタデータエージェントの管理
- 技術ユーザー
- TIBCO® メタデータエージェント
- メタデータエージェントによって管理される項目
- 収集の構成

メタデータ管理のパーспекティブ

画面の左側にあるメインナビゲーションペインには、パーспекティブが含まれています。パーспекティブを使用すると、プロファイルに関連付けられた権限によって有効になっているすべての機能にアクセスできます。機能はセクションごとにグループ化されており、各セクションは独自の色で識別されます。パーспекティブのサイズは、右の境界線をドラッグして変更したり、右上隅の矢印をクリックしてアイコンのみに表示を縮小したりできます。

メタデータ管理パーспекティブでは、次のセクションにアクセスできます。

Metadata Administration

Metadata harvesting

-  Technical Users
-  Agents
-  Datasources
-  Assets
-  Links
-  Datasource types
-  Harvesting configurations
-  Intropection jobs
-  Intropection jobs history

Sampling

-  Sampling configuration

Permissions

-  Dynamic roles
-  Access rights
-  Access rules

- このセクションでは、TIBCO® メタデータエージェントを構成できます。使用可能なエージェント、サポートされているデータソースの種類、およびこれらのエージェントによって管理されるさまざまなオブジェクトのリストを表示できます。メタデータ収集ジョブと構成のステータスとトラブルシューティングを追跡できます。ここから、エージェントおよびメタデータ API を呼び出すその他のアプリケーションのテクニカルユーザーを作成することもできます。 - このセクション

では、サンプリングのデフォルトサイズを構成できます。 - このセクションでは、EBX® メタデータ管理アプリケーションでのさまざまなユーザーグループとさまざまなタイプのアクションおよびサービスに対するアプリケーション権限の管理について説明します。ユーザーにロールを割り当て、ユーザープロファイルに従ってメタデータのアクセスルールを定義し、ワークフロータスクに動的ロールを適用できます。コラボレーション権限を定義することもできます。

Workflows

-  Active workflows
-  Completed workflows
-  Activation
-  Provisioning workflow

Repository

-  Digital assets
-  Built-in assets
-  Samples and learning
-  Tags

UI Configuration

-  Default home page configuration

Trial

-  Load Trial Data

Downloads

-  Download agent

Advanced services

-  Factory reset

About

- このセクションでは、システムで実行されたアクティブなワークフローと完了したワークフローを追跡できます。また、作成アクションと更新アクションのワークフロー実行をアクティブ化および非アクティブ化したり、プロビジョニングパラメーターを定義したりすることもできます。 - このセクションから、メタデータにアタッチされたドキュメントや、資産サンプリングを使用して取得されたサンプルファイルにアクセスできます。ここには、ユーザーデータドライブ、ビルトインドライブ、機械学習/サンプルドライブの3つのドライブがあります。 - このセクションでは、デフォルトのホームページ構成とグリッドフォームを定義できます。 - このセクションは、トライアルデータにサブスクライブしているユーザーのみが使用できます。トライアル メタデータサンプルをシステムに読み込むことができます。 - ここから、TIBCO® メタデータエージェントのインストール準備完了パッケージをダウンロードできます。 - このセクションでは、システムの構成をデフォルト状態にリセットできます。リセットはすべての権限とビューに適用され、元の状態にリセットされます。 - ここから、EBX® メタデータ管理アプリケーションに関する情報にアクセスできます。

ユーザーロール

動的ロール

ロールは、データに関連している場合、動的です。特定のエンティティのすべての発生に対して、静的ロールを人のグループに付与できる場合、各レコードの異なるユーザーグループに動的ロールを付与できます。たとえば、エンティティの特定の発生の所有者は、人のグループを参照するこの発生のフィールドとして設定できます。このグループ自体は静的ロールを参照します。これらの関係を調べることで、発生を静的ロールにリンクできます。したがって、動的ロールは、特定の発生の値に基づいて静的ロールを解決する方法を説明します。

動的ロールの構成は、パースペクティブから行われます。

Metadata harvesting

- Technical Users
- Agents
- Datasources
- Assets
- Links
- Introspection jobs

Permissions

- Users
- Roles
- Dynamic roles
- Access Rights

+ Actions ▾

	Identifier	Label	Description
☐	1	Asset Owner	
☐	2	Instance Owner	
☐	3	Application Owner	

各動的ルールに対して、エンティティから解決パスを定義できます。以下の例では、特定の資産のアプリケーション所有者は、その資産を含むインスタンスとこのインスタンスのメインアプリケーションを取得することで解決できます。指定されたパスに従ってソースレコードが取得されたので、グループへの参照であるソース値 (ここでは所有者) を取得できます。このグループは静的ルールにリンクされているため、動的ルールはこの静的ルールによって解決できます。

フィールドに入力する値は、ソース値が選択したエンティティの現在のオカレンス内に直接存在する場合は空の文字列にすることができます。それ以外の場合は、参照するレコード内の外部キーまたは関連付けへのパスの連続にする必要があります。

Metadata harvesting

- Technical Users
- Agents
- Datasources
- Assets
- Links
- Introspection jobs

Permissions

- Users
- Roles
- Dynamic roles

Actions ▾

Identifier  2

Dynamic role * ▾

Entity * ▾

Source record

Source entity *  ▾

Source value * ▾

メインタブ

- このフィールドは必須です。識別子は自動的に生成されます。変更できません。 - このフィールドは必須です。このプランで解決する動的ルールを定義します。 - 動的ルールを解決するエンティティ。このエンティティの発生が解決の開始点になります。 - スペースで区切られます。空の場合、ソースレコードは前のフィールドエンティティで定義されたエンティティの現在のオカレンスです。 - このフィールドは必須です。

権限

権限の構成には、**パースペクティブからアクセスできます**。グループを使用すると、さまざまなデータカテゴリとアクセスルールに対するアクセス権限を定義できます。

アクセス権限

デフォルトのアクセス権限は、各ルールに期待される操作に応じて、すべてのカテゴリのすべてのデータエンティティに設定されます。これは、メニューのカテゴリにアクセスして変更できるデフォルトの構成です。アクセス権を定義することで、エンティティになり得るすべてのエレメント、そのフィールドの1つまたはセット、表示するかどうか、表示される場合は特定のユーザーロールで編集可能かどうかを指定します。アクセス権は、新しいレコードの作成などの操作や、ファイルのインポートやエクスポートなどのサービスにも適用されます。

アクセスルール

アクセス権にはユーザーロールとデータ構造が関係しますが、アクセスルールはデータ値に従ってアクセスを制限します。アクセスルールは、エンティティのオカレンスまたはフィールドに適用できます。フィールドが未定義のままの場合、アクセスルールによってすべてのオカレンスへのアクセスが決定されます。次の例では、アプリケーションのオカレンスへのアクセスを決定します。ラベルに「Secret」という単語を含むすべてのアプリケーションは、ビジネスユーザーに対して非表示になります。条件は、エンティティのルートから始まる XPath 述語として表現され、すべてのフィールドがパスのステップとして認識される必要があります。詳細については、「サポートされている XPath 構文」を参照してください。

ユーザーインターフェイスの構成

デフォルトホームページテンプレートの変更

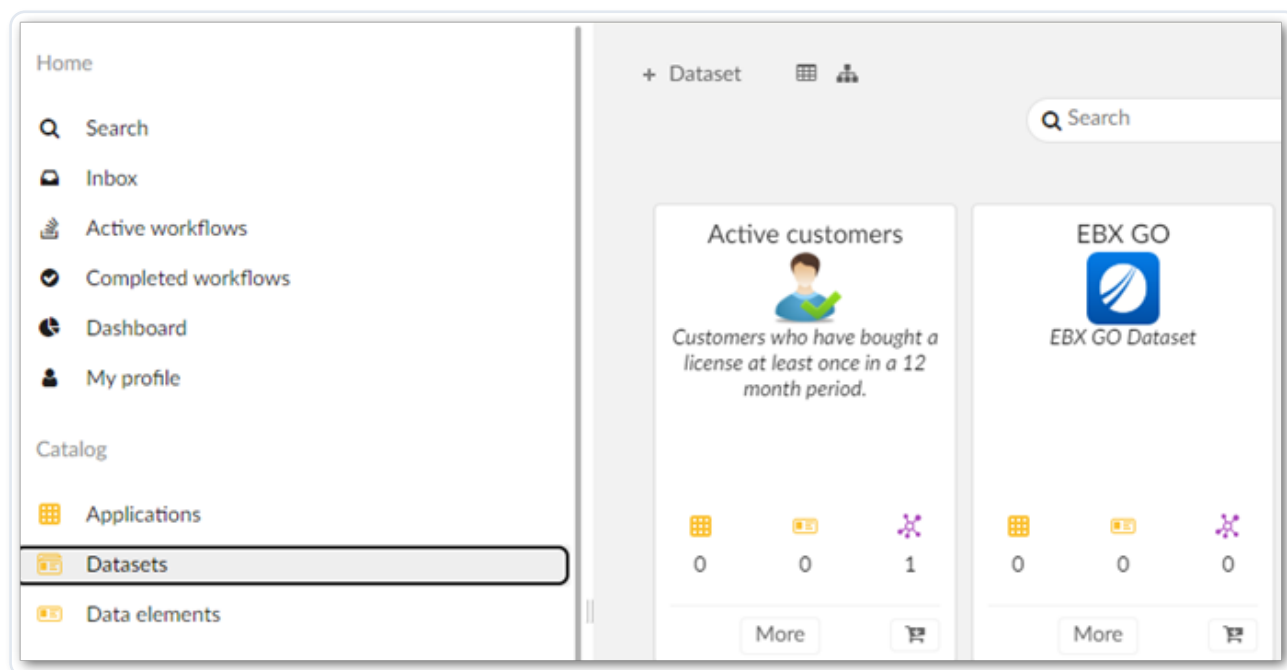
管理パースペクティブで、を選択します。

カスタマイズできるデフォルトのホームページテンプレートが表示されます。ページを変更するには、歯車アイコンをクリックします。ホームページウィジェットを追加、削除、編集、並べ替え

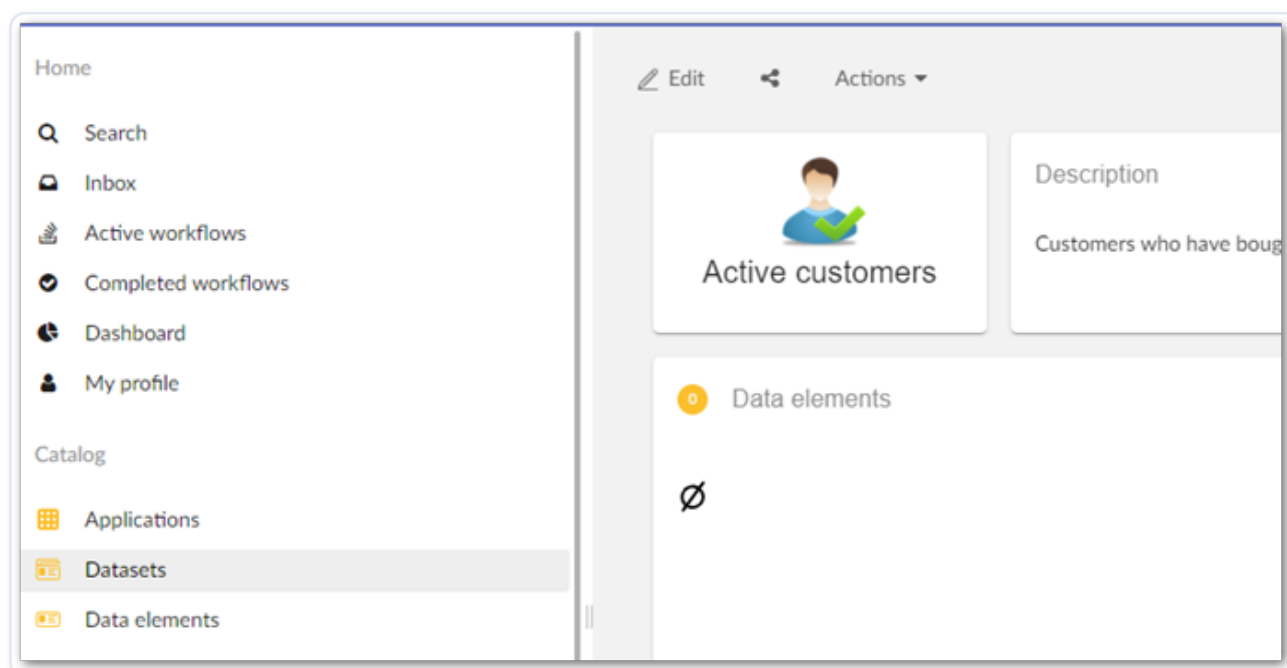
ることができます。ホームページのカスタマイズの詳細については、[ホームページ](#)を参照してください。

ワークフロー

エンティティの新しいレコードを作成することを許可されているすべてのユーザーは、プラス記号の後にエンティティの名前が付いたアイコンにアクセスできます。このアイコンは、カード、テーブル、階層ビューなどのグローバルビューの左上にあります。



同じロジックに従って、ユーザーはアイコンが表示されたときにエンティティのレコードを更新できます。このアイコンを選択すると、編集画面にリダイレクトされます。



デフォルトでは、これらのアクションは制御されず、影響を受けるレコードの表示を許可されたユーザーはこれらの変更を表示できます。制御を強制するには、一部のプロファイルによって操作される作成または更新をレビュー、承認、および完了するアクターを関与させるワークフローをアクティブ化できます。ワークフローをアクティブ化するには、パースペクティブを使用します。

Identifier	Entity	Operation
2	Application	Create
3	Data Element	Create
4	Data Element	Update
5	Dataset	Update

作成および更新操作のワークフローをアクティブ化できます。レコードの複製は作成と同じと見なされるため、作成のワークフローをアクティブ化すると、複製にも適用されます。

作成時にワークフローをアクティブ化すると、作成アイコンによって、ワークスペースでのレコードの作成から成るワークフローの最初のステップにユーザーがリダイレクトされます。このワークスペースは、ワークフローが完了するまで、ワークフローで操作されたすべての変更を分離します。ワークフローが完了すると、作成または更新が検証されて全員に表示されるか、キャンセルされます。各操作とエンティティごとに異なるアクターを定義できます。

各アクターをユーザーロールのコレクションとして定義できます。これらのロールは静的または動的です。静的ロールはディレクトリで定義され、ユーザーに直接影響します。動的ロールはデータから解決され、ワークフローのサブジェクトがエンティティのオカレンスであるという条件付きです。動的ロールは、たとえば、この発生の所有者であり、これは静的ロールを参照する**グループ**自体への参照としてオカレンス自体に設定されます。動的ロールは、データに応じて静的ロールを動的に割り当てることです。詳細については、**動的ロール**を参照してください。

アクターにロールが定義されていない場合、または動的ロールのみが定義されているが後者が静的ロールにマップされていない場合、レビュー、承認、または最終処理の動作はスキップされます。

レビュー担当者は、オカレンスの作成または更新をレビューして、必要な情報がすべて存在し、承認者に提出できるほど質が高いことを確認します。そうでない場合は、必須のコメントを残して変更を拒否できます。ワークフローの開始者は、操作をキャンセルするか、レビュー担当者のコ

メントに従って必要な変更を加えて新しいレビューを依頼するかのタスクを受け取ります。このレビューは、多くのロールに提供される場合でも、タスクを実行する 1 人のユーザーによってのみ実行されます。

承認者は、操作を承認または拒否する必要があります。多くのロールに提供されており、各ロールの 1 人のユーザーがこのアクションを実行する必要があります。そのうちの 1 人が操作を拒否した場合、ワークフローの開始者は、操作をキャンセルするか、承認者のコメントに従って必要な変更を加えて新しいレビューを依頼するタスクを受け取ります。すべての承認者によって承認されると、操作を完了できます。

追加のアクションを実行する必要があるが、以前のアクターに許可されていない場合は、ユーザーが操作を完了するために関与できます。

プロビジョニングワークフローの構成

プロビジョニングワークフローは、3 番目の種類の承認者である資産所有者を構成できることを除いて、同じパターンに従います。

このワークフローの最終目的は、収集された資産からビューを構成することです。これらの各資産には、所有者が定義されている場合と定義されていない場合があります。最後に、ビューの作成を承認し、資産の使用を承認するために関与できます。

Actions

Identifier 1

Activated Yes No ✕

Authorized users profiles 1. [all profiles]

+

Reviewer ▼

Reviewer dynamic role(s) +

Reviewer static role(s) 1. [Data Analyst]

+

Approver(s) ▼

Approver(s) dynamic role(s) +

Approver(s) static role(s) 1. [Application Architect]

+

Asset owners Yes No ✕

Finalizer ▼

Finalizer dynamic role(s) +

Finalizer static role(s) 1. [Data Architect]

+

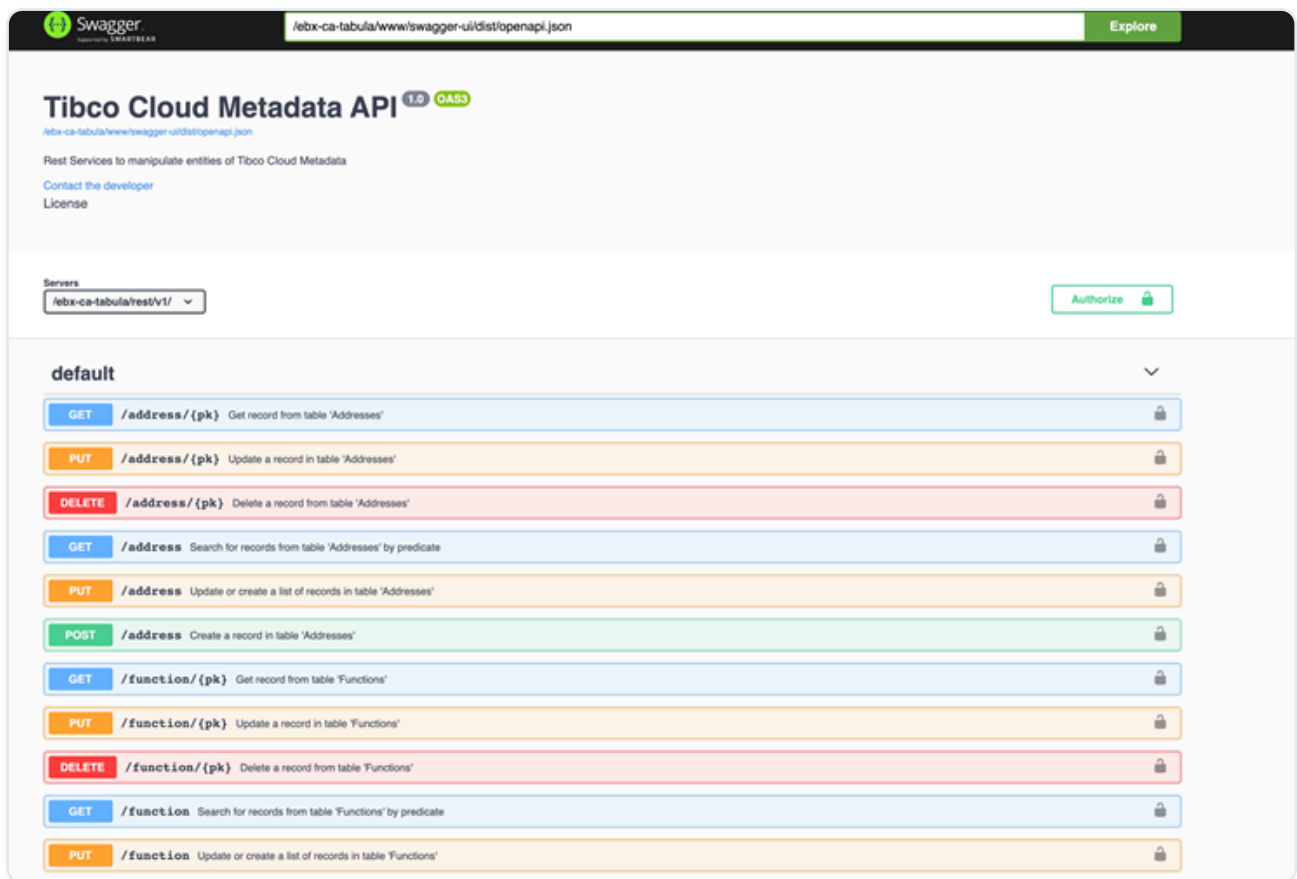
Save **Revert**

Web サービスと API

****注意：****これらは高度な操作のため、注意して使用する必要があります。

Swagger Web サイトでは、TIBCO Cloud Embedded Web サービスを使用できます。

アプリケーションコンテキストルートから、ブラウザーで `[CONTEXT-ROOT]/ebx-ca-tabula/api-documentation` に移動します。

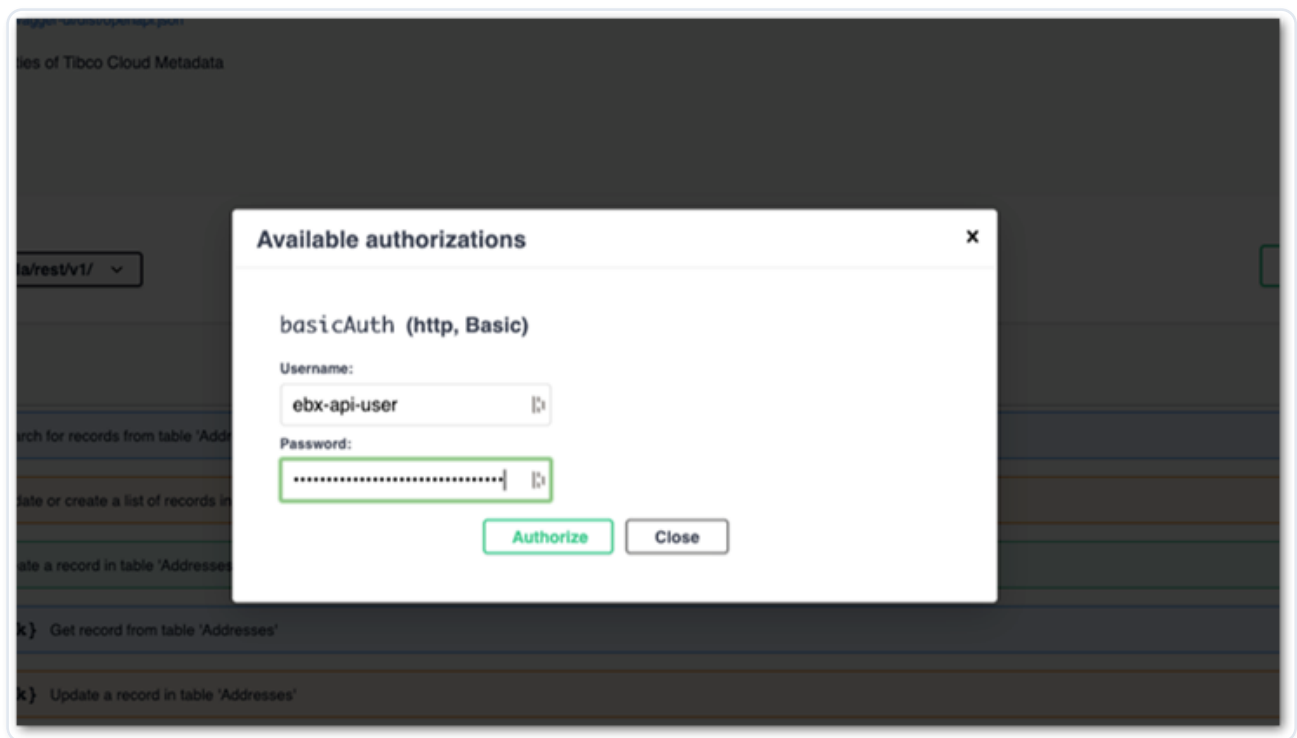


これらのサービスは認証なしでは利用できません。

認証を有効にするには、 ボタンをクリックし、資格情報を入力します。

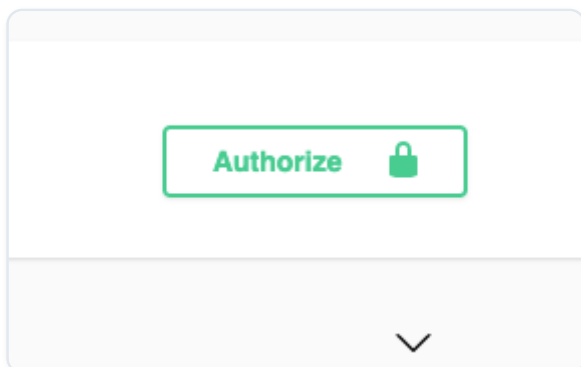
現在、基本認証のみが利用可能であるため、テクニカルユーザーの資格情報を使用する必要があります。

このテクニカルユーザーに適用された権限 (プロファイルを含む) は、API 操作にも適用されます。



テクニカルユーザーを定義するには、[テクニカルユーザー](#)を参照してください。

接続すると、キーロックは次のように表示されます。



次に、実行する操作を選択します。構文の例は、Swagger インターフェイスに示されています。

GET**/asset** Search for records from table 'Assets' by predicate

Search for records from table 'Assets' by predicate

Parameters

Name	Description
------	-------------

predicate string (<i>query</i>)	predicate
---	--

limit integer (<i>query</i>)	limit
--	------------------------------------

offset integer (<i>query</i>)	offset
---	-------------------------------------

選択をクリックすると、結果 (HTTPステータス+JSON レスポンス構造) を取得できます。

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
```

これらの API は、あらゆるエンタープライズサービスバスまたは統合プラットフォームから同じ認証および権限管理で使えます。

Metadata Agent

- TIBCO® メタデータエージェントのインストール
- TIBCO TDV メタデータアダプタのインストール
- TIBCO® EBX® メタデータアダプタのインストール
- TIBCO® メタデータエージェントフレームワーク
- TIBCO® メタデータエージェントの管理
- 技術ユーザー
- TIBCO® メタデータエージェント
- メタデータエージェントによって管理される項目
- 収集の構成

TIBCO® メタデータエージェントのインストール

- インストールと構成
 - 前提条件
 - セキュリティとクラウドアクセス
 - インストールプロセス
 - パッケージ
 - プロパティファイル
 - TIBCO® メタデータエージェントプロパティテンプレート
 - プロパティの構成
 - データソースタイププロパティファイル
- EBX® メタデータ管理アプリケーション接続暗号化
- エージェントの実行 (スタンドアロン Java アプリケーション)
- トラブルシューティング

インストールと構成

TIBCO® メタデータエージェントは、jar ファイルとして提供され、プロパティファイルを使用して構成される Java アプリケーションです。その主な目的は、サーバーにインストールされたソースシステムインスタンス (特定のアダプタ経由) と EBX® メタデータ管理アプリケーション間の対話を確立することです。1 つのエージェントをソースシステムの 1 つのインスタンスに接続できま

す。EBX® メタデータ管理アプリケーションは、異なるデータソースを管理する複数のエージェントへの接続をサポートします。

エージェントには、さまざまなデータソースからのメタデータ抽出をサポートする 1 つまたは複数のアダプタを設定できます。アダプタは、特定のデータソースタイプに対して 1 つまたは複数のメタデータ機能を保証する独立した Java アプリケーションです。

- データソース管理
- メタデータの収集
- データサンプリング
- メタデータのプロビジョニング
- リネージ抽出

エージェントを構成する前に、**管理**の観点からテクニカルユーザーを作成する必要があります。詳細については、[テクニカルユーザー](#)を参照してください。

注意：作成されたテクニカルユーザーには、自動的に ebx- というプレフィックスが付けられます。

前提条件

TIBCO® メタデータエージェントには以下が必要です。

- JRE 17 LTS 以上

TIBCO® TDV メタデータアダプタを使用する場合 ([TIBCO TDV メタデータアダプタのインストール](#)を参照)

- TIBCO® Data Virtualization バージョン 8.3 以上

セキュリティとクラウドアクセス

EBX® (CIC Cloud) にアクセスし、EBX® メタデータ管理アプリケーションが TIBCO® メタデータエージェントにアクセスできるようにするには、次のインフラストラクチャーの注意事項を考慮する必要があります。

- TIBCO® メタデータエージェントホスティングサーバーはインターネットに公開されている必要があります。
- 証明書の管理を容易にするために、TIBCO® メタデータエージェントホスティングサーバーの DNS 名を定義することをお勧めします。

- TIBCO® メタデータエージェントホスティングサーバーは、CA 署名証明書 (自己署名証明書ではない) を使用して、HTTP (SSL 経由の HTTP) 経由でアクセスする必要があります。
- TIBCO® メタデータエージェントの外部ポートは、リバースプロキシ (IIS、Apache、Nginx など) を使用するポート443 (デフォルトの HTTPs ポート) にする必要があります (ma.http.ws-url は、インターネットからアクセス可能なエージェントへのリバースプロキシ URL に設定する必要があります)。
- TIBCO® メタデータエージェントの内部ポートは、マシン上の使用可能な任意のポートで構成でき (エージェント構成プロパティの ma.http.port を参照)、リバースプロキシ構成で使用する TIBCO® メタデータエージェントをインターネットに公開できます。

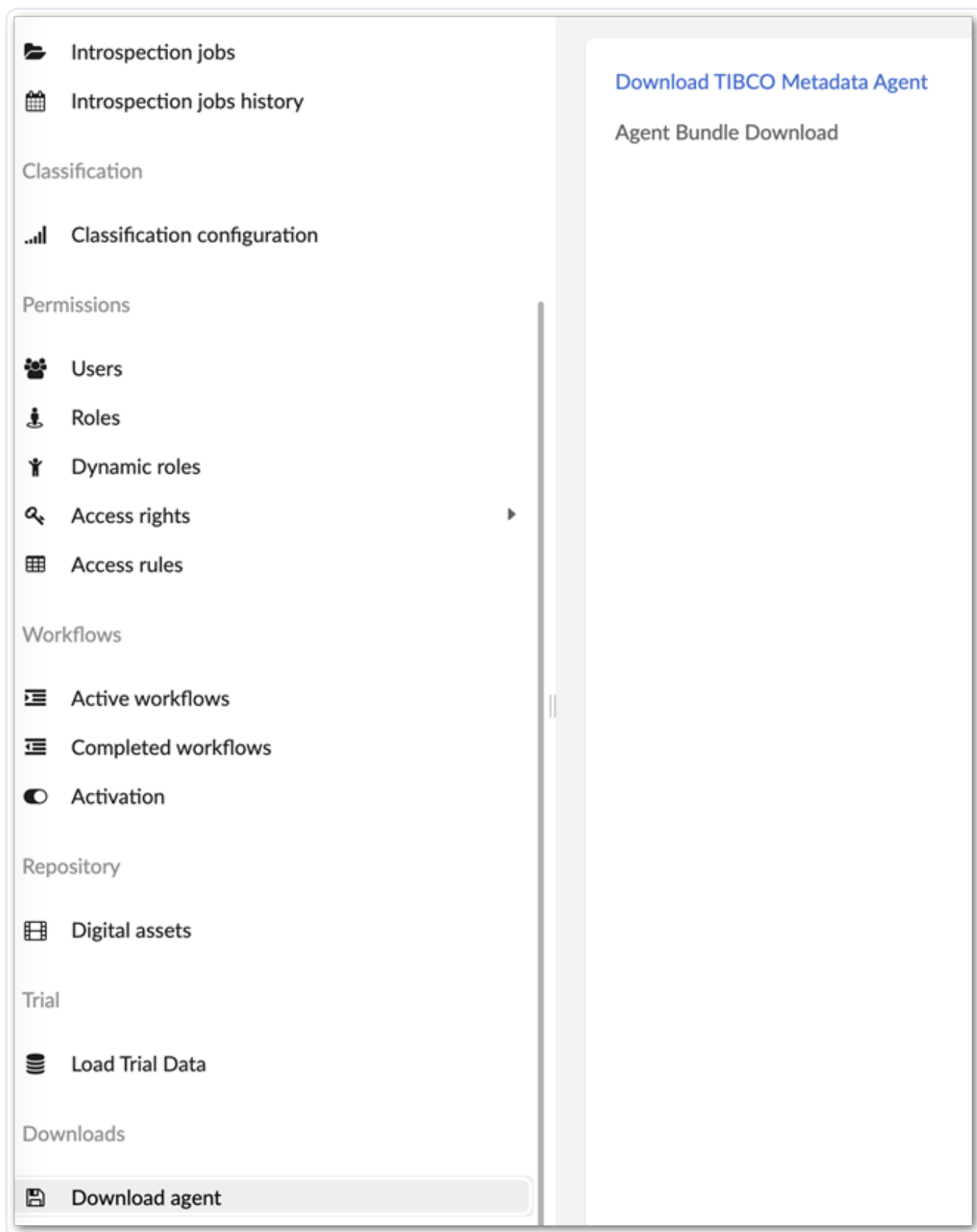
インストールプロセス

次の手順では、システム上で TIBCO® メタデータエージェントを実行する方法について説明します。

1. **メタデータ管理** パースペクティブから TIBCO® メタデータエージェントパッケージをダウンロードします。
2. システム上で解凍します。
3. プロパティ構成セクションで以下に示すように、プロパティテンプレートに入力します。一部のプロパティでは、EBX® にログインしたり、EBX® 管理者から情報を取得したりする必要があります。あることに注意してください。
4. ソースシステムに必要な EBX® メタデータ管理アプリケーションアダプタをダウンロードしてインストールします。
5. 各アダプタのプロパティテンプレートを入力します。
6. メインエージェントプロパティテンプレートで必要なアダプタをアクティブ化します。
7. 以下の実行セクションの説明に従って、TIBCO® メタデータエージェントを起動します。

パッケージ

エージェントはスタンドアロンの Java アプリケーション (jar ファイル) として利用できます。パースペクティブからダウンロードできます。



プロパティファイル

TIBCO® メタデータエージェントの主なパッケージには、次のプロパティファイルが含まれています。

- エージェントの構成のための application-metadata-agent.properties

- アダプタごとにサポートされるデータソースタイプを構成するための application-datasources.properties
- TIBCO® TDV メタデータアダプタを構成するための application-tdv.properties (オプション)

TIBCO® メタデータエージェントプロパティテンプレート

```
#####

#####

##### ##### 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
```

プロパティの構成

TIBCO® メタデータエージェントを起動する前に、プロパティファイルの情報を入力して、EBX® メタデータ管理アプリケーションへの接続を確認する必要があります。

ユーザーは、以下の表に記載されていないがプロパティファイルに存在するプロパティ、特に EBX® メタデータ管理アプリケーション操作に構成された特定の URL または H2 データベース JDBC URL を変更することは強く推奨されません。構成ファイル内の特定のプロパティのみを保持するには、バージョン 4 より前のエージェントから古い特定のプロパティを削除することをお勧めします。

プロパティ名	説明
TIBCO EBX® のメタデータ接続	
<code>ma.metadata.context</code>	EBX® メタデータ管理アプリケーション Advanced Data Services の REST エンドポイント。想定される形式： https:///
<code>ma.metadata.username</code>	メタデータテクニカルユーザー名。この情報を受け取るには、管理者に問い合わせてください。
<code>ma.metadata.password</code>	メタデータテクニカルユーザーのパスワード。この情報を受け取るには、管理者に問い合わせてください。
<code>ma.metadata.public-key</code>	EBX® メタデータ管理アプリケーションのテクニカルユーザーに関連付けられた公開鍵 (JWT)。この情報を受け取るには、管理者に問い合わせてください。

プロパティ名	説明
<code>ma.metadata.label</code>	このエージェントのメタデータに表示されるユーザーフレンドリーなラベル。
<code>ma.metadata.useEncryption</code>	<code>ma.tdv.datasource.password</code> 、 <code>ma.tdv.soap.password</code> 、 <code>ma.metadata.password</code> 、 <code>ma.proxy.proxyPassword</code> に暗号化を使用するかどうかを定義します。有効にした場合、すべてのパスワードは TIBCO メタデータエージェントツールを使用して暗号化する必要があります (以下を参照)。
エージェントのプロパティ	
<code>ma.http.port</code>	エージェントが着信 REST 接続をリッスンするポート。
<code>ma.http.ws-url</code>	エージェントの REST エンドポイントの URL。エージェントに HTTPS を設定する必要があります (たとえば、リバース プロキシを使用します。そうしないと、EBX® メタデータ管理アプリケーションは接続できません)。また、クラウド (インターネット) からエージェントのホストとポートへの着信接続を許可する必要があります。
エージェントのログプロパティ	
<code>ma.logging.path</code>	エージェントのログが保存されるフォルダー。エージェントにはこのフォルダーへの書き込み権限が必要です。
<code>ma.logging.level.root</code>	エージェントが実行される技術フレームワークによって生成される技術ログのログレベル。
<code>ma.logging.level.agent</code>	エージェントによって生成される機能ログのログレベル。
<code>ma.logging.level.soap.sent</code>	エージェントから TDV に送信される SOAP メッセージのログレベル。デフォルトでは、

プロパティ名	説明
	ログに記録されません。デバッグやその他の目的でログに記録する場合は、このプロパティを TRACE に設定します。
<code>ma.logging.level.soap.received</code>	TDV からエージェントに受信される SOAP メッセージのログレベル。デフォルトでは、ログに記録されません。デバッグやその他の目的でログに記録する場合は、このプロパティを TRACE に設定します。
エージェントプロキシプロパティ	
<code>ma.proxy.useProxy</code>	エージェントに HTTP(s) プロキシを使用する場合は、この行のコメントを解除します。
<code>ma.proxy.proxyHost</code>	プロキシホスト。
<code>ma.proxy.proxyPort</code>	プロキシポート。
<code>ma.proxy.proxyAuthentication</code>	プロキシの使用に認証が必要な場合は、このプロパティを true に設定します。
<code>ma.proxy.proxyUsername</code>	プロキシに使用するユーザー名。
<code>ma.proxy.proxyPassword</code>	プロキシに使用するパスワード。
エージェントアダプタのプロパティ	
<code>agent-adapter.base.package</code>	アダプタクラスをスキャンするパッケージのカンマ区切りリスト。
<code>agent-adapter.active-profiles</code>	プロファイルのカンマ区切りリスト (アダプタの技術名に対応)。

これらのプロパティが設定されると、コマンドラインまたはその他の適切なツールを使用して、エージェントを Java アプリケーションとして起動できます (このページの下記を参照)。

構成ファイルの名前は `application.properties` にする必要があります。

最初の起動時に、エージェントは EBX® メタデータ管理アプリケーション REST API に登録を問い合わせ、2 つのアプリケーション間のすべての情報交換で使用する GUID を取得します。起動後にエージェントのログ ファイルに「Started MetadataAgentApplication in xx seconds」という

メッセージがあることを確認します。これは、起動が正しく行われ、エージェントが実行中であることを示します。このメッセージが表示されない場合は、エージェントの起動を妨げている問題を説明するエラーメッセージがログファイルにあるかどうか確認する必要があります。

登録されると、エージェントはTIBCO® メタデータエージェントで説明されているように、EBX® メタデータ管理パースペクティブに表示されます。

データソースタイプのプロパティファイル

プロパティ ファイル `application-datasources.properties` を使用すると、エージェントはサポートされているデータソースタイプ、メタデータ管理機能、および使用するアダプタ間の接続を確立できます。

このファイル内のプロパティは、パターン

`datasource.operation\_name.data\_source\_type=adapter\_profile` に従う必要があります。サポートされている操作名は次のとおりです。

- `dsmanagement`
- `harvesting`
- `sampling`
- `provisioning`
- `lineage`

データソースタイプ名は、特殊文字を使用せず、すべて小文字の英数字名としてフォーマット設定する必要があります。EBX® メタデータ管理アプリケーションに表示されるデータソースタイプのラベルを定義することもできます。操作に対して定義できるアダプタは、データソースタイプ 1 つだけです。

構成パターンは次のとおりです。

```
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>=
```

デフォルトでは、このファイルには **TIBCO TDV メタデータアダプタ**のインストールでサポートされるデータソースタイプの構成が含まれています。

EBX® メタデータ管理アプリケーション接続暗号化

`ma.metadata.useEncryption` が有効になっている場合は、エージェントを起動する前に、`ma.tdv.datasource.password`、`ma.tdv.soap.password`、`ma.metadata.password`、および `ma.proxy.proxyPassword` を `application.properties` に暗号化する必要があります。

- 前提条件として、対応する EBX® メタデータ管理アプリケーションのテクニカルユーザーに対応する `ma.metadata.public-key` が `application.properties` で定義されている必要があります。
- TIBCO® メタデータエージェントツールを使用して元のパスワードを暗号化します。ターミナルで起動します。

```
java -jar metadata-agent-tools.jar MY_PLAIN_PASSWORD
```

以下の出力サンプルを参照してください。

- 対応するパスワードプロパティごとに、暗号化された値を `application.properties` にコピーして貼り付けます。

提供される出力は次のとおりです。

```
`*****`

`Value to encrypt is MY_PLAIN_PASSWORD`

`*****`

`Encrypted value is : I7U0EP0HtF+J4EYPTPRxR0++QG4hoXFe87uSPluI1lbi`

`*****`
```

注意：この機能を有効にすると、暗号化されたパスワードは JWT キーの更新ごとに再暗号化する必要があります。

エージェントの実行 (スタンドアロン Java アプリケーション)

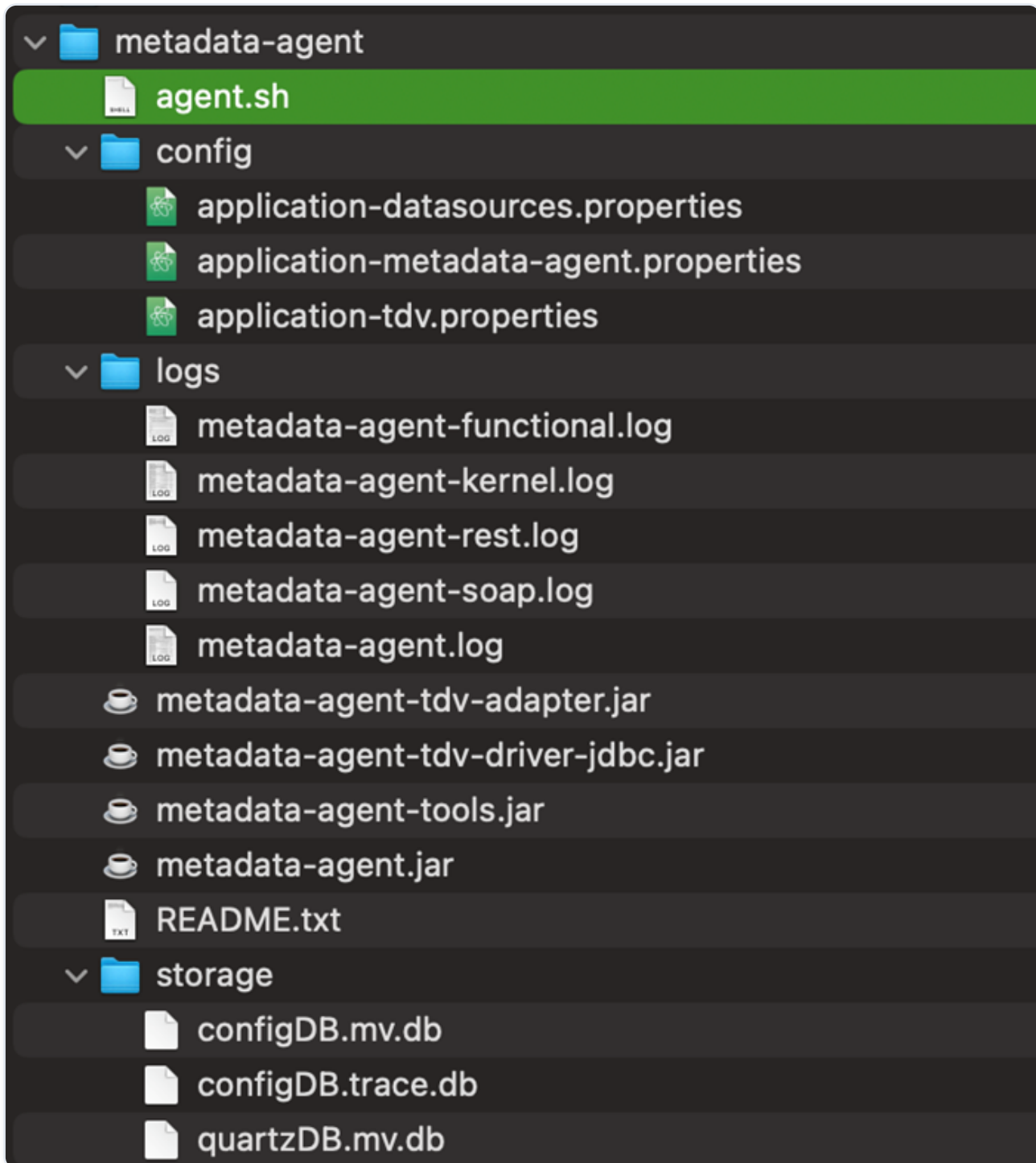
エージェントを実行するには、ランチャーを構成して、エージェントが配置されているディレクトリの下に config ディレクトリに構成ファイルを追加する必要があります。

ログとストレージ ディレクトリは、エージェント自体によって構造の直下に作成されます。

ストレージの下に、エージェントのローカルデータベース用 (configDB.mv.db)、スケジュール用 (quartzDB.mv.db) の 2 つのファイルが作成されます。以前の構成でこれらのファイルに異なる名前を使用していた場合は、現在のエージェントを停止した後、ファイルシステムでこれらのファイルの名前を適宜変更してください。

エージェントを起動するには、次のようにシェルフファイルを作成します (この例では、jar は / metadata-agent ディレクトリに配置されます)。

まだ設定されていない場合は、`-Dfile.encoding=UTF-8` を使用して、JVM のデフォルトのエンコーディングを UTF-8 として定義する必要があります。



Linux の例 (TDV アダプタが有効) : `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 &`

Windows の例 (TDV アダプタが有効) : `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`

起動すると、エージェントのプロセス ID を含む `application.pid` ファイルが作成されます。このプロセスを強制終了すると、エージェントを停止できます。 `kill $(cat application.pid)`

エージェントは、専用のエンドポイントで HTTP POST リクエストを使用してシャットダウンすることもできます： `curl -X POST http://<agent_host>:<agent-port>/manage/shutdown`

トラブルシューティング

次の表は、考えられる 2 つのエラーの原因と解決方法を示しています。

エラーメッセージ	原因と解決策
EBX® メタデータ管理アプリケーションとのやり取りでエラーが発生しました: { "localizedMessage":"HTTP エラー 401。", "messages": }	このエラーは、エージェントがプロパティ <code>ma.metadata.username</code> および <code>ma.metadata.password</code> で提供される資格情報を使用して EBX® メタデータ管理アプリケーションに接続できない場合に表示されます。
EBX® メタデータ管理アプリケーションへの接続に失敗しました。「http://localhost:8081/ebx-ca-tabula-tdv/tdv-rest/health」の GET リクエストで I/O エラーが発生しています。	このエラーメッセージは、EBX® メタデータ管理アプリケーションが <code>ma.metadata.context</code> で応答していないことを示しています。EBX® URL が正しいこと、およびエージェントがインストールされているマシンからアクセスできることを確認してください (ファイアウォールとプロキシ設定を含む)。

TIBCO TDV メタデータアダプタのインストール

- 概要
 - データソースの作成
 - 公開データソースの表示
 - 一般的なエラーメッセージ
- TDV 接続権限
- プロパティテンプレート
 - プロパティの構成
- トラブルシューティング

概要

このアダプタを使用して、メタデータエージェントを TIBCO® Data Virtualization (TDV) サーバーに接続し、収集機能を使用して管理対象データソースを EBX® メタデータ管理アプリケーションに送信します。

2 つの TDV アプリケーションパーツをインストールする必要があります。

- TDV Sever - TDV のサーバー部分。システムからメタデータを収集するには、TDV サーバーのインストールが必須です。
- TDV Studio - TDV のクライアント部分。データソースを作成し、TDV からローカルでパブリックビューにアクセスする場合にのみインストールする必要があります。

インストールの要件とプロセスについては、TDV に付属の『インストールガイド』を参照してください。

メタデータエージェントに必要な TDV バージョンは TDV 8.3 以降です。

EBX® メタデータ管理アプリケーションと組み合わせて使用できる機能の概要を以下に示します。

これらの機能に関するガイダンスについては、対応する TDV ドキュメントを使用することをお勧めします。

データソースの作成

EBX® メタデータ管理アプリケーションで既存のデータソースに関する情報を収集する場合は、まず TDV でコードを構成する必要があります。これは、TDV Studio の **新しいデータソース機能**を使用して実行できます。

作成の最初のステップでは、新しいデータソースのコネクターを選択するように求められます。

コネクターを選択すると、このデータソースのラベルと、TDV がシステム上の物理データソースにアクセスするための接続の詳細を入力するよう求められます。

EBX® メタデータ管理アプリケーションが自動的に実行できるため、新しいデータソースでイントロスペクションを実行する必要はありません。ただし、接続をテストし、TDV がシステム上の物理データソースにアクセスできることを確認するため、ではなく ボタンを使用してデータソースの作成を完了することをお勧めします。

データソースが TDV で正しく構成されると、メタデータエージェントに表示されるようになり、このソースのメタデータ収集を実行できるようになります。

公開データソースの表示

メタデータエージェントは、メタデータプロビジョニング機能を実行することで、資産とデータソースの公開ビューを作成できます。これにより、TDV Studio を使用してアクセスできる TDV のパブリックデータソースが作成されます。

パブリックデータソースは、TDV のすべてのユーザーが共有し、JDBC クエリを介して外部ソフトウェアに公開できるフォルダー **Composite Data Services** の下にある TDV で使用できます。メタデータエージェントは、メタデータプロビジョニングの起動時に定義されたラベルを使用して、パブリック データソースを **Composite Data Services - Databases** の下に作成します。

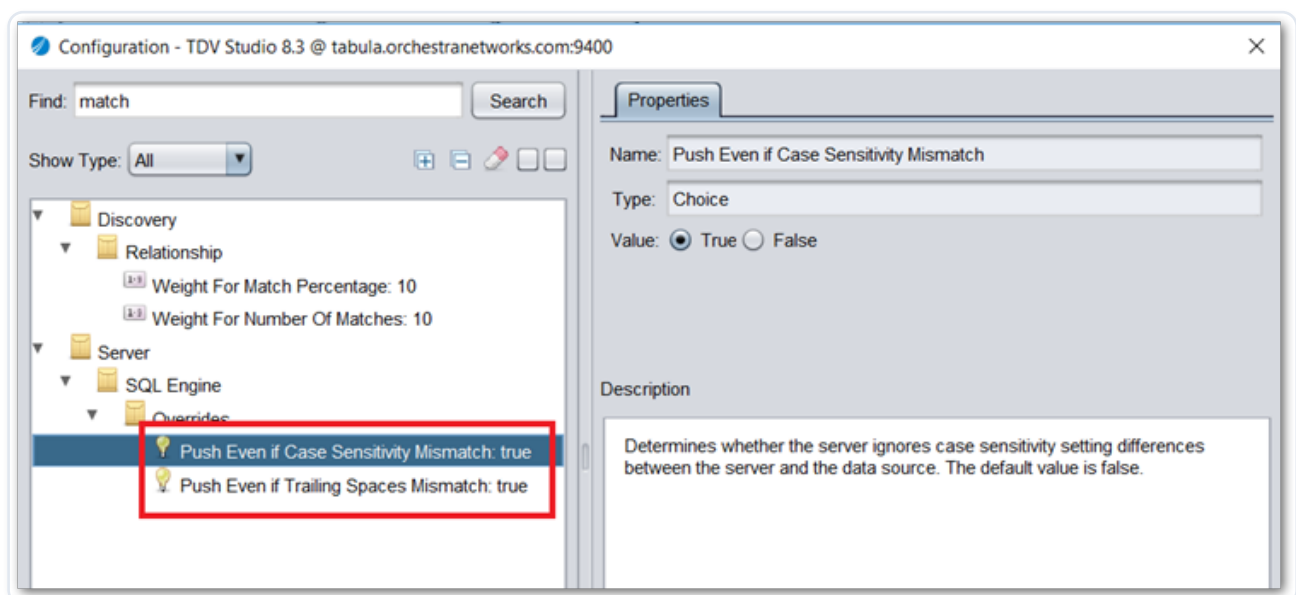
プロビジョニング対象として選択した資産に対応するデータソースとリソースは、このパブリックデータソースに集められ、そのメタデータは TDV Studio と、サードパーティソフトウェアがクエリできるシステムデータベースに表示されます。

一般的なエラーメッセージ

場合によっては、設定が誤っていたり互換性がなかったりして、エージェントと TDV 間のやり取りが失敗することがあります。以下は、TDV またはエージェント/EBX® メタデータ管理アプリケーション側で微調整することで解決できる一般的なエラーの一覧です。

エラーメッセージ/理由と解決策：データソースが上位関数をサポートしておらず、大文字と小文字の区別の設定が一致しないため、選択可能な EBX\HG_ACCOUNT.OP をプッシュできません。

このエラーメッセージは、特定のデータソースのデータのサンプリングまたはプロファイリング中に表示されることがあります。これを回避するには、TDV Studio - 管理 - 構成メニューに移動し、検索バーに *match* と入力します。設定を探し、*true* に切り替えます。**についても同じ設定にすることをお勧めします。これらの変更を保存して適用し、問題を解決します。



TDV 接続権限

エージェントは、次の 2 つの接続フローを使用して TDV インスタンスに接続することを許可される必要があります。

- プロパティ `ma.tdv.datasource.url` で定義された JDBC URL を使用した、TDV systcode データソースへの JDBC 接続。
- プロパティ `ma.tdv.soap.resources-url` で定義された URL を使用した SOAP 接続。

エージェントを起動する前に、これらのタイプの接続が TDV インスタンスで許可されていることを確認してください。

必要に応じて、[TIBCO® メタデータエージェントのインストール](#) の接続暗号化の詳細を参照してください。

プロパティテンプレート

以下にサンプルテンプレートを示します。

```
#####

#####

#####                                     #####

#####          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=
```

プロパティの構成

TIBCO® TDV メタデータアダプタを使用して TIBCO® メタデータエージェントを起動する前に、プロパティファイルの情報を入力して TDV への接続を確認する必要があります。

以下の表には記載されていませんが、プロパティファイルに存在するプロパティを変更することはお勧めしません。構成ファイルに特定のプロパティのみを残すために、バージョン 4 より前のエージェントから古い特定のプロパティを削除することをお勧めします。

プロパティ名の説明

TDV データベース接続

TDV SOAP 接続

プライベートプロパティ (特定の場合にのみ上書きする必要があります)

プロパティ名	説明
TDV データベース接続	
<code>ma.tdv.datasource.url</code>	エージェントを TDV の systcode データベースに接続するための複合データソース URL。想定される形式は <code>jdbc:compositesw:dbapi@<host>:<port>?domain=composite&dataSource=systcode</code> です。
<code>ma.tdv.datasource.username</code>	システムデータソースに対する読み取り権限を持つ TDV ユーザー名。
<code>ma.tdv.datasource.password</code>	対応するパスワード
TDV SOAP 接続	
<code>ma.tdv.soap.base-url</code>	TDV SOAP 操作の URL。想定される形式は https:// です。
<code>ma.tdv.soap.username</code>	SOAP 操作の読み取り/書き込み権限を持つ TDV ユーザー名
<code>ma.tdv.soap.password</code>	対応するパスワード

プロパティ名	説明
<code>ma.tdv.soap.datasources-root-folder</code>	プライベートデータソースを取得するためのルートフォルダー。このフォルダーとそのサブフォルダーがスキャンされます。複数のパスをカンマで区切って指定できます。
<code>ma.tdv.soap.new-datasources-folder</code>	エージェントが EBX® メタデータ管理アプリケーションから新しいプライベートデータソースを作成するフォルダー。
<code>ma.tdv.profiling.not-supported-types</code>	プロファイリング機能でサポートされていない SQL データ型のリスト。型は小文字のカンマ区切りでリストする必要があります。
<code>ma.tdv.profiling.not-supported-types-for-cardinality</code>	プロファイリングではサポートされているが、カーディナリティ関数ではサポートされていない SQL データ型のリスト。タイプは小文字のカンマ区切りでリストする必要があります。
プライベートプロパティ (特定の場合にのみ上書きする必要があります)	
<code>ma.tdv.soap.public-databases-folder</code>	パブリックデータソースを作成するためのフォルダー。メタデータプロビジョニング機能は、このフォルダーの下にパブリックコンテナを作成します。このフォルダーは、TDV / <i>services/databases</i> フォルダーを指している必要があります。このプロパティはパブリックプロパティで定義されておらず、TDV のアップグレードに内部構造の変更が含まれる場合を除き、上書きしてはなりません。
<code>ma.tdv.soap.introspect-result-timeout</code>	イントロスペクション タスク実行のタイムアウト (実時間 (秒) は、タイムアウト * 間隔として計算されます)。プロパティが指定されていない場合のデフォルト値は 600 秒です。このタイムアウトに達すると、エージェントは TDV でのイントロスペクション タスクの状態のチェックを停止し、進行中の収集要求に対してエラーを返します。

プロパティ名	説明
<code>ma.tdv.soap.introspect-result-interval</code>	エージェントが TDV のイントロスペクションタスクの状態をチェックする間隔 (秒単位)。このプロパティが指定されていない場合のデフォルト値は 5 秒です。
<code>ma.tdv.datasource.max-query-timeout</code>	TDV データソースのクエリ タイムアウトの間隔 (秒単位)。デフォルトでは 600 秒です。

これらのプロパティが設定されると、コマンドラインまたはその他の適切なツールを使用して、エージェントを Java アプリケーションとして起動できます (このページの下記を参照)。

起動するたびに、アダプタは TIBCO® Data Virtualization JDBC および SOAP 接続をテストします。

アダプタの起動が成功すると、TIBCO® **メタデータエージェント**で説明されているように、対応するエージェントの**メタデータ管理**パースペクティブに表示されます。

トラブルシューティング

エラーメッセージ	原因と解決策
TDV データソースの初期化に失敗しました：無効なログイン試行です。	このエラーは、ログインまたはパスワードが正しくないためにエージェントが TDV インスタンスに接続できない場合に表示されます。プロパティ <code>ma.tdv.datasource.username</code> および <code>ma.tdv.datasource.password</code> に設定した情報を確認してください。
TDV データソースの初期化に失敗しました：接続を完了できませんでした：接続がタイムアウトしました：詳細情報はありません	この種の問題は、さまざまな原因で発生する可能性があります。最も一般的な原因は次のとおりです。 - TDV データソース接続のホストとポートが正しくありません。 <code>ma.tdv.datasource.url</code> プロパティの値を確認してください。 - TDV サーバーでポートが開かれていません。<code>ma.tdv.datasource.url</code> プロパティで指定したポートがサーバーで開かれており、エージェントがインストールされているマシンからの接続が許可されていることを確認してください。 - ファイアウォールが、TDV サーバーとエージェントがインストールされ

エラーメッセージ	原因と解決策
	ているマシン間のアクセスをブロックしています。
TDV データソースの初期化に失敗しました： ドライバークラスを読み込めません	ドライバークラスがクラスパスに存在することを確認します。

TIBCO® EBX® メタデータアダプタのインストール

- 概要
- EBX® 接続権限
- プロパティテンプレート
 - プロパティの構成
- トラブルシューティング

概要

このアダプタを使用して、TIBCO メタデータエージェントを 1 つ以上の EBX® インスタンスに接続し、収集機能を使用して EBX® メタデータ管理アプリケーションのデータをソースします。

注意:

メタデータエージェント機能には、EBX® バージョン 6.2.0 が必要です。

EBX® 接続権限

エージェントが EBX® インスタンスに接続することを許可する必要があります。ターゲット EBX® インスタンスのユーザーとして接続を構成し、エージェントに適切な権限を付与します。

詳細については、[EBX® メタデータ管理アプリケーション接続暗号化](#)を参照してください。

プロパティテンプレート

以下にサンプルテンプレートを示します。

```
#####
```

```
#####
```

```

#####                                #####

#####                                #####
                                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
```

プロパティの構成

TIBCO EBX® メタデータアダプタを使用して TIBCO® メタデータエージェントを起動する前に、プロパティファイルの情報を更新して EBX® との接続を確認します。

注意:

以下の表に記載されていないが、プロパティファイルに存在するプロパティを変更することは強くお勧めしません。

プロパティ	説明
<code>ebx.instances</code>	エージェントが接続するインスタンスのコンマ区切りリスト。 <code>ebx.instances=instanceId1,instanceId2</code>
<code>ebx.base-url.instanceId</code>	EBX® インスタンスのベース URL。

プロパティ	説明
<code>ebx.username.instanceId</code>	エージェントがターゲット EBX® インスタンスに接続するユーザー名。
<code>ebx.password.instanceId</code>	エージェントがターゲット EBX® インスタンスに接続するときのユーザーパスワード。
<code>ebx.locale.instanceId</code>	オプション。EBX® インスタンスでローカライズされた情報を収集するために使用されるロケール。例： <code>ebx.locale.instanceId1=fr-FR</code>
<code>ebx.public-key.instanceId</code>	オプション。接続に暗号化を使用する場合は、公開鍵を入力してください
<code>ebx.useEncryption.instanceId</code>	オプション。暗号化されたパスワード接続を有効にします。 <code>ebx.useEncryption.instanceId1=true</code>

上記のプロパティを構成した後、コマンドラインまたはその他の適切なツールを使用して、エージェントを Java アプリケーションとして起動できます (このページの下記を参照)。

アダプタは起動するたびに TIBCO EBX® REST 接続をテストします。アダプタの起動が成功した場合は、EBX® メタデータ管理アプリケーションの**メタデータ管理**パースペクティブで、**TIBCO® メタデータエージェント**の説明に従って、対応するエージェントを確認できます。

トラブルシューティング

エラーメッセージ	原因と解決策
EBX® データソースの初期化に失敗しました：無効なログイン試行です。	このエラーは、ログインまたはパスワードが正しくないためにエージェントが EBX® インスタンスに接続できない場合に表示されます。プロパティ <code>ebx.instance.username</code> および <code>ebx.instance.password</code> に設定した情報を確認してください。
EBX® データソースの初期化に失敗しました：バージョンはサポートされていません	この種の問題は、互換性のないバージョンの EBX® に接続しようとすることで発生します。

エラーメッセージ	原因と解決策
	このアダプタは、EBX® 6.2.0 以降とのみ互換性があります。
EBX® データソースへの接続に失敗しました	これには複数の原因が考えられますが、最も可能性が高いのは、 <i>ebx.instance.url</i> プロパティのエラー、またはルーティングの問題です。

TIBCO® メタデータエージェントフレームワーク

- 概要と目的
- メタデータ抽出機能の拡張
 - 推奨されるアダプタプロジェクト構造
 - TIBCO® メタデータエージェントフレームワークへのアダプタの登録
 - 主な機能の実装
 - 例外管理
 - ロギング

概要と目的

TIBCO® メタデータエージェントフレームワークを使用すると、顧客は特定のアダプタを通じて TIBCO® メタデータエージェント内の特定のデータソースタイプのサポートを実装できます。

アダプタは、特定のデータソースタイプに対して 1 つまたは複数の EBX® メタデータ管理機能を保証する個別の Java アプリケーションです。

- データソース管理
- メタデータの収集
- データサンプリング
- メタデータのプロビジョニング
- リネージ抽出

TIBCO® メタデータエージェントフレームワークには次のエレメントが含まれます。

- 特定のデータソースタイプに対して特定のメタデータ管理機能をカスタマイズするために実装する必要がある Java ベースのインターフェイス
- 機能が実装されていることを示すために使用する必要がある Java アノテーション

- アダプタをアクティブ化し、このアダプタとサポートされているデータソースタイプ/EBX® メタデータ管理機能との関係を定義できるプロパティ
- アダプタからエージェントに国際化されたエラーを適切に発生させるための拡張可能な例外およびエラーコード。

メタデータ抽出機能の拡張

既存のメタデータ抽出機能を拡張するには、定義されたインターフェイスコントラクトに従って Java プログラミング言語を使用してアダプタを実装する必要があります。

アダプタは次の規則に従う必要があります。

- Java アプリケーション (jar) ファイルとしてコンパイルされる
- 登録サービスの `addAdapterInfos()` メソッドを実装して、エージェントに必須の登録データを提供する
- インターフェイスコントラクトと必須のアノテーションを使用した少なくとも 1 つのメタデータ抽出サービスを実装する
- エージェントプロパティファイルに登録される。

メタデータエージェントは Spring Boot 2.5.4 フレームワークを活用します。アダプタ実装には、サービス用の Spring アノテーション (`org.springframework.stereotype.Service`) を含める必要があります。

推奨されるアダプタ プロジェクト構造

アダプタは、担当開発チームにとって快適な方法で開発できます。以下は、Java プロジェクトの Maven 依存関係管理とアーキテクチャのベストプラクティスに基づいた推奨構造です。

```

v tcmd-agent-example
  > tcmd-agent-example-bundle
  > tcmd-agent-example-common
  > tcmd-agent-example-conf
  > tcmd-agent-example-service
  pom.xml
  
```

メインプロジェクトには、3 つのモジュールプロジェクトと 1 つの構成プロジェクトが含まれています。

モジュール名	説明
tcmd-agent-example-bundle	アダプタを 1 つの jar ファイルにアセンブルする方法を定義し、依存関係を簡単な方法で管理できるようにするバンドルプロジェクト。
tcmd-agent-example-common	アダプタの共通ライブラリコード。通常、これはエラー管理フレームワーク、いくつかの共通プロパティとユーティリティ、および i18n リソースをホストします。
tcmd-agent-example-conf	アダプタのプロパティと構成ファイルをホストするプロジェクトフォルダー。
tcmd-agent-example-service	アダプタ内にエージェントサービスを実装するコードを作成します。

TIBCO® メタデータエージェントフレームワークへのアダプタの登録

TIBCO® メタデータエージェントフレームワークにアダプタを登録するには、次の手順に従う必要があります。

1. 登録サービスを実装する

アダプタの登録サービスでは、専用のエージェントインターフェイス

`com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterRegistrationService` サービスとそのメインメソッド `addAdapterInfos()` を実装する必要があります。

このメソッドを使用すると、アダプタ情報オブジェクト

`com.tibco.ebx.ca.tabula.shared.agent.dto.AdapterDTO` を作成できます。このオブジェクトには、アダプタに関する次の情報を指定する必要があります。

プロパティ	説明
name	アダプタの技術名。実装アノテーションとエージェントのプロパティ <code>agent-adapter.active-profiles</code> に使用する値と一致する必要があります。
version	アダプタの実装バージョン。
provider	使用者の会社名。

プロパティ	説明
productVersion	サポートされているメタデータプロバイダーの対応するバージョン。
description	EBX® に表示されるアダプタの説明。

2. 適切なアノテーションを付ける

TIBCO® メタデータエージェントフレームワークによって認識されるためには、実装されていると考えられる各サービスクラスに、アダプタの技術名 (エージェントのプロパティ `agent-adapter.active-profiles` およびアダプタ情報オブジェクトの属性 `name` で提供される) を値として持つ

`com.tibco.ebx.ca.tabula.agent.common.annotation.MetadataAdapterProfile` でアノテーションを付ける必要があります。

実装されていると考えられるサービス内の各操作には、

`com.tibco.ebx.ca.tabula.agent.common.annotation.MetadataOperationImplemented` のアノテーションを付ける必要があります。

3. メインエージェントのプロパティにアダプタ名とプロパティを追加する

エージェントがアダプタをロードする必要があることを示すには、次のプロパティを変更する必要があります。

```
#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. アクティブプロファイルにアダプタ名を追加する

エージェントの Java 起動構成を編集して、アダプタ名をプロファイルとして追加します。

```
-Dspring.profiles.active=tdv,myadapter
```

5. アダプタ jar とオプションのプロパティをエージェントの CLASSPATH に追加します。

アダプタ jar と、オプションでコードに必要なプロパティ/構成ファイルをエージェントの `CLASSPATH` に展開します。

TIBCO® メタデータエージェントのインストールの説明に従って、`application-datasources.properties` ファイルで適切なデータソースタイプに対してアダプタをアクティブ化します。

主な機能の実装

アダプタによって実装できるメタデータ機能は次のとおりです。

- データソース管理
- メタデータの収集
- データプロファイリング
- データサンプリング
- メタデータのプロビジョニング
- リネージ抽出

一般的な実装規則

主要なメタデータ機能を実装する各カスタムサービスは、次の規則に準拠する必要があります。

1. 実装すると考えられる各サービスクラスには、アダプタの技術名 (エージェントのプロパティ `agent-adapter.active-profiles` およびアダプタ情報オブジェクトの属性 `name` で指定) を値として持つ `com.tibco.ebx.ca.tabula.agent.common.annotation.MetadataAdapterProfile` でアノテーションを付ける必要があります。
2. 各サービスクラスには、Spring `org.springframework.stereotype.Service` アノテーションを付ける必要があります。
3. 実装されていると考えられるサービス内の各操作には、`com.tibco.ebx.ca.tabula.agent.common.annotation.MetadataOperationImplemented` のアノテーションを付ける必要があります。
4. アダプタによって作成されたタイプ `com.tibco.ebx.ca.tabula.shared.agent.landing.dto.MetadataExchangeDTO` の各オブジェクトに対して、少なくとも `sourceId`、`sourceType`、`datasourceType`、および `name` 属性を設定する必要があります。

データソース管理

データソース管理機能は、メタデータプロバイダーシステム内のデータソースに対する次の操作をカバーします。

- 利用可能なすべてのデータソースの読み取り
- 新しいデータソースの作成
- 既存データソースの削除
- 既存データソースの可用性の確認

この機能は、

`com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterDatasourceManagementService`

サービスを実装することで拡張できます。

メタデータの収集

メタデータ収集機能は、メタデータプロバイダーシステムのデータソースと基礎となるリソースに対する次の操作をカバーします。

- 新しいデータソースを作成し、そのすべてのメタデータを取得
- リソース/データソースのメタデータまたはリソース/データソースのリストを取得
- ルートリソース/データソースの子リソースのリストを取得

この機能は、

`com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterHarvestingService` サービス

を実装することで拡張できます。

データプロファイリング

データプロファイリング情報をメタデータ交換オブジェクトに追加できます。

この機能は、

`com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterHarvestingService` サービス

のメソッド `addProfilingToAsset` を実装することで拡張できます。

データサンプリング

特定のデータソース/リソースのソースシステムからサンプルデータを要求できます。

この機能は、 `com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterSamplingService`

サービスを実装することで拡張できます。

メタデータのプロビジョニング

メタデータプロビジョニング機能は、メタデータプロバイダーシステム上の `パブリック` スペースに既存のデータソース/リソースを提供することを目的とした操作をカバーします。

この機能を使用すると、複数の既存のデータソース/リソースに基づいて新しいデータソース/リソースを作成することもできます。

以下の操作がサポートされています。

- メタデータのプロビジョニング (新しいパブリックソースを作成し、そこにリソースを公開)
- すでにプロビジョニングされたメタデータリソースの変更
- すでにプロビジョニングされたメタデータリソースの削除
- リソースがすでにプロビジョニングされているかどうかの確認
- 指定されたリソースがプロビジョニングされたリソースのソースとしてサポートされているかどうかの確認

この機能は、

`com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterProvisioningService` サービスを実装することで拡張できます。

リネージ抽出

ソースシステムがサポートしている場合は、リネージ情報をソースシステムから抽出できます。この情報は、メタデータ交換オブジェクトに追加し、TIBCO® メタデータエージェントを通じて `{{ site.PRODUCT\_NAME\_R }}` にプッシュして、テクニカルフローとして保存することもできます。

この機能は、`com.tibco.ebx.ca.tabula.agent.adapter.service.AdapterLineageService` サービスを実装することで拡張できます。

例外管理

特定の例外管理フレームワークは、TIBCO® メタデータエージェントフレームワークの一部として提供されます。

このフレームワークには次のエレメントが含まれます。

- `com.tibco.ebx.ca.tabula.agent.common.adapter.exception.MetadataAdapterException` : コードから TIBCO® メタデータエージェント および `{{ site.PRODUCT\_NAME\_R }}` に特定の例外をスローして処理するために実装する必要がある国際化例外クラス。
- `com.tibco.ebx.ca.tabula.agent.common.exception.MetadataExceptionReason` : カスタム例外理由を定義するために列挙クラスによって実装されるインターフェイス。

特定の例外を実装するには、適切なコンストラクターを使用して汎用例外クラスを拡張し、`getErrorBundle()` メソッドをオーバーライドしてエラーメッセージファイルの名前を返す必要があります。このファイルは、次の命名規則に従って、`src/main/resources` フォルダー内で使用可能にする必要があります。

```
<error bundle name>_<language 2 letters code>.properties
```

言語コードはオプションです。指定されていない場合は、バンドルは英語に適用されるものと見なされます。

```

▼ 📁 src/main/resources
  📄 adapterAgentErrorMessages_en.properties
  📄 adapterAgentErrorMessages.properties
  
```

例外の特定のエラー理由を追加するには、プロジェクト内に `MetadataExceptionReason` インターフェイスを実装する `enum` を作成し、そこにエラーメッセージキーを定義する必要があります。

```

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;
    }
}
  
```

```

    }

}

```

特定の例外を定義したくない場合は、クラス

'com.tibco.ebx.ca.tabula.agent.common.adapter.exception.MetadataAdapterTechnicalException' で表される TIBCO® メタデータエージェントフレームワークによって提供される一般的な技術例外を使用できます。

この例外は、文字列メッセージまたは `Throwable` オブジェクトを使用してインスタンス化できます。

ロギング

現時点では、特定のログ記録フレームワークは提供されていません。

TIBCO メタデータエージェントは、slf4j ロギングフレームワークを活用して、適切なログファイルに機能メッセージを書き込みます。

一貫性を保つために、アダプタには同じフレームワークを使用し、メッセージの書き込みに

`System.out.print/println` を使用しないことをお勧めします。

TIBCO® メタデータエージェントの管理

- [EBX® メタデータ管理アプリケーションサブスクリプション](#)
 - 以前に削除されたエージェントの新しい接続
 - 公開鍵の更新
- [エージェントの移行とバックアップ](#)

EBX®メタデータ管理アプリケーションサブスクリプション

以前に削除されたエージェントの新しい接続

最初の起動時に、エージェントは対応する EBX® システムにサブスクライブします。このやり取りの結果、GUID がエージェントストレージに保存され、エージェントと EBX® 間のすべてのやり取りに使用されます。EBX® **管理** パースペクティブで管理対象エージェントを削除すると、エージェント側に保存されている GUID は無効になります。削除されたエージェントを再作成するには、エージェントの H2 データベース (`configDB.mv.db`) を削除し、エージェントを再起動する必要があります。エージェントは自動的に EBX® メタデータ管理アプリケーションにサブスクライブし、新しい GUID を取得します。

公開鍵の更新

公開鍵 (JWT) は、エージェントと EBX® メタデータ管理アプリケーション間のやり取りにおける追加のセキュリティのために使用される一意のトークンです。

有効期間は 60 日間で、その後は自動的に期限切れになります。期限切れの JWT を使用した接続は EBX® メタデータ管理アプリケーションでは受け入れられず、エージェントに公開鍵が無効になったことを通知するために、「ID xxx のエージェントの JWT は期限が切れているか、無効です。エージェントのプロパティで JWT を更新してください。」という、例外が生成されます。

このメッセージが表示された場合、EBX® 管理者から新しい JWT を取得し、それをエージェントのプロパティファイルの公開鍵プロパティにコピーして貼り付け、エージェントを再起動する必要があります。

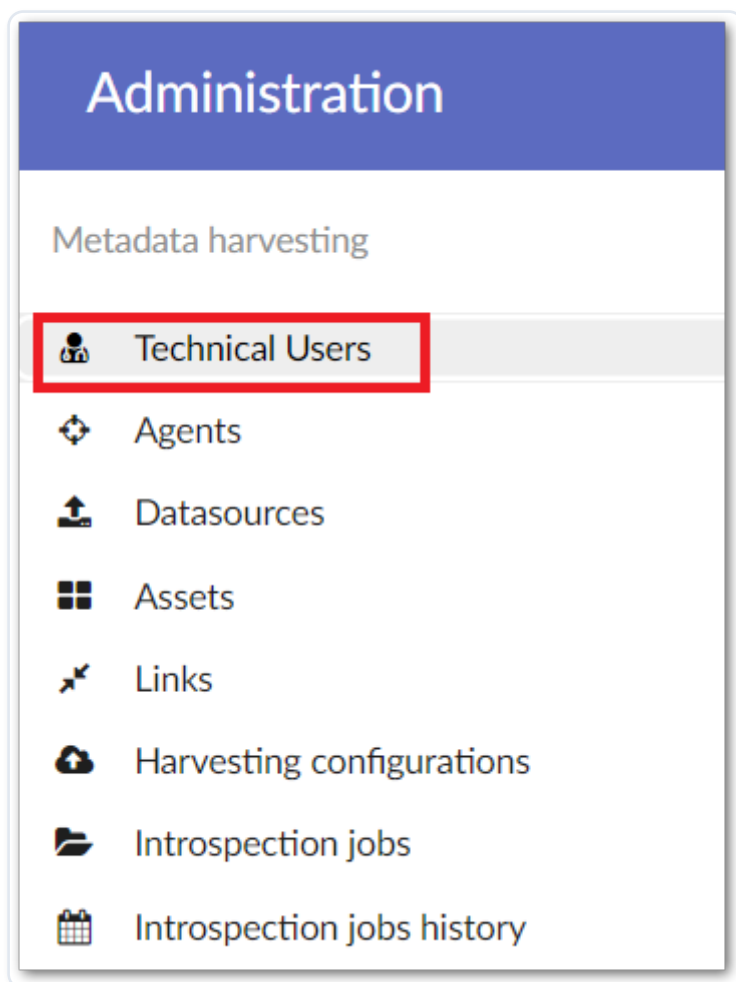
エージェントの移行とバックアップ

既存のエージェントを移行し、EBX® メタデータ管理アプリケーションへの接続を維持する必要がある場合は、以下のプロセスに従ってください。

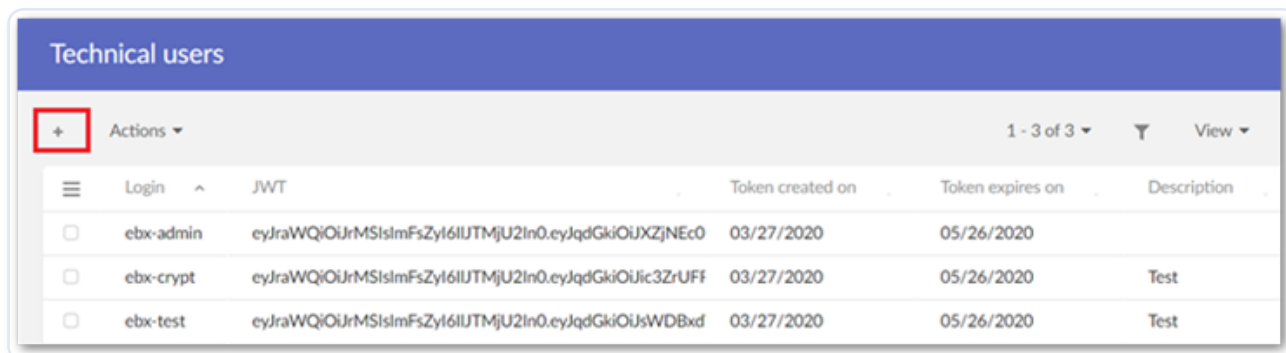
1. エージェントを停止します。
2. `configDB.mv.db` および `quartzDB.mv.db` ファイルをコピーして、エージェントのデータベースと Quartz Scheduler データベースをバックアップします。
3. `application.properties` ファイルをバックアップします。
4. 新しいインスタンスにエージェントをインストールします。
5. データベースのバックアップを `[[インストールフォルダー]]/storage` にコピーします。
6. プロパティファイルを `[[インストールフォルダー]]/conf` にコピーします。
7. `ma.http.ws-url` と `ma.http.port` を更新します。
8. 必要に応じて他のプロパティを更新します。
9. エージェントを再起動します。EBX® の登録情報が自動的に更新されます。

テクニカルユーザー

テクニカルユーザーは、UI アクセス権限を持たず、メタデータエージェントまたは別のテクニカルシステムのみを使用する特定のタイプのユーザーです。**メタデータ管理**パースペクティブを選択し、**テクニカルユーザー**セクションに移動すると、テクニカルユーザーのリストにアクセスできます。



メタデータ管理パースペクティブのテクニカルユーザーで、既存のユーザーを選択して、エージェントと EBX® メタデータ管理アプリケーション間の接続を行う権限があることを指定できます。



システムは、このユーザーに対して **JSON Web Token** を自動的に生成します。このテクニカルユーザーがメタデータエージェントに関連付けられる場合は、**JWT** をメタデータエージェントのプロパティファイルにコピーして貼り付ける必要があります。そうしないと、EBX® メタデータ管理アプリケーションとメタデータエージェント間の通信が許可されません。

JWT は 60 日間有効であり、期限が切れると EBX® メタデータ管理アプリケーションによって自動的に再生成されることに注意してください。

有効期限が切れると、エージェント操作 (収集、TDV からのデータソースの作成など) で有効期限切れメッセージが表示され、エージェントは EBX® に新しい JWT キーの生成を要求します。

この有効期限のメッセージが表示された場合は、次の操作を実行する必要があります。

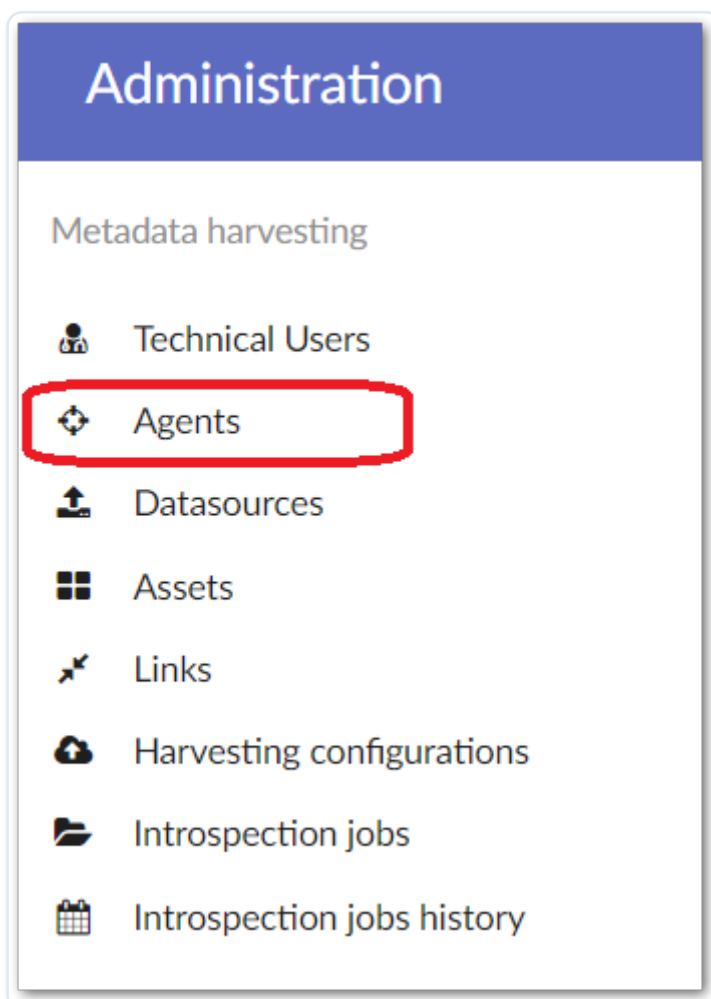
- エージェントを停止します。
- EBX® から新しい JWT キーを取得します。
- このキーをコピーし、エージェントの `application.properties` に貼り付けます。
- エージェントを再起動します。

TIBCO® メタデータエージェント

- 概要
- メタデータエージェントの登録
- メタデータエージェントのお気に入り設定
- メタデータエージェントの無効化
- メタデータエージェントの削除

概要

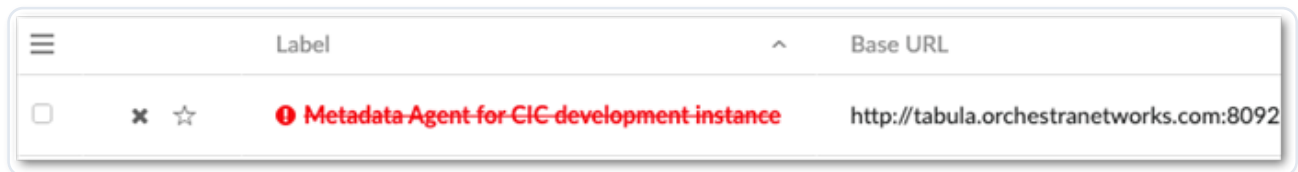
パースペクティブで を選択し、登録済みのメタデータエージェントのリストにアクセスします。



デフォルトでは、アクティブなメタデータエージェントのみが表示されます。非アクティブなメタデータエージェントを表示するには、**メニューに移動し、** に切り替えます。

このテーブル内のレコードは、メタデータエージェントが起動し、メタデータに接続してサブスクライブするか、サブスクリプションが最新であることを確認すると、自動的に作成および更新されます。

エージェントが対応できない場合は、赤で表示されます。マウスオーバーすると、対応できない理由が表示されます。



メタデータエージェントの登録

メタデータエージェントが EBX® メタデータ管理アプリケーションにサブスクライブすると、テーブルに次のエレメントが作成されます。

モジュール名	説明
GUID	サブスクリプション中にエージェントに対して EBX® メタデータ管理アプリケーションによって生成された GUID。
ラベル	ユーザーとのやり取りでメタデータエージェントを表示するために使用されるラベル。
ベース URL	エージェントのベース URL。
エージェント API バージョン	このエージェントの REST API バージョン。利用可能な操作のリストを定義します。
タイムゾーン	スケジュールされたジョブの日付を変換するために使用されるエージェントのタイムゾーン。
JWT	メタデータエージェントが EBX® メタデータ管理アプリケーションに接続するテクニカルユーザーに対応する JSON Web トークン。詳細については、 テクニカルユーザー を参照してください。

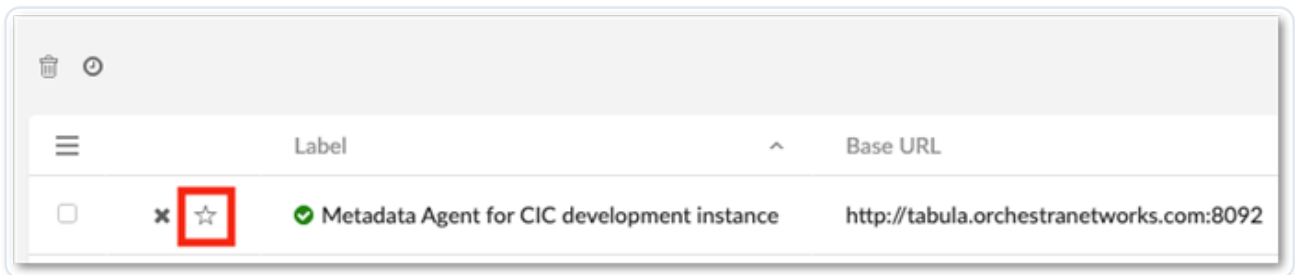
モジュール名	説明
テクニカルユーザー名	テクニカルユーザーのログイン。詳細については、 テクニカルユーザー を参照してください。
ランディングデータスペース	メタデータエージェントが TDV から取得した生のメタデータを保存するために使用するデータスペースの識別子。
宛先データスペース	メタデータが統合されるターゲットデータスペースの識別子。
TDV JDBC URL	TDV システムデータソースの JDBC URL。
TDV ホスト	TDV インスタンスが配置されているホスト名。
TDV ポート	TDV インスタンスが JDBC 接続をリッスンしているポート番号。
TDV のバージョン	TDV のバージョン。
公開ビューフォルダー	TDV 公開ビューフォルダーへのパス。 TIBCO® メタデータエージェントの管理 を参照してください。
操作	メタデータエージェントで利用できる API 操作のリスト。

これらのエレメントは、メタデータエージェントインスタンスが再起動されるたびに検証され、必要に応じて更新されます。

サブスクリプション時に、TDV システムデータソースに対応するインスタンスが EBX® メタデータ管理アプリケーションに作成され、TDV で利用可能な公開ビューの収集が可能になります。

メタデータエージェントのお気に入り設定

複数のメタデータエージェントを同じ EBX® メタデータ管理アプリケーションインスタンスにサブスクライブできます。まだ別のエージェントにリンクされていないインスタンスに対して優先的に使用するメタデータエージェントを定義するには、メタデータエージェントの近くにあるアイコンを選択します。



お気に入りのメタデータエージェントを変更するには、別のメタデータエージェントのアイコンを選択します。お気に入りのエージェントは1つだけ設定できます。

メタデータエージェントの無効化

メタデータエージェントと関連するすべてのオブジェクト (管理対象データソース、資産、リンク、イントロスペクションジョブ) を無効にするには、メタデータエージェントラベルの近くにあるアイコンを選択します。

このメタデータエージェントは登録済みエージェントのリストに表示されなくなり、このエージェントを使用して操作を実行することもできなくなります。無効なエージェントを表示するには、メニューに移動し、に切り替えます。

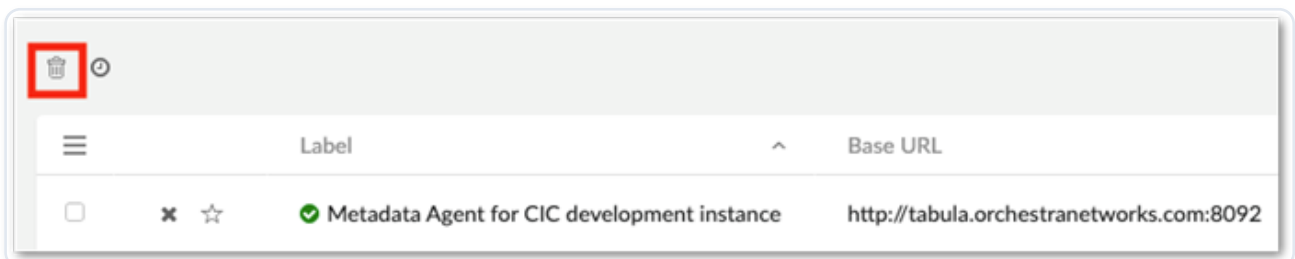
☰	Deactivated?	Label	⤴	Base URL	Agent API Version
☐ ✓	Yes	Metadata Agent for CIC development instance		http://tabula.orchestranetworks.com:8092	3.0.0

このビューから、メタデータエージェントの近くにある ボタンをクリックすると、メタデータエージェントを再度有効化できます。ただし、関連する管理対象オブジェクトはエージェントによって再度有効化されません。

メタデータエージェントの削除

EBX® メタデータ管理アプリケーションからメタデータエージェントを完全に削除するには、このエージェントを選択し、テーブルの上にあるアイコンを選択します。

注意：この操作は元に戻せません。 |



この操作により、システムから関連オブジェクトも削除されます。

- 管理対象のデータソース
- 管理対象の資産

- 管理対象のリンク
- イントロスペクションジョブと対応するジョブ実行履歴

メタデータエージェントによって管理される項目

- 概要
- 管理対象データソース
 - 削除されたデータソースのクリーンアップ
- 管理対象資産
 - 削除された資産のクリーンアップ
- 管理対象リンク
 - 削除されたリンクのクリーンアップ
- 管理対象アイテムとメタデータオブジェクトの関係

概要

メタデータエージェントを使用すると、次の 3 つの管理対象オブジェクトカテゴリのデータソースから生のメタデータを収集できます。

1. 管理対象データソース - プライベート部分で定義された TDV データソース、または TDV 公開ビューを含むフォルダーに対応します。
2. 管理対象資産 - メタデータを取得できるデータソースの子リソース。
3. 管理対象リンク - リレーショナルデータベースの外部キーなど、リソース間の関係。

管理パースペクティブで、メタデータエージェントによって管理されるオブジェクトのリストにアクセスできます。

管理対象のデータソース

管理パースペクティブでデータソースを選択し、サブスクライブされたメタデータエージェントによって管理されるデータソースのリストにアクセスします。

Administration

Metadata harvesting



Technical Users



Agents



Datasources



Assets



Links



Harvesting configurations



Intropection jobs



Intropection jobs history

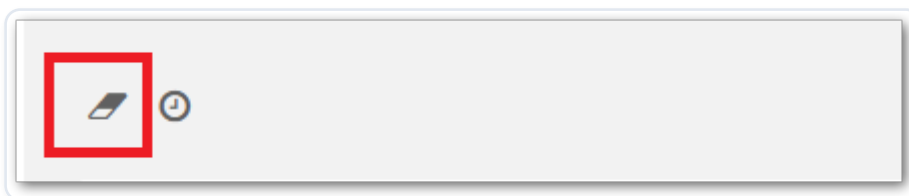
管理対象データソースごとに次の情報が利用できます。

フィールド	説明
エージェント識別子	このデータソースに接続されているメタデータエージェントの識別子。
TDV のデータソース識別子	TDV インスタンス内のこのデータソースのリソースパス。インスタンス。
EBX® メタデータ管理アプリケーションの識別子	管理対象データソースが EBX® メタデータ管理アプリケーションのインスタンスにすでに接続されている場合は、対応する識別子がこのフィールドに表示されます。
ステータス	データソースのステータス

フィールド	説明
	• NEW - EBX® メタデータ管理アプリケーションにまだ転送されていない新しいリソースを表します。 • STORED - EBX® メタデータ管理アプリケーションにインスタンスとしてすでに保存されているデータソースを表します。 • DELETED - TDV インスタンスで削除されたデータソースを表します。
作成日	このレコードの作成日。
最終更新日	このレコードの最終更新日。
無効	対応するエージェントが無効になっている場合、データソースは無効になります。
データソースタイプ	TDV の技術データソースタイプ。サポートされているデータソースタイプの完全なリストについては、 TIBCO® メタデータエージェントの管理 を参照してください。

削除されたデータソースのクリーンアップ

システムから DELETED ステータスのすべてのデータソースを削除するには、テーブルの上にある **削除されたアイテムのクリーンアップアイコン**を選択します。これにより、これらのデータソースがこのテーブルから完全に削除されます。関連するリンクと資産も削除されます。



管理対象の資産

管理 パースペクティブで**資産**を選択し、サブスクライブされたメタデータエージェントによって管理される資産のリストにアクセスします。

Administration

Metadata harvesting

-  Technical Users
-  Agents
-  Datasources
-  Assets
-  Links
-  Harvesting configurations
-  Introspection jobs
-  Introspection jobs history

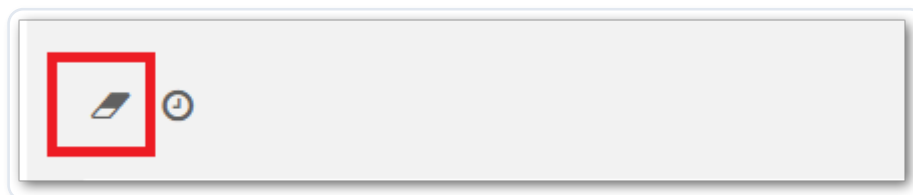
管理対象データソースごとに次の情報が利用できます。

フィールド	説明
データソース	資産が属するデータソース。
エージェント識別子	この資産に接続されているメタデータエージェントの識別子。
TDV のリソース ID	TDV インスタンス内のこの資産のリソースパス。
EBX® メタデータ管理アプリケーションの資産 ID	管理対象資産が EBX® メタデータ管理アプリケーション内の資産にすでに接続されている場合は、対応する識別子がこのフィールドに表示されます。
ステータス	資産ステータス

フィールド	説明
	• NEW - EBX® メタデータ管理アプリケーションにまだ転送されていない新しいリソースを表します。 • STORED - EBX® メタデータ管理アプリケーションに資産としてすでに保存されているリソースを表します。 • DELETED - TDV インスタンスで削除されたリソースを表します。
作成日	このレコードの作成日。
最終更新日	このレコードの最終更新日。
無効	対応するエージェントが無効になっている場合、資産は無効になります。
TDV 親資産 ID	TDV 内の親エレメントへの技術的なパス。

削除された資産のクリーンアップ

システムから DELETED ステータスのすべての資産を削除するには、テーブルの上にある **削除されたアイテムのクリーンアップ** アイコンを選択します。これにより、これらの資産がこのテーブルから完全に削除されます。関連リンクも削除されます。



管理対象のリンク

管理パースペクティブでリンクを選択し、サブスクライブされたメタデータエージェントによって管理されるリンクのリストにアクセスします。

Administration

Metadata harvesting

-  Technical Users
-  Agents
-  Datasources
-  Assets
-  **Links**
-  Harvesting configurations
-  Introspection jobs
-  Introspection jobs history

管理対象データソースごとに次の情報が利用できます。

フィールド	説明
エージェント識別子	このリンクに接続されているメタデータエージェントの識別子。
TDV の ID	外部キーの名前
PK 資産	このリンク関係の主キーを表す管理対象資産。
FK 資産	このリンク関係の外部キーを表す管理対象資産。
EBX® メタデータ管理アプリケーションのリンク ID	この管理対象リンクが EBX® メタデータ管理アプリケーションにリンクとしてすでに保存されている場合は、対応する識別子がこのフィールドに表示されます。

フィールド	説明
ステータス	リンクステータス • NEW - EBX® メタデータ管理アプリケーションにまだ転送されていない新しいリソースを表します。 • STORED - EBX® メタデータ管理アプリケーションにリンクとしてすでに保存されているリソースを表します。 • DELETED - TDV インスタンスで削除されたリソースを表します。
無効	対応するエージェントが無効になっている場合、リンクは無効になります。
TDV 親 ID	親リンクオブジェクトの識別子。

削除されたリンクのクリーンアップ

システムから DELETED ステータスのすべてのリンクを削除するには、表の上にある**削除されたアイテムのクリーンアップ**アイコンを選択します。これにより、これらのリンクがこの表から完全に削除されます。



管理対象アイテムとメタデータオブジェクトの関係

保存されている各管理対象アイテムは、メインのメタデータデータモデル内のアイテムと関係があります。

リンクは、メタデータの収集サービスまたはデータソースからのインスタンスの作成サービスの実行によるメタデータの収集中に作成されます。メタデータが保存されると、管理対象アイテムのステータスは STORED になり、対応する EBX® メタデータ管理アプリケーションオブジェクトの識別子がそれにアタッチされます。

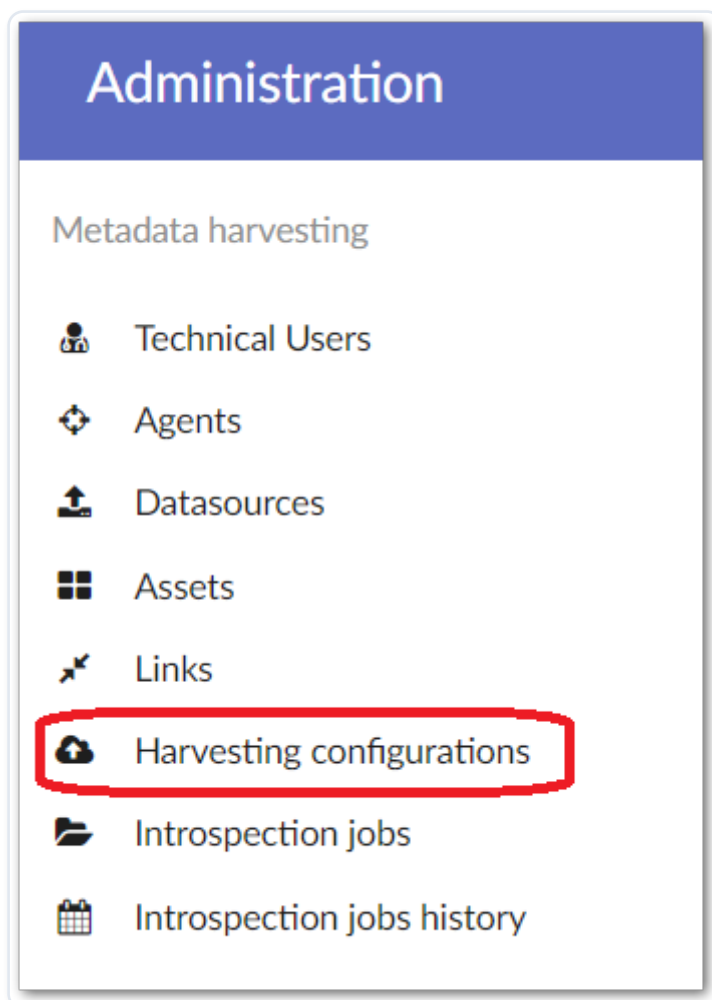
管理対象アイテムにリンクされた EBX® メタデータ管理アプリケーションオブジェクトが削除されると、リンクは切断され、管理対象アイテムのステータスは再び NEW になります。

TDV で物理データソースまたはリソースが削除された場合、EBX® メタデータ管理アプリケーションで対応するオブジェクトに対して実行される次の収集中に、管理対象アイテムは DELETED ステータスを受け取り、リンクされた EBX® メタデータ管理アプリケーションオブジェクトは無効としてマークされます。

リンクされた管理対象アイテムを持つインスタンスと資産の場合、属性名は、メタデータエージェントを介して EBX® メタデータ管理アプリケーションと TDV 間の技術的な交換に使用されるため、読み取り専用になります。

収集の構成

メタデータ管理パースペクティブで **収集の構成** を選択して、収集構成のリストにアクセスします。



収集の構成は、インスタンステーブルと資産テーブルから利用できる **メタデータ収集サービス** を使用して作成されます。

メタデータ管理パースペクティブでは、収集構成の技術的な表現が提供され、テーブルの上にある削除アイコンを使用して 1 つまたは複数の構成を削除できます。

- 収集実行のトラブルシューティング
- イントロスペクションジョブ
 - 失敗したジョブのクリーンアップ
- イントロスペクションジョブの履歴

収集実行のトラブルシューティング

収集の構成テーブルは、収集実行の 1 つをトラブルシューティングする必要がある場合に役立ちます。詳細にアクセスするには、失敗した実行が報告された構成をダブルクリックします。

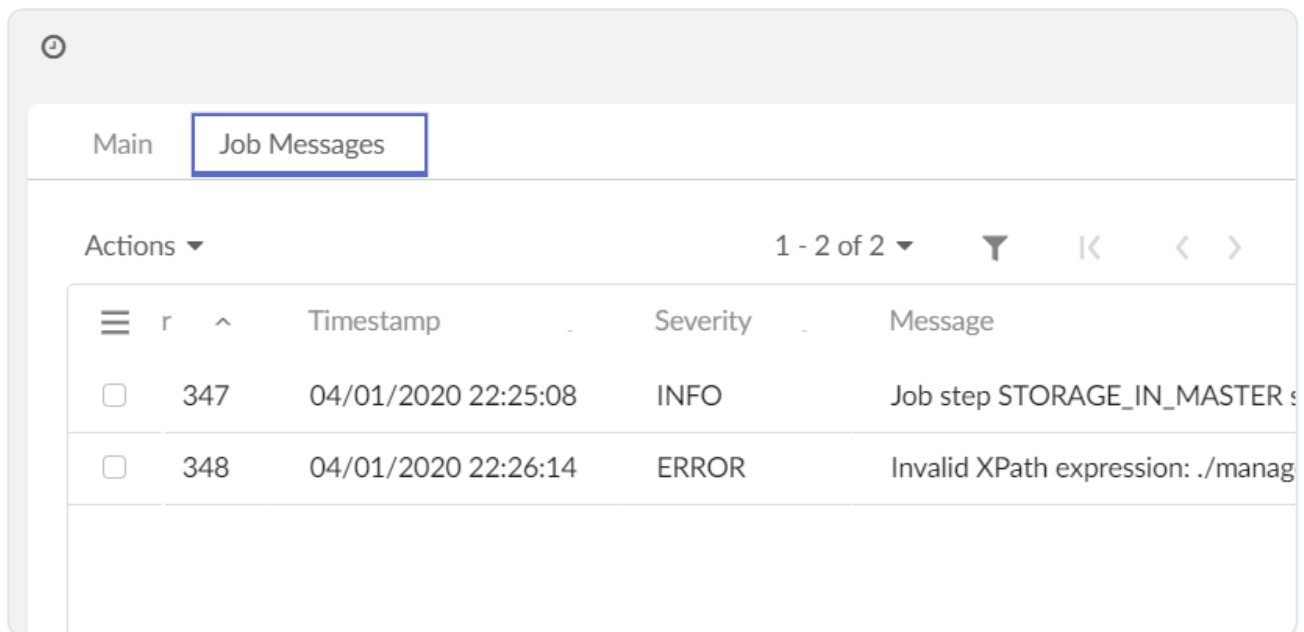
イントロスペクションジョブの近くにあるプレビューアイコンをクリックして、詳細を表示します。

このジョブの実行リストにアクセスするには、**ジョブ実行**タブに切り替えます。

Main Managed Assets Managed Datasources <u>Job Runs</u>				
Actions ▼		1 - 8 of 8 ▼		
≡	ne	End Date	Status	Current Step
☐	00:18	04/01/2020 22:01:24	Failed	STORAGE_IN_MASTER
☐	01:31	04/01/2020 22:02:51	Failed	STORAGE_IN_MASTER
☐	24:53	04/01/2020 22:26:14	Failed	STORAGE_IN_MASTER
☐	19:53	04/02/2020 10:23:15	Accepted	END
☐	24:12	04/02/2020 10:24:27	Accepted	END
☐	49:24	04/02/2020 10:49:36	Accepted	END
☐	50:44	04/02/2020 10:52:41	Accepted	END
☐	53:34	04/02/2020 11:53:51	Accepted	END
Close				

失敗ステータスの最新のジョブ実行をダブルクリックして、その詳細にアクセスします。

ジョブ実行中に発生した重要なメッセージのログにアクセスするには、**ジョブメッセージ**タブに切り替えます。



Main		Job Messages		
Actions ▼		1 - 2 of 2 ▼		
		Timestamp	Severity	Message
☐	347	04/01/2020 22:25:08	INFO	Job step STORAGE_IN_MASTER s
☐	348	04/01/2020 22:26:14	ERROR	Invalid XPath expression: ./manag

重大度が ERROR のメッセージは、実行失敗の根本原因 (エージェントに接続できない、オブジェクトを保存できない、サーバー障害のため実行を完了できないなど) を見つけるのに役立ちます。

イントロスペクションジョブ

管理パースペクティブでイントロスペクションジョブを選択して、収集ジョブのリストにアクセスします。

Administration

Metadata harvesting

-  Technical Users
-  Agents
-  Datasources
-  Assets
-  Links
-  Harvesting configurations
-  Intropection jobs
-  Intropection jobs history

各ジョブの状態を追跡し、必要に応じて実行のトラブルシューティングを行うために役立つ次の情報が各ジョブについて利用できます。

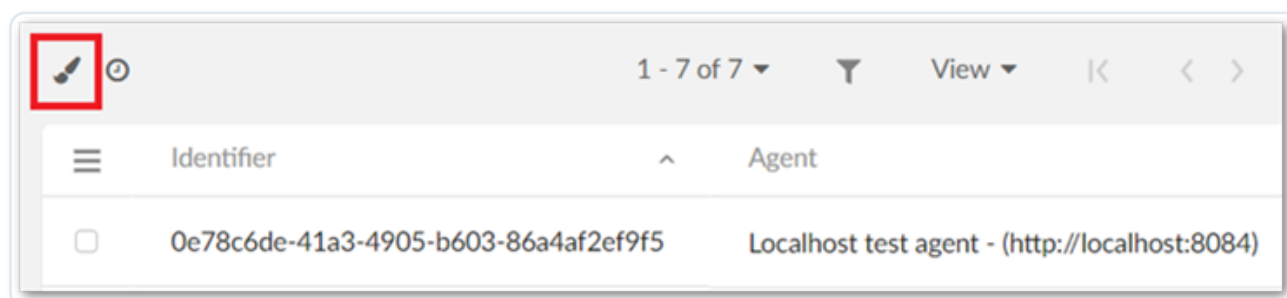
フィールド	説明
識別子	このジョブを識別するために、EBX® メタデータ管理アプリケーションとメタデータエージェント間のやり取りで使用される一意の技術識別子。
エージェント	このジョブに関連するメタデータエージェント。
開始日時	最初のジョブ実行が開始された日時。
間隔 (分)	スケジュールされたジョブの場合、このジョブの実行間隔 (分単位)。
繰り返し回数	スケジュールされたジョブの実行回数。

フィールド	説明
次の実行	ジョブの次回実行が行われる日時。
ステータス - 作成済み - ジョブは作成されましたが、まだ実行されていません。 - スケジュール済み - ジョブは対応するメタデータエージェントでスケジュールされています。 - 進行中 - 最初の実行が開始されてから最後の実行が終了するまで、ジョブにこのステータスが与えられます。 - 失敗 - 最後の実行は失敗しました。 - 成功 - このジョブのすべての実行がエラーなしで完了しました。	実行ステータス
ワークフローの自動マージ	このジョブによって収集されたメタデータがマスターデータに自動的にマージされるか、またはユーザーの確認が必要かどうかを示します。
サンプリングの有効化	true の場合、このジョブの実行中に、適格だがまだサンプルがアタッチされていないすべての資産に対してサンプリングが開始されます。
サンプリングの強制	true の場合、このジョブの実行中に、対象となるすべての資産に対してサンプリングが開始され、既存のサンプルが上書きされます。
自動分類の提案	true の場合、このジョブの実行中に、まだ分類されていないすべての資産に対して分類が開始されます。

フィールド	説明
分類提案の強制更新	true の場合、このジョブの実行中にすべての資産に対して分類が開始されます。
資産の自動分類	true の場合、分類の提案が資産に自動的に適用されます。
無効	対応するエージェントが無効になっている場合、ジョブは無効であると見なされます。
収集構成 ID	履歴のための収集構成 ID。
管理資産	このジョブが収集するルート資産のリスト。ジョブ内のルート資産ごとに、収集範囲から除外する子資産のリストが存在する場合があります。
管理データソース	このジョブが収集するデータソースのリスト。それぞれについて、収集範囲から除外する子資産のリストが存在する場合があります。
ジョブ実行	現在のジョブの実行のリスト。

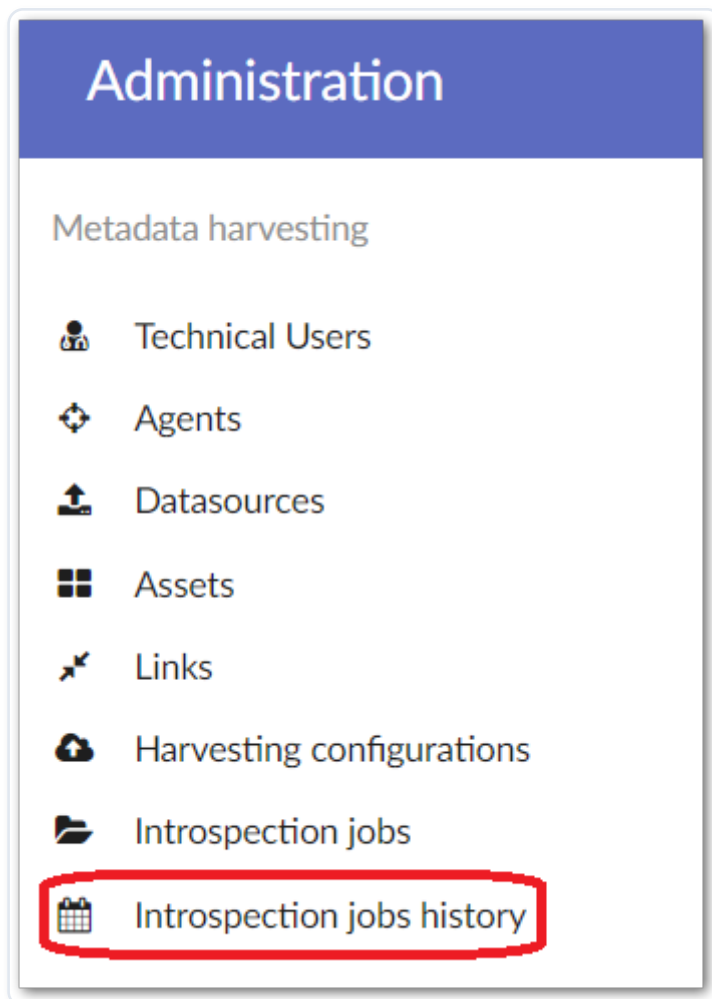
失敗したジョブのクリーンアップ

失敗ステータスのイントロスペクションジョブを削除するには、テーブルの上にある**失敗したジョブのクリーンアップ**アイコンを選択します。これにより、すべてのジョブと、関連するジョブメッセージを含む対応するジョブ実行が削除されます。



イントロスペクションジョブの履歴

管理パースペクティブでイントロスペクションジョブの履歴を選択して、収集ジョブの実行リストにアクセスします。



各ジョブの実行について、各ジョブの状態を追跡し、必要に応じて実行のトラブルシューティングを行うのに役立つ次の情報が利用できます。

フィールド	説明
識別子	実行の自動増分識別子。
イントロスペクションジョブ	イントロスペクションジョブ。
繰り返し番号	手動ジョブ実行 (ユーザーがメタデータの収集サービスから直接起動) の場合、繰り返し番号は常に 0 になります。スケジュールされた実行の場合、繰り返し番号は実行が実行される順序に対応します。

フィールド	説明
開始日時	この実行の開始日時。
終了日	この実行の終了日時。
ステータス • 進行中 • 失敗 - 実行に失敗しました。 • 成功 - スケジュールされた実行が正常に実行され、メタデータがマスターデータに自動的にマージされました。 • 承認待ち - 手動実行が正常に実行され、結果がユーザーによって承認されるのを待っています。 • 承認済み - 手動実行の結果がユーザーによって承認され、マスターデータにマージされました。 • 拒否 - 手動実行の結果がユーザーによって拒否され、削除されました。	実行ステータス - 実行が開始されました。
現在のステップ • STORAGE IN MASTER - 収集されたメタデータをマスターデータの子データスペースに統合します。 • SAMPLING - サンプルを要求しています (サンプリングが有効な場合)。 • CLASSIFICATION - 資産の分類 (有効な場合)。 • PURGE_LANDING - メタデータエージェントのランディングデータスペースのページ。 • END - 実行終了。	現在の実行ステップ

フィールド	説明
EBX® メタデータ管理アプリケーション子データスペース	この実行用に作成された EBX® メタデータ管理アプリケーションの子データスペース。
結果を自動的にマージ	このジョブ実行によって収集されたメタデータがマスターデータに自動的にマージされるか、またはユーザーの確認が必要かどうかを示します。
収集構成 ID	履歴のための収集構成 ID。
収集されたデータソース	この実行によって収集されたデータソースのリスト。
収集された資産	この実行によって収集された資産のリスト。
収集された関係	この実行によって収集されたリンクのリスト。
ジョブメッセージ	このジョブ実行によって記録された重要なメッセージのリスト。通常は、実行手順と発生したエラーが記録されます。

セキュリティガイド

- セキュリティのベストプラクティス

セキュリティのベストプラクティス

ここでは、EBX® セットアップに適切なセキュリティレベルを適用するのに役立つと考えられるベストプラクティスのリストを掲載します。これらのベストプラクティスは、EBX® およびその他の環境、それらの構成、プロトコル、およびポリシーに適用されます。これらのプラクティスは一般的に有益であると見なされていますが、特定のインフラストラクチャーおよびセキュリティポリシーに関連していない場合があります。

- 暗号化アルゴリズム
- HTTPS
- インストール
- Web サーバー
- アプリケーションサーバー
- Java
- データベース
- アーカイブディレクトリ
- ユーザーディレクトリと管理者権限
- 権限

暗号化アルゴリズム

Web サーバーまたはアプリケーションサーバーは、HTTPS パラメーターを設定するときに暗号化アルゴリズムを指定する場合があります。これらのアルゴリズムに関するいくつかの推奨事項は、[HTTPS](#) セクションに記載されています。タイプとして `osd:password` を持つパスワードとフィールドは、SHA1_512 アルゴリズムを使用して、それらの値のハッシュを格納します。これは特に、デフォルトディレクトリのユーザーのパスワードの場合です。

HTTPS

クライアントとの通信には HTTPS を使用することをお勧めします (GUI および REST または SOAP)。すべての HTTP トラフィックは HTTPS にリダイレクトする必要があります。

安全な暗号スイートとプロトコルを可能な限り使用する必要があります。これは、たとえば、Webサーバー、アプリケーションサーバー、および JDBC 接続に適用されます。

TLS v1.2 は、最新の認証済み暗号化 (AEAD と呼ばれる) を提供する唯一のバージョンであるため、メインプロトコルにする必要があります。

いくつかの廃止された暗号化プリミティブは回避する必要があります。

- 匿名の Diffie-Hellman (ADH) スイートは認証を提供しません
- NULL 暗号スイートは暗号化を提供しません
- エクスポート暗号スイートは、接続でネゴシエートされると安全ではありませんが、より強力なスイートを優先するサーバーに対しても使用できます (FREAK 攻撃)
- 耐性の低い暗号 (通常は 40 ビットと 56 ビット) を使用するスイートは、簡単に破られる可能性のある暗号化を使用します
- RC4 は安全ではありません
- 3DES は低速かつ脆弱です

一方、許可された暗号を制限しすぎると、HTTPS 接続をネゴシエートできない可能性があるため、一部のクライアントが接続できなくなる可能性があります。

以下の構成は、EBX® でサポートされているブラウザーと互換性があります。

- 暗号化スイート：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
- バージョン：TLSv1.2

インストール

Web サーバーやアプリケーションサーバーなどの展開済みコンポーネントは、root 以外のユーザーまたは権限のないユーザーを使用して、可能な限り **最小権限の原則**に従ってインストールする必要があります。たとえば、必要なポートとプロトコルのみを開く必要があります。

Web サーバー

インターネット上で Web アプリケーションを公開する必要がある場合は、[DMZ \(非武装地帯\)](#) の Web サーバーでそれらを保護することをお勧めしますが、EBX® とデータベースサーバーは実稼働ゾーンに置くことができます。構成については、次の方法を検討してください。

安全な暗号スイートとプロトコルは、上記の [HTTPS](#) セクションに従って設定する必要があります。

デフォルトの構成を使用しないでください。また、Web サーバーのバージョンとタイプを公開する可能性のあるバナーを削除してください。

たとえば、Apache2 では、バナー (ルートに返されるデフォルトのページ) を削除するには、フォルダー `/var/www/html` を削除するだけです。

また、Apache2 では、Web サーバーを識別するヘッダーを削除するために、ファイルから `ServerTokens` と `ServerSignature` の値を削除します。`security.conf` の値は次のとおりです。

```
# 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
```

Web サーバーを使用して、HTTP セキュリティヘッダーで制限を設定します。オリジンに関連するヘッダーは、EBX® によって返されるすべてのリソースの許可された URL に影響を与えることに注意してください。これには、URL タイプのフィールドのコンテンツが含まれます (例：アバターの画像)。

セキュリティヘッダーのリストと、EBX® 用にそれらを設定する方法は次のとおりです。まず、HTTP セキュリティヘッダーを設定しないように EBX® を構成します。これを行うには、プロパティ `ebx.security.headers.activate` を `false` に設定するか、設定を解除します。

X-XSS-Protection

`x-xss-protection` ヘッダーは、最新の Web ブラウザーに組み込まれているクロスサイトスクリプティング (XSS) フィルターを有効にするように設計されています。ヘッダーは次のようになります。

```
x-xss-protection: 1; mode=block
```

Nginx での有効化

```
proxy_hide_header x-xss-protection;

add_header x-xss-protection "1; mode=block" always;
```

Apache2 での有効化

```
header always unset x-xss-protection

header always set x-xss-protection "1; mode=block"
```

x-Frame-Options

`x-frame-options` ヘッダーは、iframe がサイトに読み込まれないようにすることで、クリックジャッキング保護を提供します。たとえば、EBX® がフレームを介して統合されている場合、これは構成と互換性がない可能性があることに注意してください。ヘッダーは次のようになります。

```
x-frame-options: SAMEORIGIN
```

Nginx での有効化

```
add_header x-frame-options "SAMEORIGIN" always;
```

Apache2 での有効化

```
header always set x-frame-options "SAMEORIGIN"
```

X-Content-Type-Options

`x-content-type-options` ヘッダーは、Internet Explorer と Google Chrome が宣言されたコンテンツタイプから応答をスニффイングするのを防ぎます。これにより、ドライブバイダウンロードの危険性が軽減され、コンテンツが適切に処理されます。ヘッダーは次のようになります。

```
x-content-type-options: nosniff
```

Nginx での有効化

```
add_header X-Content-Type-Options "nosniff" always;
```

Apache2 での有効化

```
header always set X-Content-Type-Options "nosniff"
```

Strict-Transport-Security

`strict-transport-security` ヘッダーは、HTTPS 経由でのみ Web サーバーにアクセスするように Web ブラウザーを制限するセキュリティ拡張機能です。これにより、攻撃に対して脆弱である可能性のある安全でない HTTP 接続を介して接続を確立できなくなります。ヘッダーは次のようになります。

```
strict-transport-security: max-age=31536000; includeSubDomains
```

Nginx での有効化

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains"
always;
```

Apache2 での有効化

```
header always set Strict-Transport-Security "max-age=31536000;
includeSubDomains"
```

Content-Security-Policy

`content-security-policy` HTTP ヘッダーは、セキュリティの追加レイヤーを提供します。このポリシーは、承認されたコンテンツソースを定義し、ブラウザがそれらをロードできるようにすることで、クロスサイトスクリプティング (XSS) やその他のコードインジェクション攻撃などの攻撃を防ぐのに役立ちます。ヘッダーは次のようになります。必ずドメイン名 (例では `server.company.com`) に適合させてください。

```
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';
```

Nginx での有効化

```
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;
```

Apache2 での有効化

```
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

Referrer-Policy HTTP ヘッダーは、行われたリクエストに含めるリファラー情報を管理します。リファラーポリシーは、ユーザーが別のページにつながるリンクをクリックしたときに送信されるリファラー情報を処理する方法を Web ブラウザーに指示します。次のように指定します。

```
Referrer-Policy: strict-origin
```

Nginx での有効化

```
add_header Referrer-Policy: "strict-origin" always;
```

Apache2 での有効化

```
header always set Referrer-Policy "strict-origin"
```

Permissions-Policy

Permissions-Policy HTTP ヘッダーは、Web 開発者が iframe で使用できる機能と使用できない機能を明示的に宣言するメカニズムを提供します。サイトのコードがアクセスまたは変更できる API を制限する一連の「ポリシー」が定義され、特定の機能に対するブラウザーのデフォルトの動作が上書きされます。これは次のようになります。

```
Permission-Policy: "*=()"
```

Nginx での有効化

```
add_header Permissions-Policy: "*=()" always;
```

Apache2 での有効化

```
Header always set Permissions-Policy "*=()"
```

この値はニーズに応じて変更される可能性があることに注意してください。EBX® インストールに [関連 API](#) を使用するカスタマイズされたフロントエンドコードがある場合は、それを適応させる必要がある場合があります。

アプリケーションサーバー

Web サーバーについても、同じベストプラクティスが適用されます。アプリケーションサーバーで技術情報を公開しないでください。たとえば、Tomcat の場合、`server.xml` の `connector` の属性 `server` に一般的な値 `AppServer` を入力することをお勧めします。

```
<Connector port="8080" enableLookups="false" protocol="HTTP/1.1"
useBodyEncodingForURI="true"

        allowBackslash="true" encodedSolidusHandling="passthrough"
server="AppServer"/>
```

アプリケーションサーバーが HTTPS を介して公開されている場合は、上記のセクション [HTTPS](#) に従って安全な暗号スイートとプロトコルを設定する必要があります。

Web サーバーがある場合は、1024 を超えるポートを使用し、Web サーバーにプロキシを実行させることもお勧めします。

Web サーバーがない場合は、上記のようにアプリケーションサーバーがセキュリティヘッダーを設定する必要があります。

Java

Oracle のセキュリティのベストプラクティスに従うことをお勧めします。最後にサポートされたパッチも、特にセキュリティパッチが含まれている場合は、利用可能になり次第適用する必要があります。アプリケーションサーバーやその他の長時間実行されるバックエンドプロセスなどのサーバーシステムには、サーバー JRE の使用を検討してください。サーバー JRE は、Web ブラウザープラグインが含まれていないことを除いて、通常の JRE と同じです。

EBX® では、カスタムコードを使用して非常に高度なカスタマイズが可能です。統合されたすべての Java モジュールは、EBX® によって信頼できると見なされます。したがって、EBX® に基づくすべての開発をレビューおよび検証する必要があります。例として、開発者は適切なエスケープなしでデータベースからの値から HTML を生成するべきではありません。詳細については、[Cross Site Scripting prevention on the OWASP site](#) を参照してください。適切なエスケープの例を次に示します。ストアの名前は、HTML フォームに表示される前にエンコードされます。Apache CommonsLang に含まれている `StringEscapeUtils` クラスは、文字列のエンコードに使用されます。

```
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>");

        // ...

    }

}

```

データベース

データベースは、保存時および転送中に暗号化する必要があります。暗号化用のプライベートキーがある場合は、データファイルと同じ場所に保存しないでください。JDBC 接続に関しては、SSL/TLS を使用するように JDBC ドライバーを構成することを検討してください。詳細な手順については、データベース管理者に問い合わせてください。ドライバーを含め、常に最後にサポートされたバージョンまたは RDBMS を使用する必要があります。

アーカイブディレクトリ

サーバー上では、[アーカイブディレクトリ](#)は適切に保護され、かつ/または暗号化されている必要があります。実際、EBX® インスタンスからエクスポートされたアーカイブはすべてそこに作成され、これらのアーカイブは暗号化もパスワードによる保護もされていません。その結果、これらのファイルにアクセスできるユーザーは、EBX® で定義された権限に関係なく、コンテンツを見ることができるようになります。

ユーザーディレクトリと管理者権限

実稼働プラットフォームとテストプラットフォームの場合、EBX® を[カスタムディレクトリ](#)と統合して、会社のパスワードポリシーを適用する必要があります。デフォルトのディレクトリは、開発プラットフォームでのみ使用できます。

[Separation of Duties](#) のベストプラクティスによると、管理者はユーザーを管理してアクセスを許可できますが、機能的な権限を持たないようにする必要があります。

権限

EBX® で権限を定義する場合は、特別な注意が必要です。担当者は、[権限](#)の内容、特に[権限に関する重要な考慮事項](#)セクションで提供される情報を把握する必要があります。

開発者ガイド

- はじめに
 - TIBCO EBX® モジュールのパッケージ化
 - Java へのマッピング
 - Java 開発者向けのツール
 - 用語の変更
- データモデル
 - はじめに
 - データ型
 - テーブルと関係
 - 制約
 - トリガーと関数
 - ラベルとメッセージ
 - 追加のプロパティ
 - データサービス
 - ツールバー
 - カスタムフォーム
 - パブリケーション REST API
- ワークフローモデル
- ユーザーインターフェイス
 - インターフェイスのカスタマイズ
 - ユーザーサービス
 - 概要
 - クイックスタート
 - ユーザーサービスの実装
 - ユーザーサービスの宣言

- 開発の推奨事項
- SOAP データサービス
 - はじめに
 - WSDL の生成
 - SOAP 操作
- REST データサービス
 - はじめに
 - ビルトイン RESTful サービス
 - JSON 形式
 - はじめに
 - 拡張
 - コンパクト形式
 - 共通
 - その他
- EBX® の SQL
 - はじめに
 - 比較演算子
 - 算術演算子と関数
 - 論理演算子
 - 文字列の演算子と関数
 - 日付と時刻の関数
 - EBX® SQL 関数
- REST Toolkit
- レコードの権限
- EBX® スクリプト
 - リファレンス
 - 使用方法

- 関数フィールド
- スクリプトタスク
- テーブルトリガー
- データセットトリガー
- スクリプト化された REST サービス
- REST サービス
- ロールマッパー
- 制約検証
- API
 - ユニットの概要
 - デフォルトのユニット
 - ユニット core.complex
 - ユニットコアデータ
 - ユニット core.date
 - ユニット core.datetime
 - ユニット core.dictionary
 - ユニット core.http
 - ユニット core.json
 - ユニット core.list
 - ユニット core.locale
 - ユニット core.log
 - ユニット core.math
 - ユニット core.message
 - ユニット core.profile
 - ユニット core.resource
 - ユニット core.set
 - ユニット core.string
 - ユニット core.time
 - ユニット core.uri

- ユニット core.validation
- ユニット core.rest
- EBX® Script IDE
 - EBX® Script IDE の概要
 - 基本操作ガイド
 - チュートリアル
 - 関数の作成
 - トリガーの作成
 - ワークフロースクリプトの作成
 - スクリプト化された REST サービスの作成チュートリアル
 - ユーザーインターフェイスリファレンス
- Java APIポップアップで Java API を開く

はじめに

- TIBCO EBX® モジュールのパッケージ化
- Java へのマッピング
- Java 開発者向けのツール
- 用語の変更

TIBCO EBX® モジュールのパッケージ化

EBX® モジュールは、XML スキーマドキュメント、Java クラス、静的リソースなどのさまざまなリソースをパッケージ化した標準の Jakarta EE Web アプリケーションです。

EBX® モジュールは Web アプリケーションであるため、クラス読み込みの分離、WAR または EAR パッケージ化、Web リソースの公開などの機能のメリットを享受できます。

- モジュール構造
- モジュール宣言
- モジュール登録
 - 展開記述子の例
 - 登録例

- パッケージ化されたリソース
 - ディレクトリ構造
 - 例

モジュール構造

EBX® モジュールには次のファイルが含まれています。

<code>/WEB-INF/ebx/module.xml</code>	この必須ドキュメントは、モジュールの主なプロパティとサービスを定義します。 モジュール宣言 を参照してください。
<code>/WEB-INF/web.xml</code>	これは標準の Jakarta EE 展開記述子です。アプリケーションサーバーの起動時に EBX® モジュールの登録を実行できます。 モジュール登録 を参照してください。
<code>/META-INF/MANIFEST.MF</code>	オプション。存在する場合、EBX® は「実装タイトル」と「実装バージョン」の値を <code>[管理] > [技術構成] > [モジュールとデータモデル]</code> に報告します。
<code>/www/</code>	このオプションのディレクトリには、パブリック URL 経由でアクセスできるすべてのパッケージ化されたリソースが含まれます。 パッケージ化されたリソース を参照してください。

モジュール宣言

モジュールは、ドキュメント `/WEB-INF/ebx/module.xml` を使用して宣言されます。以下はその例です。

```
<?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>

```

各プロパティに関するドキュメントについては、[関連スキーマ](#)を参照してください。主なプロパティは次のとおりです。

エレメント	説明	必須
<code>name</code>	サーバーインスタンス内のモジュールの一意的識別子を定義します。モジュール名は通常、Web アプリケーションの名前 (ディレクトリの名前) に対応します。	はい
<code>publicPath</code>	パブリック URL で Web アプリケーションを識別するモジュール名以外のパスを定義します。このパスは、絶対 URL を計算するときにモジュールの外部リソースの URL に追加されます。このフィールドが定義されていない場合、パブリックパスは上記で定義されたモジュールの <code>名前</code> になります。	いいえ
<code>services</code>	レガシー API を使用してユーザーサービスを宣言します。レガシーユーザーサービスの 宣言と構成 を参照してください。バージョン 5.8.0 からは、 新しいユーザーサービス を使用することを強くお勧めします。	いいえ

エレメント	説明	必須
beans	再利用可能な Java Bean コンポーネントを宣言します。 ワークフローパッケージを参照してください。	いいえ
ajaxComponents	Ajax コンポーネントを宣言します。Java API のモジュールでの Ajax コンポーネントの宣言を参照してください。	いいえ

モジュール登録

EBX® によって識別されるためには、アプリケーションサーバーの起動時にモジュールを登録する必要があります。Web アプリケーションの場合、すべての EBX® モジュールは次の条件を満たす必要があります。

- クラス `ModuleRegistrationListener` を拡張するアノテーション `@WebListener` を持つ Java クラスが含まれます。

注意:

`@WebListener` アノテーションを使用する場合は、Web アプリケーションのサーブレット 3.0 アノテーションスキャンをアクティブにするようにアプリケーションサーバーが構成されていることを確認してください。詳細については、`JSR 315 : Java™ Servlet 3.0 仕様` を参照してください。

または

- クラス `ModuleRegistrationServlet` を拡張するサーブレットを含みます。
- 展開記述子 `/WEB-INF/web.xml` でこのサーブレットの標準宣言を行います。
- 展開記述子に次の標準エレメントを追加して、このサーブレットがサーバーの起動時に登録されるようにします - `<load-on-startup>1</load-on-startup>`

追加の推奨事項と情報

- `ModuleRegistrationServlet` のメソッド `handleRepositoryStartup` を使用すると、モジュールに関連付けられたロガーを設定したり、一般的な JavaScript や CSS リソースなどの追加の動作を定義したりできます。

- `ModuleRegistrationServlet` を拡張する特定のクラスは、Web アプリケーション内に配置する必要があります (`/WEB-INF/classes` または `/WEB-INF/lib` の下。このクラスは、モジュールに関連付けられたデータモデルで使用する Java クラスをロードするために、アプリケーションのクラスローダーへのフックとして内部的に使用されるため)。
- アプリケーションサーバーの起動プロセスは非同期で、Web アプリケーション / EBX® モジュールは動的に検出されます。EBX® リポジトリの初期化はこのプロセスに依存しており、使用されているすべてのモジュールの登録を最大無制限の時間待機します。その結果、使用されているモジュールが何らかの理由で展開されていない場合は、EBX® メイン構成ファイルで宣言する必要があります。詳細については、プロパティ [モジュールの展開解除宣言](#) を参照してください。
- すべてのモジュールの登録と登録解除は、`log.kernel` カテゴリに記録されます。
- モジュールのロード中に例外が発生した場合、その原因はアプリケーションサーバーログに書き込まれます。
- サーブレットがサービスを停止すると、モジュールは登録解除され、データモデルと関連するデータセットは使用できなくなります。ホットデプロイメント/アンデプロイメントは [サポートされません](#)。

展開記述子の例

以下は、Jakarta EE 展開記述子 (`/WEB-INF/web.xml`) の例です。

```
<?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>

```

登録例

以下は `ModuleRegistrationServlet` の実装例です。

```

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_STYLESHEE
T_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
```

```
    ...
```

```
}
```

}

パッケージ化されたリソース

パッケージ化されたリソースは、クライアントブラウザから直接アクセスでき、

`osd:resource` フィールドまたは Java API 経由で管理および指定できるファイルとドキュメントです。さまざまなタイプがあり、ローカライズすることもできます。

以下も参照してください。:

- `ResourceType`
- **タイプ** `osd:resource`

ディレクトリ構造

パッケージ化されたリソースは、次のディレクトリ構造の下に配置する必要があります。

1. 最初のレベルでは、ディレクトリ `/www/` がモジュール (Web アプリケーション) のルートに配置されている必要があります。
2. 2 番目のレベルでは、ディレクトリはローカリゼーションを指定する必要があります。次のようになります。
 - `common/` には、ロケールに依存しないため、またはデフォルトのローカリゼーションとして使用されるすべてのリソースが含まれている必要があります (EBX®では、デフォルトのローカリゼーションは `en`、つまり英語です)。
 - `{lang}/` は、その下にあるリソースのローカリゼーションが必要な場合に使用します。`{lang}` は実際のロケールコードに置き換えられます。これは、EBX®でサポートされているロケールに対応している必要があります。詳細については、[EBX® ローカリゼーションの構成](#) を参照してください。
3. 3 番目のレベルでは、ディレクトリはリソースタイプを指定する必要があります。次のようになります。
 - JavaScript リソースの場合は `jscripts/` です。
 - `stylesheets/` はカスケードスタイルシート (CSS) リソース用です。
 - HTML リソースの場合は `html/` です。

- アイコン型のリソースの場合は `icons/` です。
- 画像型のリソースの場合は `images/` です。

例

この例では、画像 `logoWithText.jpg` のみがローカライズされるリソースです。

```
/www
├─ common
│   └─ images
│       └─ myCompanyLogo.jpg
│       └─ logoWithText.jpg
│   └─ jsconfigs
│       └─ myCompanyCommon.js
│   └─ stylesheets
│       └─ myCompanyCommon.css
├─ de
│   └─ images
│       └─ logoWithText.jpg
└─ fr
    └─ images
        └─ logoWithText.jpg
```

Java へのマッピング

- Java からのデータアクセス
 - 読み取りアクセス

- 書き込みアクセス
- 可変オブジェクトの変更
- トランザクションと並行処理
 - 並行処理
 - クエリスナップショット分離
 - Adaptation オブジェクト
- データ型のマッピング
 - 単純データ型
 - 複合データ型
- Java バインディング
 - 利点
 - XML 宣言

Java からのデータアクセス

読み取りアクセス

データは、主に `Adaptation` と `ValueContext` などのさまざまな汎用 Java クラスから読み取ることができます。これらのクラスの getter メソッドは、セクション**データ型のマッピング**で説明されているマッピングルールに従って型指定されたオブジェクトを返します。

書き込みアクセス

データの更新は、適切に管理されたコンテキストで実行する必要があります。

- プロシージャ実行のコンテキストでは、インターフェイス `ValueContextForUpdate` のメソッド `setValue...` を呼び出すか、
- ユーザー入力の検証中に、クラス `ValueContextForInputValidation` のメソッド `setNewValue` を呼び出します

可変オブジェクトの変更

データ型のマッピング セクションで説明されているマッピングによると、アクセスされる Java オブジェクトの一部は可変オブジェクトです。これらは、`リスト`、`日付`、または任意の `JavaBean` のインスタンスです。したがって、これらのオブジェクトは、独自のメソッドによって

ローカルに変更できます。ただし、上記の setter のいずれかが呼び出され、現在のトランザクションが正常にコミットされない限り、このような変更は返されたオブジェクトに対してローカルのままになります。

トランザクションと並行処理

並行処理

データスペースレベル	1 つのデータスペースでは、1 つの読み書き可能な <code>Procedure</code> と、複数の同時実行可能な <code>ReadOnlyProcedure</code> の実行をサポートしています。任意の <code>Procedure</code> 以外の同時アクセスもサポートしています。
リポジトリレベル	リポジトリレベルでは、同時実行性は特定の操作に対してのみ制限されます。例 (網羅的ではないリスト)： - データモデルのパブリケーションでは、多くの操作が除外されています - データスペースのマージでは、マージに関係する 2 つのデータスペースに対する書き込み操作は除外されます

クエリスナップショット分離

次の表は、クエリの分離に関連するプロパティを定義しています。クエリ結果は、`QueryResult` または `RequestResult` のいずれかの Java API によって表されることに注意してください。

<code>Procedure</code> 以外のクエリ	クエリ結果のフェッチ時にデータがフリーズされます。より正確には、クエリ結果は、この結果をフェッチする時点で最後にコミットされたトランザクションの時点でコミットされたデータのみにアクセスします。この結果の内容は、後で変更されることはありません。
-------------------------------	---

	Procedure 外部のクエリは、自己完結型の ReadOnlyProcedure と見なすことができます。
Procedure の基になるデータスペースと同じデータスペース内の Procedure 内のクエリ	クエリ結果は、プロシージャが開始される前に最後にコミットされた状態と、クエリ結果のフェッチ前に Procedure で発生した変更を反映します。 Procedure で何が起こっても、それ以降に、この結果の内容が変更されることはありません。
別のデータスペースの Procedure 内のクエリ	一貫性はリポジトリレベルで保証されるため、クエリ結果は Procedure が開始される前に最後にコミットされた状態を反映します。この結果の内容は、リポジトリ全体で何が起こっても、クエリがフェッチされた後に変更されることはありません。

Adaptation オブジェクト

Java では、永続データセットまたは永続レコードは両方とも **Adaptation** クラスのインスタンスによって表されます。

次の表は、**Adaptation** オブジェクトに関連するプロパティを定義しています。

不変性	Adaptation オブジェクトインスタンスは不変です。 したがって、クライアントコードは、Adaptation オブジェクトを長期間 (特にトランザクションの境界を超えて) 「保持」しないでください。ただし、メソッド Adaptation.getUpToDateInstance を呼び出すことは可能です。
フェッチ	Adaptation が QueryResult からフェッチされる場合、前のセクションで説明したスナップショット分離ルールが適用されます。それ以外の場合、実行中の

`Procedure` から `Adaptation` がフェッチされると、`Procedure` が開始する前に最後にコミットされた状態が反映されます。それ以外の場合、`QueryResult` または実行中の `Procedure` の外部では、`Adaptation` はレコードの状態をフェッチ時に反映します。

以下も参照してください。:

- `AdaptationHome.findAdaptationOrNull`
- `AdaptationTable.lookupAdaptationByPrimaryKey`

データ型のマッピング

このセクションでは、XML スキーマの型定義とエレメント宣言を Java 型にマッピングする方法について説明します。

単純データ型

単純なデータ型の基本ルール

各 XML スキーマの単純型は Java クラスに対応します。マッピングは、[XML スキーマのビルトイン純型](#) の表に記載されています。

以下も参照してください。:

- `SchemaNode.createNewOccurrence`

単純なエレメントの複数のカーディナリティ

属性 `maxOccurs` が 1 より大きい場合、エレメントは集約リストであり、Java の対応するインスタンスは `java.util.List` のインスタンスです。

リストのエレメントは、単純型のマッピングから決定される Java クラスのインスタンスです (前のセクションを参照)。

複合データ型

クラス宣言のない複合型の定義

デフォルトでは (属性 `osd:class` なし)、複合型のターミナルノードは内部クラスを使用してインスタンス化されます。このクラスは、汎用の JavaBean 実装を提供します。ただし、カスタムクライアント Java コードがこれらの値にアクセスする必要がある場合は、カスタム JavaBean を使用してください。これを行うには、次のセクションで説明する `osd:class` 宣言を使用します。

属性 `osd:class` の有無にかかわらず、メソッド `SchemaNode.createNewOccurrence`、`SchemaNode.executeRead`、および `SchemaNode.executeWrite` を呼び出すことにより、マップされた Java オブジェクトを透過的にインスタンス化、読み取り、変更できます。

複合型のカスタム JavaBeans へのマッピング

XML スキーマ複合型をカスタム Java クラスにマップできます。これは、属性 `osd:class` を複合ノード定義に追加することによって行われます。エレメントに `xs:maxOccurs> 1` がない限り、ノードを **ターミナル** ノードと見なすには属性 `osd:access` も指定する必要があります。エレメントが `xs:maxOccurs> 1` の場合、自動的にターミナルと見なされます。

カスタム Java クラスは、JavaBean プロトコルに準拠している必要があります。これは、複合型の各子がクラスの JavaBean プロパティに対応している必要があることを意味します。さらに、各 JavaBean プロパティは読み取り/書き込みプロパティにする必要があります、その実装では、setter メソッドによって設定された値が getter メソッドによってそのまま返されるようにする必要があります。これらのメソッドでは、コンテキスト計算は許可されていません。

例

この例では、Java クラス `com.carRental.Customer` は、メソッド `getFirstName()` および `setFirstName(String)` を定義する必要があります。

JavaBean は、`UIBeanEditor` を使用して、TIBCO EBX® 内にカスタムユーザーインターフェイスを持つことができます。

```
<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>

```

複雑なエレメントの複数のカーディナリティ

属性 `maxOccurs` が 1 より大きい場合、Java の対応するインスタンスは次のとおりです。

- 集約リストの `java.util.List` のインスタンス。リスト内のすべてのエレメントは、[単純型のマッピング](#) によって決定される Java クラスのインスタンスです
- プロパティ `osd:table` が指定されている場合、`AdaptationTable` のインスタンス

Java バインディング

Java バインディングは、データモデルの構造を反映する Java タイプの生成をサポートします。Java コードの生成は、ユーザーインターフェイスで実行できます。[Java バインディングの生成](#) を参照してください。

利点

XML スキーマ構造と Java コードの間のリンクを確保することには、いくつかの利点があります。

- **開発支援**：IDE でサポートされている場合、パラメーターへのアクセスパスを入力するとオートコンプリートされます
- ****アクセスコードの検証**：**パラメーターへのすべてのアクセスは、コードのコンパイル時に検証されます
- ****影響の検証**：**データモデルを変更するたびに、コードのコンパイル状態に影響があります
- **相互参照**：IDE の参照ツールを使用すると、パラメーターが使用されている場所を簡単に確認できます

したがって、Java バインディングを使用することを強くお勧めします。

XML 宣言

データモデルから生成される Java タイプの仕様は、メインスキーマに含まれています。

各バインディングエレメントは、生成ターゲットを定義します。XPath 表記では、`xs:schema/xs:annotation/xs:appinfo/ebxbnd:binding` に配置する必要があります。ここで、プレフィックス `ebxbnd` は名前空間への参照です。URI `urn:ebx-schemas:binding_1.0` で識別されます。世代ターゲットが異なる場合は、いくつかのバインディングエレメントを定義できます。

エレメント `ebxbnd:binding` の属性 `targetDirectory` は、Java タイプの生成に使用されるルートディレクトリを定義します。通常、これはプロジェクトのソースコード `src` を含むディレクトリです。相対パスは、XML スキーマではなく、VM の現在のランタイムディレクトリに基づいて解釈されます。

[XML スキーマのバインディング](#) を参照してください。

XML バインディングの例

```
<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>

    </ebxnd:binding>

</xs:appinfo>

</xs:annotation>

...

</xs:schema>

```

Java 定数は、XML スキーマパスに対して定義できます。これを行うには、ルートノード `/` を含むスキーマノードから 1 つ以上のインターフェイスを生成します。この例では、2 つの Java パス定数インターフェイスを生成します。1 つはノード `/rules` から、もう 1 つはスキーマのノード `/stylesheet` から生成します。インターフェイス名は、属性 `typeName` を持つ要素 `javaPathConstants` によって記述されます。関連するノードは、属性 `root` を持つ要素 `nodes` によって記述されます。

Java 開発者向けのツール

TIBCO EBX® は、Java 開発者に EBX® API の使用と開発環境との統合を容易にするツールを提供します。

- 開発ツールの有効化
- データモデル更新ツール
- Java バインディングの生成
- ノードへのパス
- Web コンポーネントリンクジェネレーター

開発ツールの有効化

開発ツールを有効にするには、EBX® を *開発モード* で実行します。これは、EBX® メイン構成ファイル *EBX® 実行モード* で、プロパティ `backend.mode=development` を使用して指定されます。

データモデル更新ツール

EBX® が提供するデータモデリングツールを使用せずに、データモデルを XML スキーマドキュメントとして直接編集する場合は、アプリケーションサーバーを再起動せずにデータモデルを更新できます。

領域で、>>> (または) を選択します。

注意:

この操作はデータの一貫性に関して重要であるため、データモデルを更新すると、リポジトリに対するグローバル排他ロックが取得されます。つまり、他のほとんどの操作 (データアクセスと更新、検証など) は、データモデルの更新が完了するまで待機することになります。

Java バインディングの生成

Java バインディングによって指定された Java タイプは、ナビゲーションペインで > を選択することで、データセットまたはデータモデルから生成できます。

以下も参照してください。:

- [Java バインディング](#)

ノードへのパス

フィールドは、ノードのドキュメントペインに表示されます。このフィールドはノードへのパスを示し、XPath 式を記述するときに役立ちます。

注意:

このフィールドは管理者が常に利用できます。

Web コンポーネントリンクジェネレーター

サービスは、EBX® Web コンポーネントを呼び出す HTTP リクエストを作成するために設計されたユーザーインターフェイスです。このサービスを起動するには、ナビゲーションペインで > を選択します。

用語の変更

新しい TIBCO EBX® リリースでは、ユーザーに新しい語彙を導入できます。下位互換性を維持するために、これらの用語の変更は通常 API に影響を与えません。したがって、Java クラス名、メソッド名、データサービス操作名などは、引き続き古いバージョンの用語を使用します。この章の目的は、API の古い用語と新しい用語の対応を容易にすることです。

以下も参照してください。:

- [用語集](#)

- [バージョン 5.9 での用語の変更](#)
- [バージョン 5.0 での用語の変更](#)

バージョン 5.9 での用語の変更

新しい用語	バージョン 5.9.0 より前の用語
D3 プライマリノード	D3 マスターノード
D3 レプリカノード	D3 スレーブノード

バージョン 5.0 での用語の変更

次の表は、バージョン 5.0.0 の用語と以前の用語の間のマッピングをまとめたものです。

新しい用語	バージョン 5.0.0 より前の用語
データセット	Adaptation インスタンス
子データセット	子適応インスタンス
データモデル	データモデル

新しい用語	バージョン 5.0.0 より前の用語
データスペース	ブランチ
スナップショット	バージョン
データスペースまたはスナップショット	ホーム
データワークフロー	ワークフローインスタンス
ワークフローモデル	ワークフローの定義
ワークフローパブリケーション	ワークフロー
フィールド	属性
継承されたフィールド	継承された属性
レコード	レコード/オカレンス
検証ルール	制約
単純/詳細制御	単純/詳細制約

データモデル

- はじめに
- データ型
- テーブルと関係
- 制約
- トリガーと関数
- ラベルとメッセージ
- 追加のプロパティ
- データサービス
- ツールバー
- カスタムフォーム
- 公開 REST API

はじめに

データモデルは、TIBCO EBX® リポジトリで管理されるデータの構造定義です。データモデルは、最高レベルのデータ一貫性を保証し、データ管理を容易にする EBX® の機能に貢献します。

具体的には、データモデルは XML スキーマ標準 (W3C 勧告) に準拠したドキュメントです。主な特徴は次のとおりです。

- 整数、ブール値、小数点、日付、時刻など、明確に定義された **単純なデータ型** の豊富なライブラリ。
- 追加の **単純型** と **複合型** を定義する機能。
- **集約リスト** と呼ばれる、項目の単純なリストを定義する機能。
- **検証制約** (ファセット)。例：列挙型、一意性制約、最小/最大境界

EBX® は、XML スキーマの拡張機能を使用して、次のような他の有用な情報も提供します。

- **定義済みタイプ** 例：ロケール、リソース、html など
- **テーブル** と **外部キー制約** の定義。
- EBX® 内のデータを Java Bean にマッピングします。
- **高度な検証制約** (拡張ファセット)、動的列挙など。
- ラベル、説明、エラーメッセージなどの広範な **プレゼンテーション情報**。

注意:

EBX® は、一部の機能がマスターデータ管理に関連しないため、W3C 推奨事項のサブセットをサポートしています。

- データモデルの編集
- リファレンス
- データセットとデータモデルの関係
- XML スキーマの前提条件
- 規則
- 予約名を持つスキーマ

データモデルの編集

データモデルを定義するには、次の 2 つの方法があります。

- データモデルは、XML スキーマエディターまたはデータモデルアシスタントを使用して定義できます。データモデルアシスタントには、EBX® ユーザーインターフェイスに統合されるという利点があり、詳細な基礎となる XML を抽象化します。詳細については、[データモデルの概要](#) を参照してください。

データモデルアシスタントを使用すると、ドキュメント化されていない機能を DMA の外部で使用できます。たとえば、ツールバーとウィジェットがあります。

- 外部の XML スキーマドキュメントエディターを使用します。

参照

XML スキーマの概要については、W3Schools の [XML スキーマチュートリアル](#) を参照してください。

以下も参照してください。:

- [XML スキーマ パート 0: 入門](#)
- [XML スキーマ パート 1: 構造](#)
- [XML スキーマ パート 2: データ型](#)

データセットとデータモデルの関係

各ルートデータセットは、単一のデータモデルに関連付けられています。データスペースの作成時に、データセットのベースとなる関連データモデルが選択されます。

以下も参照してください。:

- [データセットの作成](#)

XML スキーマの前提条件

XML スキーマが EBX® で受け入れられるためには、属性 `osd:access="--"` を含むグローバルエレメント宣言が含まれている必要があります。

```
<?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>
```

規則

規則により、名前空間は常に次のように定義されます。

プレフィックス	名前空間
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

予約名を持つスキーマ

EBX® のいくつかのデータモデルには予約名があります。

次のいずれかの文字列で終わる他のデータモデルへのすべての参照 (インポート、インクルード、または再定義に属性 `schemaLocation` を使用) は予約されています。

- common_1.0.xsd
- org_1.0.xsd
- coreModel_1.0.xsd
- session_1.0.xsd
- userServices_1.0.xsd
- entityMappings_1.0.xsd

これらの XSD ファイルは、パス `/WEB-INF/ebx/schemas` にあるモジュール `ebx-root-1.0` に提供されているスキーマに対応しています。属性 `schemaLocation` は、ファイル名が同一の場合は、この場所にあるファイルまたはそのコピーを参照できます。これは、モジュールが `ebx-root-1.0` に依存しないようにする場合に便利です。

セキュリティ上の理由から、EBX® はこれらのスキーマの変更を防ぐために内部定義を使用します。

データ型

この章では、TIBCO EBX® でサポートされているデータ型について詳しく説明します。

以下も参照してください。:

- テーブルと関係

- XML スキーマのビルトイン単純型
- XML スキーマの名前付き単純型
- XML スキーマ複合型
- EBX® で定義された拡張単純型
- EBX® で定義された複合型
 - ユーザー定義属性とカタログの制限
- 集約リスト
- 外部データモデルの包含

XMLスキーマのビルトイン単純型

次の表に、EBX® でサポートされている XML スキーマで定義されているすべての単純型と、それに対応する Java 型を示します。

XML スキーマタイプ	JAVA クラス	注意
<code>xs:string</code>	<code>java.lang.String</code>	文字列フィールドに定義されているデフォルトの戦略については、 文字列フィールドのデフォルトの戦略 のセクションで詳しく説明しています。
<code>xs:boolean</code>	<code>java.lang.Boolean</code>	
<code>xs:decimal</code>	<code>java.math.BigDecimal</code>	値が 15 の <code>totalDigits</code> ファセットは、マップされたテーブル (履歴テーブルまたはレプリケートテーブル) に含まれる 10 進フィールドにデフォルトで追加されます。ただし、このファセットは、データモデルでより大きな値で上書きできます。
<code>xs:dateTime</code>	<code>java.util.Date</code>	
<code>xs:time</code>	<code>java.util.Date</code>	返される <code>Date</code> の日付部分は常に「1970/01/01」に設定されます。
<code>xs:date</code>	<code>java.util.Date</code>	返される <code>Date</code> の時間部分は、常に 1 日の始まり、つまり「00:00:00」です。
<code>xs:anyURI</code>	<code>java.net.URI</code>	
<code>xs:Name</code> (<code>xs:string</code> restriction)	<code>java.lang.String</code>	
<code>xs:int</code> (<code>xs:decimal</code> 制限項)	<code>java.lang.Integer</code>	

XML スキーマタイプ	JAVA クラス	注意
<code>xs:integer</code> (<code>xs:decimal</code> 制限項)	<code>java.lang.Integer</code>	このマッピングは、XMLスキーマの推奨事項に準拠していません。XMLスキーマ仕様では、 <code>xs:integer</code> には値スペースの制限がないと記載されていますが、実際には、この値スペースは <code>java.lang.Integer</code> オブジェクトの Java 仕様によって制限されています。

XML スキーマタイプと Java タイプの間のマッピングについては、[データタイプのマッピング](#) のセクションで詳しく説明しています。

XML スキーマの名前付き単純型

名前付き単純型は、既存のビルトイン単純型を再定義するためのデータモデルを設計するときに定義できます。名前付きの単純型は、データモデルで再利用できます。

制限事項

- データモデルでは、名前付きの単純型ではエレメント `restriction` のみが許可され、それでも、制限による派生のみがサポートされます。特に、エレメント `list` および `union` はサポートされていません。
- ファセット定義は累積的ではありません。つまり、エレメントとその名前付きタイプの両方が同じ種類のファセットを定義する場合、そのタイプで定義されたファセットはローカルファセット定義によって上書きされます。

ただし、この制限はエレメント `osd:constraint` によって定義されるプログラムファセットには適用されません。`osd:constraint` の場合、エレメントとその名前付き型の両方が異なる Java クラスを持つプログラムファセットを定義する場合、これらのファセットの定義は累積されます。

XML スキーマ仕様に反して、EBX® は、エレメント内の同じ種類のファセットとその名前付き型の定義に関して厳密ではありません。つまり、エレメントで定義された同じ種類のファセットの値は、指定された型で定義されたものに従ってチェックされません。ただし、エレメントとその型の両方で定義された静的列挙の場合、ローカル列挙はこれらの列挙の共通部分に置き換えられます。

- エレメントとその名前付きタイプの両方で異なるタイプの列挙を定義することはできません。たとえば、エレメントに静的列挙型を指定したり、名前付き型に動的列挙型を指定したりすることはできません。
- エレメントとその名前付きタイプの両方でパターンファセットを同時に定義することはできません。

XML スキーマ複合型

データモデルを設計するときに、複雑な型を定義できます。*名前付き複合型* はデータモデルで再利用できます。

制限事項

- データモデルでは、エレメント `sequence` のみが許可されます。特に、属性定義はサポートされていません。
- タイプ拡張は、EBX® の現在のバージョンではサポートされていません。

EBX® で定義された拡張単純型

EBX® は、事前定義された単純なデータ型を提供します。

XML スキーマタイプ	JAVA クラス
<code>osd:text</code> (<code>xs:string</code> restriction)	<code>java.lang.String</code>
<code>osd:html</code> (<code>xs:string</code> restriction)	<code>java.lang.String</code>
<code>osd:email</code> (<code>xs:string</code> restriction)	<code>java.lang.String</code>
<code>osd:password</code> (<code>xs:string</code> restriction)	<code>java.lang.String</code>
<code>osd:color</code> (<code>xs:string</code> restriction)	<code>java.lang.String</code>
<code>osd:resource</code> (<code>xs:anyURI</code> restriction)	内部クラス
<code>osd:locale</code> (<code>xs:string</code> restriction)	<code>java.util.Locale</code>
<code>osd:dataspaceKey</code> (<code>xs:string</code> restriction)	<code>java.lang.String</code>
<code>osd:datasetName</code> (<code>xs:string</code> restriction)	<code>java.lang.String</code>

上記のタイプは、内部スキーマ `common-1.0.xsd` によって定義されています。それらは次のように定義されます。

<code>osd:text</code>	このタイプは、テキスト情報を表します。EBX® のデフォルトのユーザーインターフェイスは、入力と表示用に数行の専用エディターで構成されています。 このデータ型の使用は、非常に長いテキストを対象としています。これが、<code>'Text'</code> 検索戦略がこのタイプに関連付けられており、変更できない理由です。結果として、タイプ <code>osd:text</code> のフィールドはソートの恩恵を受けることができません。さらに、それらのインデックス作成戦略と検索結果は、<code>xs:string</code> タイプのものとは異なります。詳細については、文字列フィールドのデフォルト戦略 を参照してください。 <pre><xs:simpleType name="text"> <xs:restriction base="xs:string" /> </xs:simpleType></pre>
<code>osd:html</code>	これは、HTML形式の文字列を表します。WYSIWYGエディターはEBX®で提供されます。 <pre><xs:simpleType name="html"> <xs:restriction base="xs:string" /> </xs:simpleType></pre>
<code>osd:email</code>	これは、RFC822 標準で指定されているメールアドレスを表します。

	<pre><xs:simpleType name="email"> <xs:restriction base="xs:string" /> </xs:simpleType></pre>
<code>osd:password</code>	これはハッシュ化または暗号化されたパスワードを表します。EBX® には専用のエディターが用意されています。 <code><xs:element name="password" type="osd:password"></code> デフォルトのエディターは、SHA-512 アルゴリズムを使用してハッシュ計算を実行します。この暗号化機能は、メソッド <code>DirectoryDefault.encryptString</code> を使用して Java クライアントからも利用できます。 <code>Encryption</code> インターフェイスを実装するクラスを指定することにより、デフォルトのエディターで別の暗号化メカニズムを使用することもできます。 <code><xs:element name="password" type="osd:password"></code> <pre> <xs:annotation> <xs:appinfo> <osd:uiBean class="com.orchestranetworks.ui.password.PasswordEditor"> <encryptionClass>package.Encryptor</encryptionClass> </osd:uiBean> </xs:appinfo> </xs:annotation> </xs:element></pre>

	別のフィールドへのパスを参照し、インターフェイス <code>HashComputation</code> を実装するクラスを使用することで、ソルトを指定することができます。 <pre><xs:element name="password" type="osd:password"> <xs:annotation> <xs:appinfo> <osd:uiBean class="com.orchestranetworks" <encryptionClass>package.HashClass <saltPath>../login</saltPath> </osd:uiBean> </xs:appinfo> </xs:annotation> </xs:element></pre>
<code>osd:locale</code>	これは、地理的、政治的、または文化的な場所を表します。 ロケールタイプは、クラス <code>java.util.Locale</code> によって Java に変換されます。 <pre><xs:simpleType name="locale"> <xs:restriction base="xs:string"> <xs:enumeration value="ar" osd:label="Ar" <xs:enumeration value="ar_AE" osd:label="Ar_AE" <xs:enumeration value="ar_BH" osd:label="Ar_BH" <xs:enumeration value="ar_DZ" osd:label="Ar_DZ"</pre>

	<pre><xs:enumeration value="ar_EG" osd:label= <xs:enumeration value="ar_IQ" osd:label= ... <xs:enumeration value="vi_VN" osd:label= <xs:enumeration value="zh" osd:label="Ch <xs:enumeration value="zh_CN" osd:label= <xs:enumeration value="zh_HK" osd:label= <xs:enumeration value="zh_TW" osd:label= </xs:restriction> </xs:simpleType></pre>
<code>osd:color</code>	これは、16 進数の RGB カラー形式の文字列を表します。EBX® にはカラーピッカー <code>UIComponent</code> が用意されています。 <pre><xs:simpleType name="color"> <xs:restriction base="xs:string" /> </xs:simpleType></pre>
<code>osd:resource</code>	これはモジュールにパッケージ化されたリソースを表します。詳細については、パッケージ化されたリソース を参照してください。このタイプには、ファセット <code>FacetOResource</code> の定義が必要です。 <pre><xs:simpleType name="resource"> <xs:restriction base="xs:anyURI" /> </xs:simpleType></pre>

`osd:dataspaceKey`

この型はデータスペースへの参照を表します。

```
<xs:element name="dataspaceField" type="osd:dataspaceKey"/>
```

EBX® には、参照可能なデータスペースを表示する専用のエディターが用意されています。

`xs:annotation/xs:appInfo` の下のエレメント `osd:dataspaceSet` を使用して参照できるデータスペースを指定できます。エレメント `osd:dataspaceSet` が定義されていない場合は、デフォルトでは開いているブランチのみを参照できます。

```
<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 | all</includeDescendants>
```

```
</include>
```

```
<exclude>
```

```
<pattern>a pattern</pattern>
```

```
<type>all | branch | version</type>
```

```
<includeDescendants>none | all</includeDescendants>
```

```
</exclude>
```

```

        <filter osd:class="com.foo.MyDat

        <param1>...</param1>

        <param2>...</param2>

    </filter>

</osd:dataspaceSet>

</xs:appinfo>

</xs:annotation>

</xs:element>

```

- includes

このフィールドで参照できるデータスペースを指定します。少なくとも **include** を定義する必要があります。

pattern : データスペースをフィルタリングするパターンを指定します。パターンは、データスペースの名前と照合されます。このプロパティは必須です。

type : このフィールドで参照できるデータスペースのタイプを指定します。定義されていない場合、この制限はブランチに適用されます。 **all** の場合、ブランチとスナップショットが含まれます。 **branch** の場合、ブランチのみが含まれます。 **snapshot** の場合、スナップショットのみが含まれます。設定されていない場合、このプロパティはデフォルトで **branch** になります。

includeDescendants : 指定されたパターンに一致するデータスペースの子ま

たは子孫をセットに含めるかどうかを指定します。 **none** の場合、指定されたパターンに一致するデータスペースの子も子孫も含まれません。

allDescendants の場合、指定されたパターンに一致するデータスペースのすべての子孫が含まれます。

allBranchDescendants の場合、指定されたパターンに一致するデータスペースのすべての子孫ブランチが含まれます。 **allSnapshotDescendants** の場合、指定されたパターンに一致するデータスペースのすべての子孫スナップショットが含まれます。

directBranchChildren の場合、指定されたパターンに一致するデータスペースの直接ブランチのみが含まれます。

directSnapshotChildren の場合、指定されたパターンに一致するデータスペースの直接スナップショットのみが含まれます。設定されていない場合、このプロパティはデフォルトで **none** になります。

- **excludes**

このフィールドで参照できないデータスペースを指定します。包含が定義されていない場合、除外は無視されます。

pattern : データスペースをフィルタリングするパターンを指定します。パターンは、データスペースの名前と照合されます。このプロパティは必須です。

type : このフィールドで参照できるデータスペースのタイプを指定します。定義されていない場合、この制限はブラ

ンチに適用されます。 `all` の場合、ブランチとスナップショットは除外されません。 `branch` の場合、ブランチのみが除外されます。 `snapshot` の場合、スナップショットのみが除外されます。設定されていない場合、このプロパティはデフォルトで `branch` になります。

`includeDescendants` : 指定されたパターンに一致するデータセットの子または子孫をセットから除外するかどうかを指定します。 `none` の場合、指定されたパターンに一致するデータスペースの子も子孫も除外されません。

`allDescendants` の場合、指定されたパターンに一致するデータスペースのすべての子孫が除外されます。

`allBranchDescendants` の場合、指定されたパターンに一致するデータスペースのすべての子孫ブランチが除外されます。 `allSnapshotDescendants` の場合、指定されたパターンに一致するデータスペースのすべての子孫スナップショットが除外されます。

`directBranchChildren` の場合、指定されたパターンに一致するデータスペースの直接ブランチのみが除外されます。

`directSnapshotChildren` の場合、指定されたパターンに一致するデータスペースの直接スナップショットのみが除外されます。設定されていない場合、このプロパティはデフォルトで `none` になります。

- `filter`

	データセットまたはレコードのコンテキストでデータスペースを受け入れるか拒否するフィルターを指定します。このフィルターは、このフィールドに関連付けられている専用の入力コンポーネントでのみ使用されます。つまり、このフィールドを検証するとき、このフィルターは使用されません。特定の制約を使用して、このフィールドで特定の制御を実行できます。 属性 <code>osd:class</code> は、インターフェイス <code>DataspaceSetFilter</code> を実装するJava Bean を指定します。 <code>xs:annotation/xs:appInfo/</code> <code>osd:dataspaceSet</code> の下のエレメント <code>validation</code> を使用して、このタイプに関連付けられた検証メッセージと制御ポリシーをカスタマイズすることもできます。詳細については、重大度のあるファセット検証メッセージ および 制御ポリシー を参照してください。
<code>osd:datasetName</code>	このタイプは、データセットへの参照を表します。 <code><xs:element name="dataset" type="osd:datasetName</code> EBX® で提供される特定のエディターは、参照可能なデータセットを表示します。 <code>xs:annotation/xs:appInfo</code> の下のエレメント <code>osd:datasetSet</code> を使用して参照できるデータセットを指定することもできます。 <code><xs:element name="datasetField" type="osd:dataset</code> <code><xs:annotation></code>

```

<xs:appinfo>

    <osd:datasetSet>

        <branch>productsBranch</branch>

        <version>productsVersion</version>

        <dataspaceSelector>../dataspaceF

        <pattern>a pattern</pattern>

        <filter osd:class="com.foo.MyDat

            <param1>...</param1>

            <param2>...</param2>

        </filter>

    </osd:datasetSet>

</xs:appinfo>

</xs:annotation>

</xs:element>

```

- **branch**

ソースブランチを指定します。このブランチに含まれるデータセットのみが、データセット識別子 (osd:datasetName) タイプのフィールドで選択できます。

- **version**

ソーススナップショットを指定します。このスナップショットに含まれるデータセットのみが、データセット識別子 (osd:datasetName) タイプのフィールドで選択できます。

- `dataspaceSelector`

参照できるデータセットを含むデータスペースを定義する同じデータモデル内のフィールドを指定します。指定されたフィールドは `xs:string` または `osd:dataspaceKey` 型でなければなりません。このフィールドの値は、データスペースまたはスナップショットの永続的な識別子の表現に準拠する必要があります。

参照されるノードは、動的ファセットに存在する制限に従う必要があります。

以下も参照してください。:

- `HomeKey.format`
- [動的制約](#)

- `includes`

このフィールドで参照できるデータセットを指定します。

`pattern`: データセットをフィルタリングするパターンを指定します。パターンは、データセットの名前と照合されます。このプロパティは必須です。

`includeDescendants`: 指定されたパターンに一致するデータセットの子または子孫をセットに含めるかどうかを指定します。 `none` の場合、指定されたパターンに一致するデータセットの子も子孫も除外されません。

`directChildren` の場合、指定されたパターンに一致するデータセットの直接の子のみが除外されます。

`allDescendants` の場合、指定された

パターンに一致するデータセットのすべての子孫が除外されます。設定されていない場合、このプロパティはデフォルトで `none` になります。

- `excludes`

このフィールドで参照できないデータセットを指定します。包含が定義されていない場合、除外は無視されます。

`pattern` : データセットをフィルタリングするパターンを指定します。パターンは、データセットの名前と照合されます。このプロパティは必須です。

`includeDescendants` : 指定されたパターンに一致するデータセットの子または子孫をセットに含めるかどうかを指定します。 `none` の場合、指定されたパターンに一致するデータセットの子も子孫も除外されません。

`directChildren` の場合、指定されたパターンに一致するデータセットの直接の子のみが除外されます。

`allDescendants` の場合、指定されたパターンに一致するデータセットのすべての子孫が除外されます。設定されていない場合、このプロパティはデフォルトで `none` になります。

- `filter`

データセットまたはレコードのコンテキストでデータセットを受け入れるか拒否するフィルターを指定します。このフィルターは、このフィールドに関連付けられている専用の入力コンポーネントでのみ使用されます。つまり、このフィールドを検証するとき、このフィルターは使

用されません。特定の制約を使用して、このフィールドで特定の制御を実行できます。

属性 `osd:class` は、インターフェイス `DatasetSetFilter` を実装する Java Bean を指定します。入力データスペース参照がこのフィルターと一致しない場合、検証メッセージが関連フィールドに追加されます。

エレメント `branch`、`version`、または `dataspaceSelector` のいずれかを定義する必要があります。

`xs:annotation/xs:appInfo/osd:datasetSet` の下のエレメント `validation` を使用して、このタイプに関連付けられた検証メッセージと制御ポリシーをカスタマイズすることもできます。詳細については、[重大度のあるファセット検証メッセージ](#) および [制御ポリシー](#) を参照してください。

EBX® によって定義された複合型

EBX® は、事前定義された複雑なデータ型を提供します。

XML スキーマタイプ	説明
<code>osd:UDA</code>	**ユーザー定義属性：** このタイプを使用すると、アクセス権に応じて、すべてのユーザーが UDA カタログと呼ばれるディクショナリで定義された属性に関連付けられた値を定義できます。
<code>osd:UDACatalog</code>	**ユーザー定義属性のカタログ：** このタイプは、属性を指定できるテーブルで構成されます。このカタログは、同じデータモデルで宣言

XML スキーマタイプ	説明
	されたすべての <code>osd:UDA</code> エレメントによって使用されます。
<code>osd:UDA</code>	ユーザー定義属性 (UDA) は、<code>minOccurs</code> 属性と <code>maxOccurs</code> 属性の両方、および対応するカタログのパスを指定する属性 <code>osd:UDACatalogPath</code> もサポートします。 <pre><xs:element name="firstUDA" type="osd:UDA" minOccurs="1" maxOccurs="unbounded" osd:UDACatalogPath="//userCatalog" /></pre> <pre><xs:element name="secondUDA" type="osd:UDA" minOccurs="1" maxOccurs="unbounded" osd:UDACatalogPath="/root/userCatalog" /></pre> <pre><xs:element name="thirdUDA" type="osd:UDA" minOccurs="1" maxOccurs="unbounded" osd:UDACatalogPath="//userCatalog" /></pre> マネージャーでは、UDA を操作するときに、エディターは選択した属性のタイプに適應します。
<code>osd:UDACatalog</code>	内部的には、カタログはテーブルとして表されます。パラメーター <code>minOccurs</code> および <code>maxOccurs</code> を指定する必要があります。 同じデータモデルで複数のカタログを定義できます。 <pre><xs:element name="insuranceCatalog" type="osd:UDACatalog"> <xs:annotation> <xs:documentation xml:lang="en-US">Insurance Catalog</xs:documentation> <xs:documentation xml:lang="fr-FR">Catalogue d'assurance</xs:documentation> </xs:annotation> </xs:element></pre>

```
</xs:element>
```

```
<xs:element name="userCatalog" type="osd:UDACatalog">
```

```
  <xs:annotation>
```

```
    <xs:documentation xml:lang="en-US">User Catalog</xs:documentation>
```

```
    <xs:documentation xml:lang="fr-FR">Catalogue des utilisateurs</xs:documentation>
```

```
  </xs:annotation>
```

```
</xs:element>
```

新しい属性の作成に使用できるのは、次のタイプのみです。

- 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

ユーザー定義の属性とカタログの制限

次の機能は、UDA エlement ではサポートされていません。

- ファセット
- `osd:function` プロパティを使用する関数
- `osd:uiBean` プロパティを使用する UI Bean エディター
- `osd:checkNullInput` プロパティ
- 履歴機能
- レプリケーション
- `osd:inheritance` プロパティを使用した継承機能

UDA カタログは内部的にテーブルと見なされるため、テーブルに適用される制限は `UDACatalog` エlement にも存在します。

集約リスト

XML スキーマでは、Element が出現できる最大回数は、その宣言の `maxOccurs` 属性の値によって決まります。この値が厳密に 1 より大きい、`unbounded` である場合、データは複数回出現する可能性があります。`osd:table` 宣言が含まれていない場合、この Element は 集約リスト と呼ばれます。

Java では、クラス `java.util.List` のインスタンスとして表されます。

以下は、借入額に応じてローン商品の価格を定義する集計リストの例です。

```
<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>

```

集約リストには、EBX® 専用のエディターがあります。このエディターを使用すると、オカレンスを追加または削除できます。

注意:

`maxOccurs > 1` のエレメントに `osd:table` 宣言を追加することは、設計プロセス中に考慮しなければならない非常に重要な考慮事項です。集約リストは、テーブルでサポートされている多くの機能に関して厳しく制限されています。テーブルでサポートされている集約リストでサポートされていない機能は次のとおりです。

- パフォーマンスとメモリの最適化
- ルックアップ、フィルター、検索
- 階層での並べ替え、表示、表示
- ID の制約 (主キーと一意性の制約)
- レコードレベルでの作成、変更、削除、および特定の権限に関する詳細な権限
- 詳細な比較とマージ

したがって、集計リストは、高度な要件なしで、少量の単純なデータ (1 から 2 ダースのオカレンス) にのみ使用する必要があります。大量のデータまたはより高度な機能の場合は、`osd:table` 宣言を使用することを強くお勧めします。

テーブル宣言の詳細については、[テーブルと関係](#) を参照してください。

外部データモデルの包含

現在のモデルに別のデータモデルを含めると、そのデータモデルで定義されている再利用可能なタイプを使用できます。したがって、外部データモデルを含めることで、複数の XML スキーマドキュメント間でデータ型を共有できます。

モデルに別のXMLスキーマドキュメントを含め、それによってモデルが定義するデータ型を含めるには、次のように `xs:include` エレメントを指定します。

```
<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>
```

属性 `schemaLocation` は必須であり、含める XML スキーマドキュメントへの絶対パスまたは相対パスを指定する必要があります。

XML スキーマドキュメントを含めることはネームスペースに対応していないため、含まれるすべてのデータ型は同じネームスペースに属している必要があります。結果として、同じ名前のデータ型を定義する XML スキーマドキュメントを含めることはサポートされていません。

EBX® には、組み込みデータモデルおよびモジュールにパッケージ化されたデータモデルを含めるための特定の URN を備えた拡張機能が含まれています。

埋め込みデータモデルをモデルに含めるには、EBX® で定義された URN を指定します。以下に例を示します。

```
<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>
```

モジュールにパッケージ化されたデータモデルを含めるには、EBX® で定義された特定の URN を指定します。以下に例を示します。

```
<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>
```

EBX® でサポートされている特定の URN の詳細については、`SchemaLocation` を参照してください。

注意:

パッケージ化されたデータモデルが Java リソースを使用する場合、データモデルを含むモジュールのクラスローダーは、これらのリソースを解決するために実行時に使用されます。

テーブルと関係

- テーブル
 - 概要
 - 宣言
 - 共通のプロパティ
 - 例
 - データセットの継承に関連するプロパティ
 - ツールバーの使用
 - 例
 - レコードメタデータ
- 外部キー
 - 宣言
 - 例

- 関連付け
 - 概要
 - 宣言
 - ユーザーインターフェイスの統合
 - 検証
 - データサービスの統合
 - 例
- リンクされたフィールド
 - 概要
 - 宣言

テーブル

概要

TIBCO EBX® は、大量のレコードの処理や主キーによる識別など、リレーショナルデータベーステーブルの機能をサポートしています。

テーブルには、[集計リスト](#)では提供されない多くの利点があります。リレーショナル機能以外に、テーブルが提供するいくつかの機能は次のとおりです。

- フィルターと検索
- 並べ替え、ビュー、および階層
- ID 制約：主キー、[外部キー](#)、一意性制約
- 作成、変更、および削除のための特定の権限
- 個々のレコードレベルでの動的およびコンテキスト権限
- 詳細な比較とマージ
- レコードレベルで継承する機能 ([データセットの継承](#)を参照)
- パフォーマンスとメモリの最適化

以下も参照してください。:

- エンティティマッピング
- 外部キー
- 関連付け
- リンクされたフィールド
- 既存のデータセットの操作
- 単純な表形式のビュー
- 階層ビュー
- History

宣言

`maxOccurs> 1` のエレメントであるテーブルエレメントは、次のアノテーションを追加することで宣言されます。

```
<xs:annotation>

  <xs:appinfo>

    <osd:table>

      <primaryKeys>/pathToField1 /pathToField...n</primaryKeys>

    </osd:table>

  </xs:appinfo>

</xs:annotation>
```

注意:

データモデルにエンティティ名が明示的に設定されていない場合、デフォルトのエンティティ名がテーブルに関連付けられます。テーブルの名前は、デフォルトのエンティティ名として使用されます。エンティティ名を使用すると、WSDL 操作のサフィックスと SQL リクエストのエイリアスを定義することで、パスの代わりに一意の名前を使用して、SQL 要求と

データサービス操作でテーブルを参照できます。詳細については、[エンティティマッピング](#)を参照してください。

共通のプロパティ

エレメント	説明	必須
<code>primaryKeys</code>	テーブルの主キーフィールドを指定します。 主キーの各フィールドは、テーブルのルートエレメントのすぐ下から始まる絶対XPath 表記で示す必要があります。主キーに複数のフィールドがある場合、リストはスペースで区切られます。 注意：タイプ <code>xs:string</code> の主キーの空白は異なる方法で処理されます。文字列型の主キーの空白処理を参照してください。	はい
<code>defaultLabel</code>	レコードのエンドユーザー表示を定義します。複数のバリエーションを指定できます。 ローカライズされていない静的な式は、<code>defaultLabel</code> エレメントを使用して定義されます。次に例を示します。	いいえ

エレメント	説明	必須
	<pre><defaultLabel>Product: \${../productCode}</ defaultLabel></pre> - 静的ローカライズされた式は、属性 <code>xml:lang</code> を持つ <pre><defaultLabel xml:lang="fr-FR">Produit : \${../ productCode}</ defaultLabel></pre> <pre><defaultLabel xml:lang="en-US">Product: \${../ productCode}</ defaultLabel></pre> - インターフェイス <code>UILabelRenderer</code> および/またはインターフェイス <code>UILabelRendererForHierarchy</code> を実装するJavaBean。JavaBean は、属性 <code>osd:class</code> を使用して指定されます。次に例を示します。	

エレメント	説明	必須
	<pre data-bbox="660 255 1182 394"><defaultLabel osd:class="com.wombat.MyLabel"></ defaultLabel></pre> 注意: エンドユーザーの表示は並べ替え可能である必要があるため、静的式では並べ替え可能な検索戦略を使用するフィールドのみが許可されます。 注意: ユーザーインターフェイスを表示するときのタグの優先度は次のとおりです。 1. JavaBean を使用した <code>defaultLabel</code> タグ (ただし、同じタイプの複数のレンダラーを定義することは許可されていません)。 2. <code>xml:lang</code> 属性を使用した静的ローカライズ式の <code>defaultLabel</code> タグ。 3. 静的なローカライズされていない式を持つ <code>defaultLabel</code> タグ。 注意: レコードラベルを表示する場合、関連するデータセットで定義されたアクセス権限は適用されません。アクセス権限の制限のために通常非表示になって	

エレメント	説明	必須
	いるフィールドは、ラベルに表示されます。	
<code>recordForm</code>	データセットのレコードフォームをカスタマイズするための特定のコンポーネントを定義します。このコンポーネントは、<code>UIForm</code> を拡張する JavaBean または <code>UserServiceRecordFormFactory</code> を実装する JavaBean を使用して定義されます。 JavaBean は、属性 <code>osd:class</code> を使用して指定されます。次に例を示します。 <pre><recordForm osd:class="com.wombat.MyRecordForm"/ ></pre>	いいえ

例

以下は、製品カタログの例です。

```
<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>

```

データセットの継承に関連するプロパティ

次のプロパティは、データセットの継承のコンテキストでのみ有効です。

エレメント	説明	必須
<code>onDelete- deleteOccultingChildren</code>	レコードの削除時に、非表示モードの子レコードも削除するかどうかを指定します。有効な値は、 <code>never</code> または <code>always</code> です。	いいえ。デフォルトは <code>never</code> です。
<code>mayCreateRoot</code>	ルートレコードの作成を許可するかどうかを指定します。式は、以下の構文に従う必要があります。定義モードを参照してください。	いいえ。デフォルトは <code>always</code> です。
<code>mayCreateOverwriting</code>	子データセットでレコードを上書きできるかどうかを指定します。式は、以下の構文に従う必要があります。定義モードを参照してください。	いいえ。デフォルトは <code>always</code> です。
<code>mayCreateOcculting</code>	子データセットでレコードを非表示にできるかどうかを指定します。式は、以下の構文に従う必要があります。定義モードを参照してください。	いいえ。デフォルトは <code>always</code> です。
<code>mayDuplicate</code>	レコードの複製を許可するかどうかを指定します。式は、以下の構文に従う必要があります。	いいえ。デフォルトは <code>always</code> です。

エレメント	説明	必須
<code>mayDelete</code>	レコードの削除を許可するかどうかを指定します。式は、以下の構文に従う必要があります。	いいえ。デフォルトは <code>always</code> です。

`may...` 式は、アクションがいつ可能かを指定しますが、アクションの最終的な可用性はユーザーのアクセス権限にも依存します。式の構文は次のとおりです。

```
expression ::= always | never | <condition>*
```

```
condition ::=
```

「always」：操作は「常に」可能です (ただし、ユーザー権限によって制限される場合があります)。

「never」：操作は不可能です。

「root:yes」：レコードがルートインスタンスにある場合、操作は可能です。

「root:no」：レコードがルートインスタンスにある場合、操作はできません。

レコードで特定の条件が定義されていない場合は、デフォルトが使用されます。

以下も参照してください。:

- データセットの継承

ツールバーの使用

`xs:annotation/appinfo/osd:table` の下のエレメント `defaultView/toolbars` を使用して、ユーザーインターフェイスに表示するツールバーを定義することができます。ツールバーを使

用すると、テーブルビュー、階層ビュー、またはレコードフォームを表示するときに表示するボタンとメニューをカスタマイズできます。

次の表は、`defaultView/toolbars` で定義できるエレメントを示しています。

エレメント	説明	必須
<code>tabularViewTop</code>	デフォルトのテーブルビュー の上で使用するツールバーを 定義します。	いいえ
<code>tabularViewRow</code>	デフォルトのテーブルビュー の各行で使用するツールバー を定義します。	いいえ
<code>recordTop</code>	レコードフォームで使用する ツールバーを定義します。	いいえ
<code>hierarchyViewTop</code>	テーブルのデフォルトの階層 ビューで使用するツールバー を定義します。	いいえ

以下も参照してください。:

- ツールバー

例

以下は、製品カタログで使用するカスタムツールバーの例です。

```
<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>
```

注意:

ツールバーが存在しないか、特定の場所で使用できない場合、対応する場所のユーザーインターフェイスにツールバーは表示されません。

レコードメタデータ

EBX® は、テーブルの構造にレコードメタデータを自動的に追加します。メタデータノードを使用すると、レコードに関する次の情報を取得できます。

- 作成者
- 作成時刻
- 最終変更者
- 最終変更時刻
- レコードの一意の識別子 (UUID)。このメタデータは技術的なものであり、カスタムテーブルビュー、構造化検索ツール、およびクイック検索では使用できません。
- レコードの定義モード。このメタデータは技術的なものであるため、カスタムテーブルビュー、構造化検索ツール、クイック検索では使用できません。

レコードメタデータの使用例としては、特定の日付以降に特定のユーザーによって作成されたすべてのレコードを表示または取得することがあります。次のことが可能です。

- ユーザーインターフェイスのビジネスユーザーの場合
 - 検索条件の一部としてメタデータフィールドを使用して、クイック検索機能を使用してレコードを検索します。
 - 必要なメタデータフィールドを含むテーブルビューを定義します。
- API を介して、XPath または SQL を使用してクエリを実行します。

注意:

メタデータに関しては、次の点に注意してください。

- メタデータは、予約名 `ebx-metadata` を持つグループの下で定義されます。結果として、テーブルのルートの直下に同じ名前のグループが含まれている場合、メタデータはテーブルで使用できません。テーブルがこの予約名でグループを定義している場合、データモデルのコンパイル時に警告が表示されます。
- メタデータは読み取り専用です。
- テーブルのコンテンツを XML または CSV ファイルにエクスポートする場合、メタデータは含まれません。
- デフォルトでは、API を使用してデータモデルをインストロスペクトする場合、メタデータフィールドは含まれません。専用の [API](#) を使用して明示的に含めることができます。

以下も参照してください。:

- [SQL のメタデータフィールド](#)
- [技術データ](#)
- `SchemaMetadataConstants`
- `Constants.Data`

外部キー

宣言

`table` への参照は、拡張ファセット `osd:tableRef` を使用して定義されます。

`osd:tableRef` 宣言を保持するノードは、タイプ `xs:string` にする必要があります。

インスタンス化時に、ノードの任意の値は、[主キー構文](#)を使用してターゲットテーブル内のレコードを識別します。

この拡張ファセットは、値がターゲットテーブルのレコードを参照する列挙としても解釈されません。

エレメント	説明	必須
<code>tablePath</code>	ターゲットテーブルを指定する XPath 式。	はい
<code>container</code>	ターゲットテーブルを含むデータセットの参照。	データスペースエレメントが定義されている場合のみ。それ以外の場合、デフォルトは現在のデータセットです。
<code>branch</code>	コンテナデータセットを含むデータスペースの参照。	いいえ。デフォルトは現在のデータスペースまたはスナップショットです。
<code>display</code>	現在のレコードで選択された外部キーと可能なキーのソートされたリストを表示するためのカスタム表示。パターンベースの式、またはニーズが非常に具体的な場合は JavaBean の2つのバリエーションを指定できます。 静的な式は、<code>display</code> エレメントと <code>pattern</code> エレメントを使用して指定されます。これらの静的な式は、<code>pattern</code> エレメントの追加属性 <code>xml:lang</code> を使用してローカライズできます。次に例を示します。 <pre><display></pre>	いいえ。 <code>display</code> プロパティが指定されていない場合、 テーブルのレコードレンダリング が使用されます。

エレメント	説明	必須
	<pre><pattern>Product : \${./productCode}</ pattern></pre> <pre><pattern xml:lang="fr- FR">Produit : \${./ productCode}</ pattern></pre> <pre><pattern xml:lang="en- US">Product: \${./ productCode}</ pattern></pre> <pre></display></pre> 注意: 表示パターンはソート可能である必要があるため、ソート可能な検索戦略を使用するフィールドのみが許可されます。 • インターフェイス <code>TableRefDisplay</code> を実装する JavaBean。これは、属性 <code>osd:class</code> を使用して指定されます。次に例を示します。 <pre><display osd:class="com.wombat.MyLabel"></ display></pre>	

エレメント	説明	必須
	注意: JavaBean が定義されている場合、またはターゲットテーブルがレコードの プログラムラベル を定義している場合、ユーザーインターフェイスでのクイック検索およびソート操作では、レコードのラベルの代わりに生の値が使用されます。これらの操作でレコードのラベルを使用するには、静的式を定義する必要があります。 同じ外部キーエレメントで両方のバリエーションを定義することはできません。 注意: レコードラベルを表示する場合、関連するデータセットで定義されたアクセス権限は適用されません。アクセス権限の制限のために通常非表示になっているフィールドは、ラベルに表示されます。	
filter	ターゲットテーブルのレコードをフィルタリングする追加の制約を指定します。2 種類のフィルターを使用できます。	いいえ

エレメント	説明	必須
	• XPath フィルターは、ターゲットテーブルコンテキストの XPath 述語です。 <code>predicate</code> エレメントを使用して指定されます。次に例を示します。 <pre data-bbox="660 658 1038 801"><filter><predicate>type = \${../refType}</predicate></filter></pre> ローカライズされた検証メッセージは、エレメント <code>validationMessage</code> を使用して指定できます。このメッセージは、レコードがフィルターによって受け入れられない場合、検証時にエンドユーザーに表示されます。 特定の重大度レベルは、ネストされた <code>severity</code> エレメントで定義できます。デフォルトの重大度は「error」です。 ローカライズされた各メッセージバリエーションは、ネストされた <code>message</code> エレメントで定義され、ロケールは	

エレメント	説明	必須
	<code>xml:lang</code> 属性にあります。サポートされていないロケールのデフォルトメッセージを指定するには、<code>xml:lang</code> 属性のない <code>message</code> エレメントを定義します。 ユーザーインターフェイスでは、XPath フィルターを適用して、外部キーフィールドの値に従ってテーブルをフィルター処理します。つまり、外部キーフィールドがデータモデルで XPath フィルターを指定している場合、それはフィルターペインで再利用され、フィルターペインに表示される関連するコンボボックスの値のセットを制限します。ただし、フィルターペインで 사용되는述語は、述語の非コンテキスト部分のみを考慮します。 プログラムフィルターは、インターフェイス <code>TableRefFilter</code> を実装する JavaBean です。これは、属性 <code>osd:class</code> を使用して	

エレメント	説明	必須
	指定されます。次に例を示します。 <pre data-bbox="660 387 1198 524"><filter osd:class="com.wombat.MyFilter"></ filter></pre> プログラムフィルターのセットアップ中に、追加の検証メッセージを指定できます。 ユーザーインターフェイスでは、プログラムフィルターは、フィルターペインに表示される関連するコンボボックスの値のセットをフィルター処理するために適用されません。ただし、ユーザーインターフェイスのフィルターペインで使用する追加の XPath 述語を指定することは可能です。この XPath 述語は、メソッド <pre data-bbox="660 1525 1318 1554">TableRefFilterContext.setFilterForSearch</pre> を使用したプログラムフィルターのセットアップ中に指定されます。 注意: 属性 <code>osd:class</code> とプロパティ <code>predicate</code> を同時に設定することはできません。検証検索の XPath 関数	

エレメント	説明	必須
	は、 <code>tableRef</code> フィルターでは禁止されています。以下も参照してください。: • JavaBean 仕様 • <code>JavaBeanVersion</code>	
<code>validation</code>	<code>osd:tableRef</code> およびエラー管理ポリシーのローカライズされた検証メッセージを指定します。 特定の重大度レベルは、ネストされた <code>severity</code> エレメントで定義できます。デフォルトの重大度は「error」です。 エラー管理ポリシーは、ネストされた <code>blocksCommit</code> エレメントで定義できます。すべての操作をブロックするエラー管理ポリシーは、フィルターには適用されません。つまり、参照レコードが存在するが外部キーフィルタを満たさない場合、外部キー制約はブロックされません。この場合、更新は拒否されず、検証エラーが報告されます。 ローカライズされた各メッセージバリエーションは、ネストされた <code>message</code> エレメントで定義され、ロケール	いいえ

エレメント	説明	必須
	は <code>xml:lang</code> 属性にあります。サポートされていないロケールのデフォルトメッセージを指定するには、 <code>xml:lang</code> 属性のない <code>message</code> エレメントを定義します。	

注意:

リポジトリに存在しないコンテナへの外部キーを持つデータセットを作成できます。ただし、このデータセットのコンテンツは、コンテナが作成されるまで利用できません。コンテナの作成後、データセットを使用可能にするには、データモデルを更新する必要があります。まだ存在しないコンテナを参照するデータセットを作成する場合、次の制限が適用されます。

- データセットレベルで定義されたトリガーは実行されません。
- テーブルに含まれていないフィールドのデフォルト値は初期化されません。
- アーカイブのインポート中に、存在しないコンテナを参照するデータセットを作成することはできません。

例

以下の例では、「Products」テーブルの外部キーを「Catalogs」テーブルのレコードに指定しています。

```
<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>

```

以下も参照してください。:

- テーブル定義
- 主キーの構文
- 外部キーの抽出 (XPath 述語構文)
- 関連付け
- 高度なセレクトビュー
- `SchemaNode.getFacetOnTableReference`
- `SchemaFacetTableRef`

関連付け

概要

関連付けは、データモデルの既存の関係を抽象化し、ユーザーインターフェイスとデータサービスで関連付けされたオブジェクトをモデル主導で簡単に統合できるようにします。

いくつかのタイプの関連付けがサポートされています。

- * 「外部キーによる」* 関連付けは、既存の外部キーフィールドの逆の関係を指定します。
- * 「リンクテーブルを介した」* 関連付けは、中間リンクテーブルに基づいて関係を指定します (このようなテーブルは「結合テーブル」と呼ばれることがよくあります)。このリンクテーブルは、2 つの外部キーを定義する必要があります。1 つは「ソース」テーブル (関連付けエレメントを保持するテーブル) を参照し、もう 1 つは「ターゲット」テーブルを参照します。
- * 「XPath 述語による」* 関連付けは、XPath 述語に基づく関係を指定します。

関連付けの場合、次のことも可能です。

- 追加の XPath フィルターを指定して、関連するオブジェクトをフィルター処理します。
- 表形式のビューを構成して、関連するテーブルに表示する必要のあるフィールドを定義します。
- 関連するオブジェクトをフォームにレンダリングする方法を定義します。
- データサービスの「Select」操作で関連オブジェクトを非表示/表示します。データサービスのフィールドの非表示を参照してください。
- 必要な関連オブジェクトの最小数と最大数を指定します。

- 関連するオブジェクトを制限するために XPath 述語を使用して検証制約を追加します。

以下も参照してください。:

- `SchemaNode.getAssociationLink`
- `SchemeNode.isAssociationNode`
- `AssociationLink`

宣言

関連付けは、`xs:annotation/appInfo` の下の XML スキーマエレメント `osd:association` を使用してデータモデルで定義されます。

制限事項

- 関連付けは、タイプ `xs:string` の単純なエレメントにする必要があります。
- 関連付けは、テーブル内でのみ定義できます。

注意:

XML スキーマのインスタンスからは対応するノードが存在しないため、「公式」カードinality 制約 (`minOccurs="0" maxOccurs="0"`) が必要です。言い換えると、関連付けには値がなく、XML および XML スキーマに関する限り「仮想」エレメントと見なされます。

次の表は、`xs:annotation/appInfo/osd:association` で定義できるエレメントを示しています。

エレメント	説明	必須
<code>tableRefInverse</code>	外部キーの逆の関係である関連付けのプロパティを定義します。 エレメント <code>fieldToSource</code> は、関連付けのソーステーブルを参	関連付けが外部キーの逆の関係である場合は「はい」、それ以外の場合は「いいえ」。

エレメント	説明	必須
	照する外部キーを定義します。エレメント <code>fieldToSource</code> は必須であり、関連付けを含むテーブルを参照する外部キーフィールドを指定する必要があります。 エレメント <code>fieldToSource</code> は外部キーリスト (<code>maxOccurs</code> > 1) で定義することができ、その場合、リストはブロッキング一意性制約 (<code>onInsertUpdateOrDelete</code>) で一意として宣言する必要があります。 以下も参照してください。: • ブロッキングと非ブロッキングの制約 • 一意性の制約	
<code>linkTable</code>	リンクテーブルを介した関連付けのプロパティを定義します。 エレメント <code>table</code> は、関連付けで使用されるリンクテーブルを指定します。エレメント <code>table</code> は必須であり、既存のテーブルを参照する必要があります。	関連付けがリンクテーブル上にある場合は「はい」、それ以外の場合は「いいえ」。

エレメント	説明	必須
	重要：関連付けで使用するためには、リンクテーブルは、自動インクリメントのフィールドで構成される主キーや、関連付けのソースまたはターゲットテーブルへの外部キーを定義する必要があります。 エレメント <code>fieldToSource</code> は、関連付けのソーステーブルを参照する外部キーを定義します。エレメント <code>fieldToSource</code> は必須であり、関連付けを含むテーブルを参照する外部キーフィールドを指定する必要があります。 エレメント <code>fieldToTarget</code> は、関連付けのターゲットテーブルを参照する外部キーを定義します。エレメント <code>fieldToTarget</code> は必須であり、外部キーフィールドを指定する必要があります。	
<code>xpathLink</code>	<i>XPath 述語</i>に基づく関連付けのプロパティを定義します。 <code>predicate</code> エレメントは、現在のノードに関連す	関連付けが <i>XPath 述語</i>に基づいている場合は「はい」、それ以外の場合は「いいえ」。

エレメント	説明	必須
	る関連付けの基準を指定します。 例： <code>/root/Products[catalog_ref =\${../catalogId}]</code> または <code>//Products[catalog_ref =\${../catalogId}]</code> または <code>../Products[catalog_ref =\${../catalogId}]</code>。 述部へのパス (<code>../Products</code> など) は、関連付けのターゲットテーブルを指定します。パスのこの部分は、現在のテーブルに関して解決されます。関連付けが別のデータセットのテーブルを対象としている場合、相対パスを使用してテーブルを参照することはできません。 関連付けがソーステーブルのフィールドに依存する場合、XPath 式の述語には、<code>\${<relative-path>}</code> という表記を使用して依存するエレメントへの参照を含める必要があります。 <code>relative-path</code> は、関連付けノードに関連するエレメントを識別するパスです。	

エレメント	説明	必須
	EBX® XPath でサポートされている構文を参照してください。 注意: 検証検索 XPath 関数は、XPath リンクでは禁止されています。	
filter	predicate エレメントを使用して関連オブジェクトをフィルタリングする XPath 述語を定義します。次に例を示します。 <pre><filter><predicate>type = \${../refType}</predicate></filter></pre> XPath フィルターを定義する場合は、ソーステーブルとターゲットテーブルのフィールドのみを使用できます。つまり、リンクテーブルを介した関連付けである場合、XPath フィルターでリンクテーブルのフィールドを使用することはできません。 作成時のエラーメッセージ ：ユーザーインターフェイスで、ユーザーがフィルターに準拠していない新しい関連付けられたレコードを送信すると、レコードの作成がブロックされます。	いいえ

エレメント	説明	必須
	エラーメッセージは、エレメント <code>checkOnAssociatedRecordCreation/message</code> を使用してカスタマイズできます。ローカライズされた各メッセージバリエーションは、<code>xml:lang</code> 属性にロケールを持つネストされたメッセージエレメントで定義されます。サポートされていないロケールのデフォルトメッセージを指定するには、<code>xml:lang</code> 属性のないメッセージエレメントを定義します。このプロパティの詳細については、例を参照してください。 注意: 検証検索 XPath 関数は、関連付けフィルターでは禁止されています。	
<code>xpathFilter</code>	注意: 非推奨。このプロパティは、プロパティ <code>filter</code> に置き換えられました。 関連するオブジェクトをフィルタリングするための XPath 述語を定義します。	いいえ
<code>recordForm</code>	関連するレコードの形式をカスタマイズするための特定のコンポーネントを定義	いいえ

エレメント	説明	必須
	します。このコンポーネントは、 <code>UserServiceAssociationRecordFormFactory</code> を実装する JavaBean を使用して定義されます。 JavaBean は、属性 <code>osd:class</code> を使用して指定されます。次に例を示します。 <pre><recordForm osd:class="com.wombat.MyRecordFormFactory"/ ></pre>	

別のデータセットを参照することができます。そのためには、関連付けのタイプに応じて、エレメント `tableRefInverse`、`linkTable`、または `xpathLink` のいずれかで次のプロパティを定義する必要があります。

エレメント	説明	必須
<code>schemaLocation</code>	関連付けで使用されるフィールドを含むデータモデルを定義します。データモデルは、埋め込みデータモデルとモジュールにパッケージ化されたデータモデルを参照できる特定の URN を使用して定義されます。EBX® でサポートされている特定の URN の詳細については、<code>SchemaLocation</code> を参照してください。	はい
<code>dataSet</code>	関連付けで使用されるデータセットを定義します。このデータセットは、エレメント <code>schemaLocation</code> で指定さ	はい

エレメント	説明	必須
	れたデータモデルを使用する必要があります。	
<code>dataSpace</code>	関連付けで使用するデータセットを含むデータスペースを定義します。	いいえ

重要

- データセットを作成するときに、リポジトリにまだ存在しないコンテナへの関連付けを定義するデータセットを作成できます。ただし、このデータセットのコンテンツは、作成後すぐには利用できません。不在のコンテナが作成された後、データセットを使用可能にするためにデータモデルの更新が必要です。まだ存在しないコンテナを参照するデータセットを作成する場合、次の制限が適用されます。
 - データセットレベルで定義されたトリガーは実行されません。
 - テーブル外のフィールドのデフォルト値は初期化されません。
 - アーカイブのインポート中に、存在しないコンテナを参照するデータセットを作成することはできません。
- 別のデータセットへの関連付けの一貫性は、運用段階でチェックされます。関連付けが特定のデータセットのコンテキストで一貫していない場合、次のようになります。
 - 権限の観点からは、読み取り専用と見なされます。つまり、ユーザーインターフェイスでは関連付けは読み取り専用であり、データサービスは非表示になります。また、ユーザーインターフェイスでは、関連付けられたレコードを作成、削除、アタッチ、でタッチ、移動することはできません。ただし、関連付けられた既存のレコードを変更することは可能です。
 - コンテナデータセット、テーブル、またはレコードの検証から除外されます。

ユーザーインターフェイスの統合

`xs:annotation/appinfo` の下のエレメント `osd:defaultView/displayMode` を使用して、関連付けられたオブジェクトをフォームにレンダリングする方法を定義できます。

可能な値は次のとおりです。

- `inline` は、関連付けられたレコードがデータモデル内の関連付けの同じ位置にあるフォームでレンダリングされることを指定します。

- `tab` は、関連するレコードを特定のタブに表示することを指定します。
- `link` は、関連するレコードをモーダルウィンドウでレンダリングすることを指定します。

このプロパティが定義されていない場合、デフォルトでは、関連付けられたレコードは `inline` でレンダリングされます。

次の例では、関連付けられたオブジェクトを次の形式で `inline` でレンダリングするように指定しています。

```
<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>
```

次の例では、関連付けられたオブジェクトを特定のタブにレンダリングすることを指定しています。

```

<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>

```

ツールバーの使用

`xs:annotation/appinfo` の下のエレメント `osd:defaultView/toolbars` を使用して、ユーザーインターフェイスに表示するツールバーを定義することができます。ツールバーを使用すると、関連付けの表形式のビューを表示するときに表示するボタンとメニューをカスタマイズできます。

次の表は、`osd:defaultView/toolbars` で定義できるエレメントを示しています。

エレメント	説明	必須
<code>tabularViewTop</code>	この関連付けのデフォルトのテーブルビューで使用するツールバーを定義します。	いいえ

エレメント	説明	必須
<code>tabularViewRow</code>	この関連付けのデフォルトビューの各行に使用するツールバーを定義します。	いいえ

次の例は、カタログとその製品の間の以前の関連付けのツールバーを使用する方法を示しています。

```
<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>
```

```
</xs:annotation>

</xs:element>
```

注意:

関連付けのターゲットテーブルを含むデータモデルで定義されたツールバーのみを使用できます。つまり、関連付けのターゲットテーブルが別のデータモデルで定義されている場合、このデータモデルで定義されているツールバーのみを参照でき、関連付けを保持しているツールバーでは参照できません。

以下も参照してください。:

- ツールバー

関連するオブジェクトのカスタマイズされたビュー

特定の表形式のビューを指定して、ターゲットテーブルに表示する必要のあるフィールドを定義できます。表形式のビューが定義されていない場合、付与されたアクセス権限に従って、ユーザーが表示できるすべての列が表示されます。表形式のビューは、`xs:annotation/appinfo` の下のエレメント `osd:defaultView/tabularView` を使用して定義されます。

次の表は、`osd:defaultView/tabularView` で定義できるエレメントを示しています。

エレメント	説明	必須
<code>column</code>	表示するターゲットテーブルのフィールドを定義します。指定するパスは、ターゲットテーブルから絶対パスにする必要があります。既存のフィールドを参照する必要があります。表示するフィールドを指定するために、いくつかの <code>column</code> エレメントを定義できます。	いいえ

エレメント	説明	必須
sort	関連するオブジェクトの並べ替えに使用できるフィールドを定義します。いくつかの sort エレメントを定義して、関連するオブジェクトの並べ替えに使用できるフィールドを指定できます。エレメント nodePath は、関連するオブジェクトの並べ替えに使用できるフィールドのパスを定義します。エレメント isAscending は、ソート順が昇順 (true) か降順 (false) かを指定します。	いいえ

次の例は、カタログとその製品の間の以前の関連付けから表形式のビューを定義する方法を示しています。

```
<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>

```

ユーザーインターフェイスでのアクション

ユーザーインターフェイスでは、次のアクションを実行できます。

- **作成**：関連付けのターゲットテーブルにオブジェクトを直接作成できます。新しいオブジェクトが作成されると、そのオブジェクトは現在のレコードに自動的に関連付けられます。
- **複製**：関連付けのターゲットテーブル内のオブジェクトを複製できます。新しいオブジェクトが作成されると、そのオブジェクトは現在のレコードに自動的に関連付けられます。
- **関連付け**：既存のオブジェクトを現在のレコードに関連付けます。リンクテーブルを介した関連付けの場合、リンクテーブル内のレコードが自動的に作成され、現在のレコードと既存のオブジェクトの間のリンクが実体化されます。
- **移動**：選択したオブジェクトを現在のレコードとは異なるレコードに関連付けます。リンクテーブルを介した関連付けの場合、前のリンクレコードが自動的に削除され、リンクテーブルに新しいレコードが自動的に作成されて、選択したオブジェクトとその新しい親レコードの間のリンクが具体化されます。

- **削除**：関連付けのターゲットテーブルで選択された関連付けられたオブジェクトを削除します。
- **切断**：現在のレコードと選択された関連オブジェクトの間のセマンティックリンクを切断します。リンクテーブルを介した関連付けの場合、リンクテーブル内のレコードは自動的に削除され、現在のレコードと関連付けられたオブジェクトの間のリンクが解除されます。

注意:

アクション `associate` および `detach` は、**XPath 述語** (エレメント `xpathLink`) を使用して関連付けが定義されている場合は使用できません。

アクションのカスタマイズされたビュー

表形式または階層形式の公開ビューを指定して、ユーザーインターフェイスを介してアクションを実行するときにオブジェクトを表示する方法を定義できます。公開されたビューは、

`xs:annotation/appinfo` の下のエレメント `osd:defaultView/associationViews` を使用して定義されます。

次の表は、`osd:defaultView/associationViews` で定義できるエレメントを示しています。

エレメント	説明	必須
<code>viewForAssociateAction</code>	現在のレコードに関連付けるターゲットテーブルのオブジェクトを表示するときに使用する公開ビューを定義します。指定されたビューは、関連付けのターゲットテーブルで公開および作成する必要があります。	いいえ
<code>viewForMoveAction</code>	関連するオブジェクトを現在のテーブルの別のレコードに移動するときに使用する公開ビューを定義します。指定されたビューは、現在のテーブルで公開および作成する必要があります。	いいえ

次の例は、カタログとその製品の間の以前の関連付けからビューを定義する方法を示しています。

```
<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>
```

検証

一部のコントロールは、関連付けられたオブジェクトを制限するために、関連付けに対して定義できます。これらのコントロールは、エレメント `osd:association` の下で定義されます。

次の表は、`xs:annotation/applInfo/osd:association` で定義できるコントロールを示しています。

エレメント	説明	必須
<code>minOccurs</code>	この関連付けに必要な関連付けられたオブジェクトの最小数を指定します。この最小数は、エレメント <code>value</code> を使用して定義され、正の整数にする必要があります。	いいえ。デフォルトでは最小値は制限されていません。
<code>maxOccurs</code>	この関連付けに許可される関連付けられたオブジェクトの最大数を指定します。この最大数はエレメント <code>value</code> によって定義され、正の整数またはこの最大値が制限されていないことを示す生の文字列 <code>unbounded</code> のいずれかにする必要があります。関連するオブジェクトの最大数は、関連するオブジェクトの最小数よりも大きくする必要があります。	いいえ。デフォルトでは最大値は制限されていません。
<code>constraint</code>	関連するレコードを制限するための XPath 述語を定義します。XPath 述語を定義する場合は、ソーステーブルとターゲットテーブルのフィールドのみを使用できます。つまり、リンクテーブルを介した関連付けである場合、XPath 述語でリンクテーブルのフィールドを使用することはできません。関連するデータセットでは、関連するレコードが指定された制約に準	いいえ

エレメント	説明	必須
	拠していない場合、検証時に指定された重大度の検証メッセージが追加され、エンドユーザーに表示されます。	
validation	検証メッセージは、エレメント <code>validation</code> を使用して、エレメント <code>minOccurs</code>、<code>maxOccurs</code>、および <code>constraint</code> の下で定義できます。検証メッセージの重大度は、エレメント <code>severity</code> を使用して指定されます。考えられる重大度は、<code>error</code>、<code>warning</code>、<code>info</code> です。重大度が指定されていない場合、デフォルトでは、重大度 <code>error</code> が使用されます。ローカライズされた検証メッセージは、エレメント <code>message</code> を使用して指定できます。このメッセージは、関連付けがこの制約に準拠していない場合、検証時にエンドユーザーに表示されます。ローカライズされた各メッセージバリエーションは、<code>xml:lang</code> 属性にロケールを持つネストされたメッセージエレメントで定義されます。サポートされていないロケールのデフォルトメッセージを指定するには、<code>xml:lang</code> 属性のないメッセージエレメントを定義します。	いいえ

データサービスの統合

データサービスの **選択** 操作で、関連付けられたオブジェクトを非表示にする必要があるかどうかを定義できます。このために、`xs:annotation/xs:appinfo` の下のプロパティ `osd:defaultView/hiddenInDataServices` を関連付けに設定できます。プロパティを「true」に設定すると、データサービスの **選択** 操作で関連するオブジェクトが非表示になります。このプロパティが定義されていない場合、デフォルトでは、関連付けられたオブジェクトがデータサービスの **選択** 操作に表示されます。

以下も参照してください。:

- データサービスでのフィールドの非表示
- 関連付けフィールド

例

たとえば、**以前**に定義された製品カタログデータモデルは、製品がカタログに属することを指定します (「Products」テーブルの外部キーによって明示的に定義されます)。「Catalogs」テーブルに外部キーの逆である次の関連付けが含まれていない限り、逆の関係 (カタログに特定の製品がある) は XML スキーマで簡単に表現できません。

```
<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:element>
```

```

    </xs:annotation>

</xs:element>

```

リンクテーブルを介した関連付けの場合、前の例を検討して、いくつかの更新を行うことができます。たとえば、「Products」テーブルの外部キーが削除され、製品とカタログの関係が、2つの外部キーで構成される主キーを持つリンクテーブル（「Catalogs_Products」という名前）によって再定義されます。「Products」テーブル（「productRef」という名前）と「Catalogs」テーブル（「catalogRef」という名前）の別のテーブル。次の例は、この新しい関係からリンクテーブルを介して関連付けを定義する方法を示しています。

```

<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>

```

次の例は、別のデータセットの外部キーを参照する関連付けを示しています。この例では、「Products」テーブルと「Catalogs」テーブルは同じデータセットにありません。

```

<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>

```

次の例では、「Technology」タイプの製品のみを関連付ける XPath フィルターを定義しています。

```

<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>

```

```

        </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>

```

次の例では、カタログに必要な製品の最小数を指定しています。

```

<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>

```

次の例では、カタログに最大 10 個の製品を含める必要があることを指定しています。

```

<xs:element name="products" minOccurs="0" maxOccurs="10" 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>

```

リンクされたフィールド

概要

リンクされたフィールドは、メインテーブルのフィールドを既存の関係上の別のテーブルのいくつかのフィールドと集約することにより、マルチテーブルビューをシミュレートする機能を提供します。これにより、関係を使用して他のレコードから参照されるレコードの一部のフィールドをモデル駆動型で簡単に統合できます。

以下も参照してください。:

- `SchemeLinkedField`

次の関係を横断的に対応 (トラバース) できます。

- 外部キー制約
- 「外部キーによる」 関連付け

****重要：****これらの関係は「単一値の関係」にする必要があります。つまり、関係を定義するテーブルのレコードは、関係のターゲットテーブル内の1つのレコードのみを参照する必要があります。

詳細

- 外部キーフィールドで *maxOccurs* が 1 に等しいと定義されている場合、外部キーは単一値の関係と見なされます。
- 参照される外部キーフィールドがターゲットテーブルの一意の主キーフィールドである場合、またはこの外部キーフィールドに一意性制約が定義されている場合、外部キーによる関連付けは単一値の関係と見なされます。

注意:

リンクされたフィールドは、ターゲットフィールドからいくつかのプロパティ (外部キー制約、列挙、値のラベル付け、検索戦略、検索モード、フォーマット設定) を継承します。

宣言

minOccurs = 0 および *maxOccurs* = 0 のエレメントであるリンクフィールドは、次のアノテーションを追加することで宣言されます。

```
<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:element>
```

```

        </xs:annotation>

</xs:element>

```

エレメント *osd:function* で定義された属性 *linkedField* は、コンテナテーブルの単一値の関係を参照するステップと、関係のターゲットテーブルのフィールドへのステップで構成されるパスを定義します。

重要

- パスが絶対パス (「/」で始まる) の場合、コンテナテーブルノードから解決されます。
- パスが相対パスの場合、現在のフィールドから解決されます。

以前に定義された製品カタログデータモデルのコンテキストでは、フィールド *catalog_ref_label* が製品テーブルに追加されます。最初のステップ *catalog_ref* は、製品がカタログに属していることを示す外部キーを参照します。最後のステップ *catalogLabel* は、参照されたカタログのラベルを参照します。このフィールドは、外部キー制約のターゲットテーブルである *Catalog* テーブルに属しています。

制限事項

- 現在、リンクされたフィールドでは、単一値の関係の 1 つのレベルのみをターゲットにできません。結果は次のようになります。
 - 単一値の関係は、別のリンクされたフィールドで定義することはできません。
 - リンクされたフィールドは、別のリンクされたフィールドをターゲットにすることはできません。
- リンクされたフィールドは、プロパティ *minOccurs* および *maxOccurs* (=0) を定義する必要があります。
- リンクされたフィールドは、ターゲットフィールドの1つと互換性のあるデータ型を定義する必要があります。
- リンクされたフィールドをコンテナテーブルの主キーの一部にすることはできません。
- リンクされたフィールドは、端末フィールドをターゲットにする必要があります。
- リンクされたフィールドは、集約リストまたは集約リストの下のフィールドをターゲットにすることはできません。
- リンクされたフィールドは、リンクされたフィールドでもある単一値の関係を参照することはできません。

- リンクされたフィールドには制約を定義しないでください。リンクされたフィールドで制約が宣言されている場合、警告メッセージがスローされ、宣言された制約は無視されます。
- リンクされたフィールドの場合、プロパティ `osd:disableValidation` は常に `true` と見なされ、上書きできません。

制約

ファセットを使用すると、データモデルでデータ制約を定義できます。TIBCO EBX® は、XML スキーマ制約ファセットをサポートし、高度なデータ制御のための拡張されたプログラムファセットを提供します。

- XML スキーマでサポートされているファセット
 - 一意性の制約
- 拡張ファセット
 - 外部キー
 - 動的制約
 - FacetOResource 制約
 - 値の除外
 - 別のノードを使用して定義された列挙型制約
- プログラムファセット
 - プログラム上の制約
 - プログラム上の列挙の制約
 - 「null」値の制約
 - テーブルの制約
- 制御ポリシー
 - ブロッキングと非ブロッキングの制約
 - 「null」入力を確認
 - データ型の EBX® 空白管理
 - 空の文字列管理
 - 検証メッセージのしきい値

XML スキーマでサポートされているファセット

次の表は、さまざまなデータ型でサポートされているファセットを示しています。

キー：

- **X** - サポートされています。
- **1** - `whiteSpace` ファセットは定義できますが、EBX®では解釈されません。
- **2** - XML スキーマでは、タイプ `string` で境界ファセットは許可されていません。それにもかかわらず、EBX® は拡張機能などのファセットを許可します。
- **3** - `osd:resource` タイプは、必須のファセット `Facet0Resource` のみをサポートします。
[拡張ファセット](#)を参照してください。
- **4** - `osd:dataspaceKey`、`osd:datasetName`、および `osd:color` タイプはファセットをサポートしません。

これらの型では、[プログラム上の制約のみ](#)がサポートされています。

	LENGTH	MINLENGTH	MAX LENGTH	PATTERN	ENUMERATION	WHITE SPACE
<code>xs:string</code>	X	X	X	X	X	1
<code>xs:boolean</code>				X		1
<code>xs:decimal</code>				X	X	1
<code>xs:dateTime</code>				X	X	1
<code>xs:time</code>				X	X	1
<code>xs:date</code>				X	X	1
<code>xs:anyURI</code>	X	X	X	X	X	1
<code>xs:Name</code>	X	X	X	X	X	1
<code>xs:integer</code>				X	X	1
<code>osd:resource</code> 3						1

	LENGTH	MINLENGTH	MAX LENGTH	PATTERN	ENUMERATION WHITE SPACE
<code>osd:dataspaceKey</code> 4					1
<code>osd:datasetName</code> 4					1
<code>osd:color</code> 4					1

	FRACTION DIGITS	TOTAL DIGITS	MAX INCLUSIVE	MAX EXCLUSIVE	MIN INCLUSIVE	MIN EXCLUSIVE
<code>xs:string</code>			2	2	2	2
<code>xs:boolean</code>						
<code>xs:decimal</code> X	X	X	X	X	X	X
<code>xs:dateTime</code>			X	X	X	X
<code>xs:time</code>			X	X	X	X
<code>xs:date</code>			X	X	X	X
<code>xs:anyURI</code>						
<code>xs:Name</code>			2	2	2	2
<code>xs:integer</code> X	X	X	X	X	X	X
<code>osd:resource</code> 3						
<code>osd:dataspaceKey</code> 4						
<code>osd:datasetName</code> 4						

	FRACTION DIGITS	TOTAL DIGITS	MAX INCLUSIVE	MAX EXCLUSIVE	MIN INCLUSIVE	MIN EXCLUSIVE
osd:color 4						

例

```
<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>
```

一意性の制約

標準の XML スキーマエレメント `xs:unique` を使用して、一意性制約を定義することができます。この制約は、値または値のセットがテーブル内で一意である必要があることを示します。

一意性制約で定義された文字列フィールドのチェックに使用される大文字と小文字の区別を指定できます。大文字と小文字の区別に使用できる値は、「sensitive」または「insensitive」です。デフォルトでは、一意性制約では大文字と小文字が区別されます。

注意:

大文字と小文字を区別しないオプションを適用して大規模なデータを含むテーブルをチェックすると、パフォーマンスの問題が発生する可能性があります。

例

以下の例では、「publisher」テーブルのターゲットフィールド「name」に対して一意性制約が定義されています。これは、「publisher」テーブル内の2つのレコードが同じ名前を持つことができないことを意味します。大文字と小文字を区別するオプションは、フィールド「name」の一意性をチェックするために使用されます。

```
<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:unique>

</publisher>
```

```

        </osd:validation>

        </xs:appinfo>

    </xs:annotation>

    <xs:selector xpath="." />

    <xs:field xpath="name" />

</xs:unique>

</xs:element>

```

一意性制約はテーブル内で定義する必要があり、次のプロパティがあります。

プロパティ	説明	必須
<code>name</code> 属性	データモデルの制約を識別します。	はい
<code>xs:selector</code> エレメント	制限された XPath 式を使用して一意性制約が適用されるテーブルを示します (「..」は禁止されています)。また、(制約の意味を変更せずに) テーブル内のエレメントを示すこともできます。	はい
<code>xs:field</code> エレメント	制限された XPath 式を使用して、値が一意である必要があるコンテキスト内のフィールドを示します。複数の <code>xs:field</code> エレメントを定義することにより、値のセットが一意である必要があることを示すことができます。	はい

注意:

未定義の値 (null 値) は、単一のフィールドに適用される一意性制約では無視されます。複数のフィールドでは、未定義の値が考慮されます。つまり、値のセットが同じ定義済み値と未定義値を持っている場合、それらは重複していると見なされます。

追加のローカライズされた検証メッセージは、エレメント `annotation/appinfo` の下のエレメント `osd:validation` を使用して定義できます。カスタム検証メッセージが定義されていない場合は、ビルトインの検証メッセージが使用されます。

一意性制約は、単純な集約リストにも適用できます。この場合、リストの各値は、テーブルのスコープではなく、リストのスコープ内で一意にする必要があります。

例

以下の例では、「title」テーブルのターゲットフィールド「printedEditions」に対して一意性制約が定義されています。これは、エディションがリストに 1 回だけ表示されることを意味します。「title」フィールドの検索には、大文字と小文字を区別しないオプションが使用されます。

```
<xs:element name="title">

  ...

  <xs:complexType>

    <xs:sequence>

      ...

      <xs:element name="printedEditions" type="xs:string"
minOccurs="0" maxOccurs="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
référéncée 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>

```

制限事項

1. `xs:field` エレメントのターゲットはテーブル内にある必要があります。
2. 一意性制約は、計算フィールドには適用されません。
3. 一意性制約は、集約リストを含む複数のフィールドに適用することはできません。
4. 一意性制約は、埋め込みリストには適用できません。

以下も参照してください。:

- [Java API の一意性制約](#)

拡張ファセット

EBX® は、XML スキーマで指定されていないが、マスターデータの管理に役立つ追加の制約を提供します。

XML スキーマの適合性を保証するために、これらの拡張ファセットはエレメント `annotation/appinfo/otherFacets` の下で定義されます。

外部キー

EBX® を使用すると、特定のファセットを使用して既存のテーブルへの参照を作成できます。詳細については、[外部キー](#)を参照してください。

動的制約

動的制約ファセットは XML スキーマのセマンティクスを保持しますが、value 属性は、別のエレメントから値をフェッチできるようにする path 属性に置き換えられます。使用可能な動的制約は次のとおりです。

- `length`
- `minLength`
- `maxLength`
- `maxInclusive`
- `maxExclusive`
- `minInclusive`
- `minExclusive`

これらのファセットを使用して、データモデルを動的に変更できます。

例

```
<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>
```

```

        </osd:otherFacets>

        </xs:appinfo>

        </xs:annotation>

        ...

    </xs:element>

```

この例では、ファセット `minInclusive` の境界は静的に定義されていません。境界の値は、ノード `/domain/Loan/Pricing/AmountMini/amount` から取得されます。

制限事項

- ターゲットフィールドを集約リストにすることはできません。つまり、`maxOccurs` = 1を定義することはできません。
- ターゲットフィールドのデータ型は、ファセットと互換性がある必要があります。つまり、次のようにする必要があります。
 - ファセット `length`、`minLength`、および `maxLength` の `integer` 型。
 - ファセット `maxInclusive`、`maxExclusive`、`minInclusive`、および `minExclusive` のファセットを保持するフィールドのデータ型との互換性を保持する。
- ファセットを保持しているフィールドがテーブルにない場合、ターゲットフィールドをテーブルに含めることはできません。
- ファセットを保持するフィールドがテーブル内にある場合、ターゲットフィールドは同じテーブル内またはテーブル外にある必要があります。
- ターゲットフィールドが1つ以上の集約リストの下にある場合、ファセットを保持するフィールドもこれらの集約リストの下にある必要があります。つまり、ファセットを保持するフィールドは、ターゲットフィールドと同じリストオカレンス内、または親オカレンス内にある必要があります。これにより、XPath の観点から、ターゲットフィールドは単一の値を参照します。

FacetOResource 制約

このファセットは、タイプ `osd:resource` を使用してすべての定義に対して定義し、使用可能なパッケージ化されたリソースファイルのサブセットを列挙として指定する必要があります。このタイプの詳細については、`osd:resource` を参照してください。次の属性があります。

<code>moduleName</code>	エイリアスを使用して、リソースを含む EBX® モジュールを示します。リソースが現在のモジュールに含まれている場合、エイリアスの前には「wbp」を付ける必要があります。それ以外の場合、エイリアスは、ファイル <code>module.xml</code> のエレメント <code><dependencies></code> で定義されている値のいずれかにする必要があります。
<code>resourceType</code>	「Image」、「JavaScript」、「Style sheet」、「HTML」のいずれかの値であるリソースタイプを表します。
<code>relatedPath</code>	リソースが配置されるディレクトリを示します。このディレクトリは、リソースタイプに対応するディレクトリの下に配置する必要があります。たとえば、「Image」タイプのリソースの場合、ターゲットモジュールのディレクトリ WEB-INF/ と同じレベルにあるディレクトリ <code>www/common/images/</code> が使用され、相対パスは次の場所から定義する必要があります。さらに、リソースがローカライズされたディレクトリ (<code>www/fr/</code> など) で定義されている場合、同じ名前の別のリソースがディレクトリ <code>www/common/</code> で定義されている場合にのみ考慮されます。

このファセットの動作は列挙ファセットと同じです。値は、指定されたモジュールの指定されたリソースタイプディレクトリのローカルパスにあるすべてのファイルを再帰的に一覧表示することによって収集されます。

例

```
<xs:element name="promotion" type="osd:resource">
  <xs:annotation>
    <xs:appinfo>
```

```

        <osd:otherFacets>

            <osd:Facet0Resource osd:moduleName="wbp"

                osd:resourceType="ext-images"
osd:relativePath="promotion/" />

            </osd:otherFacets>

        </xs:appinfo>

    </xs:annotation>

</ xs:element>

```

EBX® モジュール (Jakarta EE Web アプリケーション) の標準ディレクトリ構造の概要については、[モジュール構造](#) を参照してください。

値の除外

excludeValue 制約

このファセットは、値が指定された除外値と同じでないことを確認します。

この例では、空の文字列が許可された値から除外されています。

例

```

<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 制約

このファセットは、値が値の範囲に含まれていないことを確認します。境界は除外されます。

例

この例では、20000 から 20999 までの値は許可されていません。

```

<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>

```

別のノードを使用して定義された列挙型制約

注意:

この種の制約は廃止されました。[外部キー制約](#)を使用する必要があります。これには制限があります。特に、ユーザーインターフェイスでのクイック検索および並べ替え操作では、制約によって定義されたラベルの代わりに、フィールドの未加工の値が使用されます。

デフォルトでは、列挙ファセットは XML スキーマで静的に記述されます。

列挙ファセットのコンテンツは、データモデル内の単純なエレメントのリストによって動的に提供することもできます。

例

この例では、列挙ファセットのコンテンツはノード `CountryList` から提供されています。

```

<xs:annotation>

    <xs:appinfo>

        <osd:otherFacets>

            <osd:enumeration osd:path="../../CountryList" />

        </osd:otherFacets>
    
```

```

    </xs:appinfo>

</xs:annotation>

```

参照されるノード `CountryList` には、次のことが適用されます。

- 集約リスト、つまり `maxOccurs > 1` にする必要があります。
- 列挙ファセットを持つノードと同じタイプのエレメントのリストにする必要があります。
- 列挙ファセットを持つノードがテーブル内にはない場合は、テーブル外のノードにする必要があります。
- この列挙型のノードがテーブル内にある場合は、テーブルの外部のノード、または列挙型ファセットのあるノードと同じテーブル内にある必要があります。
- ターゲットフィールドが1つ以上の集約リストの下にある場合、ファセットを保持するフィールドもこれらの集約リストの下にある必要があります。つまり、ファセットを保持するフィールドは、ターゲットフィールドと同じリストオカレンス内、または親オカレンス内にある必要があります。これにより、XPath の観点から、ターゲットフィールドは単一の値を参照します。

例

```

<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:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

        <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>

```

プログラムファセット

プログラムによる制約は、単純型の任意の XML エlement 宣言に追加できます。

XML スキーマの適合性を保証するために、プログラムによる制約が Element `annotation/`
`appinfo/otherFacets` の下に指定されています。

プログラム上の制約

プログラムによる制約は、インターフェイス `Constraint` を実装する Java クラスによって定義されます。

追加のパラメーターを定義できるため、実装される Java クラスは JavaBean プロトコルに準拠している必要があります。

例

以下の例では、Java クラスでメソッドを定義する必要があります。たとえば、`getParam1()`、`setParam1(String)`、`getParamX()`、`setParamX(String)` などです。

```
<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>
```

以下も参照してください。:

- [JavaBean 仕様](#)
- [JavaBeanVersion](#)

プログラム上の列挙の制約

列挙型制約は、値の順序付きリストを基本的なプログラム制約に追加します。このファセットを使用すると、リストから値を選択できます。これは、インターフェイス

[ConstraintEnumeration](#) を実装する Java クラスによって定義されます。

例

```
<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>
```

注意:

ユーザーインターフェイスでのクイック検索および並べ替え操作では、ラベルの代わりにフィールドの未加工の値が使用されます。この列挙を、同じ値のセットを定義するテーブルへの外部キー制約に置き換えることができるかどうかを検討してください。

「null」値の制約

場合によっては、たとえば、別のフィールドに特定の値がある場合など、いくつかの条件が満たされた場合にのみ値が必須になります。この場合、標準の XML スキーマ属性 `minOccurs` は静的であるため、不十分です。

値がそのコンテキストに従って必須であるかどうかを確認するには、次の要件が満たされている必要があります。

1. プログラムによる制約は、Java クラスで定義する必要があります (上記を参照)。
2. このクラスは、インターフェイス `ConstraintOnNull` を実装する必要があります。
3. XML スキーマのカーディナリティ属性は、エレメントがオプションであることを指定する必要があります (`minOccurs="0"` および `maxOccurs="1"`)。

注意:

デフォルトでは、「null」値の制約はユーザー入力時にチェックされません。入力でチェックを有効にするには、「`checkNullInput`」プロパティを設定する必要があります。また、エレメントがターミナルの場合は、データセットもアクティブ化する必要があります。

例

```
<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>

```

以下も参照してください。:

- [JavaBean 仕様](#)
- `JavaBeanVersion`

テーブルの制約

テーブルの制約は、インターフェイス `ConstraintOnTable` を実装する Java クラスによって定義されます。テーブルノードでのみ定義できます。

追加のパラメーターを定義することができます。実装された Java クラスは、JavaBean プロトコルに準拠している必要があります。

例

以下の例では、Java クラスでメソッドを定義する必要があります。たとえば、`getParam1()`、`setParam1(String)`、`getParamX()`、`setParamX(String)` などです。

```

<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>

```

注意:

パフォーマンス上の理由から、テーブルの制約は、データセットまたはテーブルの検証レポートを取得するときのみチェックされます。これは、レコードの挿入、削除、変更などの更新がテーブルで発生したときに、これらの制約がチェックされないことを意味します。ただし、依存関係が定義されている場合、内部の増分検証フレームワークはこれらの制約の検証コストを最適化します。詳細については、[データ検証](#)を参照してください。

以下も参照してください。:

- [JavaBean 仕様](#)
- `JavaBeanVersion`

制御ポリシー

ブロッキングと非ブロッキングの制約

リポジトリ内の更新が実行され、この更新によって特定の制約に従って検証エラーが追加された場合、新しいエラーが更新をブロックする (およびトランザクションをキャンセルする) か、非ブロックと見なされるか (トランザクションをキャンセルするか) を指定できます。更新をコミットして、後でエラーを修正できるようにします)。エレメント `osd:validation` 内のエレメント `blocksCommit` は、次のサポートされている値でこの指定を許可します。

<code>onInsertUpdateOrDelete</code>	操作 (データセットの更新、データセットの削除、レコードの作成、更新、または削除) の後も、制約が常に有効である必要があることを指定します。この場合、制約に違反する操作はすべて拒否され、値は変更されません。 これは、主キー制約、データ型変換制約 (整数または日付は適切に記述されている必要があります)、およびマップされたテーブルの構造制約のデフォルトの必須ポリシーです。
<code>onUserSubmit-checkModifiedValues</code>	ユーザーが関連付けられた値を変更してフォームを送信するたびに、制約が有効なままである必要があることを指定します。この場合、制約に違反するフォーム入力はすべて拒否され、値は変更されません。 これは、前のケースで説明したすべてのブロッキング制約のデフォルトポリシーです。たとえば、外部キー制約は、フォーム送信のコンテキストを除いて、デフォルトではブロックされません (他のレコードによって参照されているレコードは削除できる、など)。
<code>never</code>	制約が操作をブロックしてはならないことを指定します。この場合、制約に違反する操作

はすべて許可されます。ユーザーインターフェイスのコンテキストでは、ユーザーがこの制約に違反する値を設定した場合、この制約はフォームの送信をブロックしません。

外部キー制約では、すべての操作をブロックする制御ポリシーは、フィルター処理されたレコードには適用されません。つまり、参照レコードが存在するが外部キーフィルタを満たさない場合、外部キー制約はブロックされません。この場合、更新は拒否されず、検証エラーが発生します。

マップされたテーブルで定義されている構造制約に制御ポリシーを指定することはできません。つまり、このプロパティは、基になる RDBMS ブロッキング制約の検証ポリシーのため、固定長、最大長、最大桁数、および小数点以下の桁数の制約には使用できません。

このプロパティは、アーカイブのインポートやデータスペースのマージには適用されません。つまり、アーカイブをインポートしてデータスペースをマージするときは、構造上の制約を除くすべてのブロッキング制約が常に無効になります。

以下も参照してください。:

- 重大度のあるファセット検証メッセージ
- 外部キー

XML スキーマファセット

制御ポリシーは、ファセット定義の下にある `annotation/appinfo` のエレメント `osd:validation` によって記述されます。

例

```
<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 スキーマ列挙ファセット

制御ポリシーは、フィールド定義の下にある `annotation/appinfo` のエレメント `osd:enumerationValidation` によって記述されます。

例

```

<xs:element name="Gender">

    <xs:annotation>

        <xs:appinfo>

            <osd:enumerationValidation>

                <blocksCommit>onInsertUpdateOrDelete</blocksCommit>

            </osd:enumerationValidation>

        </xs:appinfo>

    </xs:annotation>

</xs:element>

```

```

        </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® ファセット

制御ポリシーは、ファセット定義 (`annotation/appinfo/otherFacets` で定義) の下のエレメント `osd:validation` によって記述されます。

値が `onInsertUpdateOrDelete` および `onUserSubmit-checkModifiedValues` の制御ポリシーは、`osd:excludeSegment`、`osd:excludeValue`、および `osd:tableRef` EBX® ファセットのみで利用できます。

値が `never` の制御ポリシーは、すべての EBX® ファセットで定義できます。

プログラムによる制約では、値が `never` の制御ポリシーは、対応する制約のセットアップ中にのみ直接設定できます。詳細については、Java API の

`ConstraintContext.setBlocksCommitToNever` および

`ConstraintContextOnTable.setBlocksCommitToNever` を参照してください。

例

```

<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>

```

「null」 入力を確認

EBX® のデフォルトの検証ポリシー従うと、一時的に不完全な入力を許可するために、必須エレメントはユーザー入力時に完了がチェックされません。代わりに、データセットの検証でのみ検証されます。ユーザー入力の直後に完了を確認する必要がある場合、エレメントで属性 `osd:checkNullInput="true"` を追加で指定する必要があります。集約リスト (`maxOccurs > 1`) で定義されている場合、このプロパティは無視されます。

注意:

データモデルが `minOccurs="1"` を使用して静的に、または「null」の制約を使用して動的に必須エレメントを指定する場合、値は必須です。ターミナルエレメントの場合、必須値はアクティブ化されたデータセットに対してのみチェックされます。ターミナル以外のエレメントの場合、データセットをアクティブ化する必要はありません。

例

```
<xs:element name="amount" osd:checkNullInput="true" minOccurs="1">
    ...
</xs:element>
```

以下も参照してください。:

- 「null」値の制約
- 空白管理
- 空の文字列管理

データ型の EBX® 空白管理

XML スキーマ (<https://www.w3.org/TR/xmlschema-2/#rf-whiteSpace> を参照) によると、空白の処理は *preserve*、*replace*、*collapse* のいずれかのプロシージャに従う必要があります。

preserve	正規化は実行されず、値は変更されません。
replace	<code>#x9</code> (タブ)、 <code>#xA</code> (改行)、および <code>#xD</code> (キャリッジリターン) のすべての出現箇所が <code>#x20</code> (スペース) に置き換えられます。)
collapse	replace プロシージャの後、連続する <code>#x20</code> は 1 つの <code>#x20</code> に折りたたまれ、先頭または末尾の <code>#x20</code> は削除されます。

一般的な空白処理

EBX® は、XML スキーマの推奨事項に準拠しています。

- タイプ `xs:string` のフィールドの場合、主キーエレメントであるかどうかに関係なく、空白は常に保持され、空の文字列が `null` に変換されることはありません。
- 他のフィールド (`xs:string` 以外のタイプ) の場合、空白は常に折りたたまれ、空の文字列は `null` に変換されます。

注意:**例外:**

- タイプ `osd:html` または `osd:password` のフィールドの場合、空白は常に保持され、空の文字列は `null` に変換されます。
- プロパティ `osd:checkNullInput="true"` を定義するタイプ `xs:string` のフィールドの場合、空の文字列は EBX® によるユーザー入力として `null` として解釈されます。

ユーザー入力時の空白処理

前のセクションで説明したルールはユーザーインターフェイスに適用されますが、先頭と末尾の空白はユーザー入力時に削除されます。つまり、ユーザーインターフェイスでは、空白はデフォルトで常にユーザー入力時にトリミングされます。その他の入力方法 (XML/CSV のインポート、データサービス、API 更新) は、ユーザーインターフェイスから削除されません。

注意:**例外:**

- タイプ `osd:password` のフィールドの場合、ユーザー入力時に空白は削除されません。
- 外部キーフィールドの場合、ユーザー入力時に空白は削除されません。

データモデルで、ユーザー入力時に空白を削除しないように指定することができます。属性 `osd:trim="disable"` は、ユーザー入力時に先頭と末尾の空白を許可するフィールドに設定できます。

例

```
<xs:element name="field" osd:trim="disable" type="xs:string">
```

```
...
```



```
</xs:element>
```

文字列型の主キーの空白処理

タイプ `xs:string` の主キー列の場合、デフォルトの EBX® 制約が定義されています。この制約は、レコードを作成するときに空の文字列と折りたたまれていない空白の値を禁止します。つまり、この制約に違反するレコードの作成はすべて拒否されます。

ただし、主キーノードが独自の `xs:pattern` ファセットを指定している場合、このファセットはデフォルトの EBX® 制約をオーバーライドします。たとえば、特定のパターン「`.*`」は任意の文字列を受け入れますが、これは推奨されません。

デフォルトの制約では、特定のあいまいさを処理できます。たとえば、ユーザーが「12 34」と「12 34」の文字列を区別するのは困難です。一般的な値の場合、これによって競合が発生することはありませんが、主キーでエラーが発生します。

以下も参照してください。:

- テーブルと関係

空の文字列管理

デフォルトの変換

タイプ `xs:string` のノードの場合、ユーザー入力時に空の文字列と `null` 値は区別されません。つまり、空の文字列値は、ユーザー入力時に自動的に `null` に変換されます。

空の文字列と「null」値の区別

空の文字列と `null` 値が区別される場合があります。たとえば、次の場合です。

- 主キーは、空の文字列を許可するパターンを定義します。
- エレメントは外部キー制約を定義し、ターゲットテーブルには空の文字列を許可するパターンを定義する単一の主キーがあります。
- エレメントは、空の文字列を含む静的列挙を定義します。
- エレメントは、前述のケースの 1 つを使用して、別のエレメントへの動的な列挙を定義します。

空の文字列と `null` 値が区別される場合、これは次の動作を意味します。

- 空の文字列は、ユーザー入力時に `null` に変換されません。
- タイプ `xs:string` のノードの入力フィールドには、ノードの値を `null` に設定するための追加のボタンが表示されます。
- 検証時に、空の文字列は `minOccurs="1"` プロパティに関して準拠した値と見なされます。

検証メッセージのしきい値

検証を実行するときに、制約ごとに許可される検証メッセージの最大数をデータモデルレベルで指定できます。

例

```
<xs:schema ...>

    ...

    <xs:annotation>

        <xs:appinfo>

            <osd:validation>

                <validationMessageThreshold>250</validationMessageThreshold>

            </osd:validation>

        </xs:appinfo>

    </xs:annotation>

    ...

</xs:schema>
```

しきい値は、データモデルおよび各データセット検証レポートで定義された制約ごとに考慮されます。制約がしきい値に達すると、制約の検証が停止され、しきい値に達したことを示すエラーメッセージが検証レポートに追加されます。

検証メッセージのしきい値は、データモデルで定義されていない場合、デフォルトで `1000` に設定されます。検証メッセージの数に制限はありません。また、指定された検証メッセージのしきい値は 100 以上にする必要があります。

以下も参照してください。:

- `ValidationReport.hasConstraintsWithTooManyMessages`

トリガーと関数

EBX® データモデルでは、トリガーと計算フィールドを定義できます。また、自動インクリメントフィールドも提供されます。

- 計算値
 - 検証の無効化
- トリガー
 - データセットのトリガー
 - テーブルのトリガー
- 自動インクリメントされた値

計算値

デフォルトでは、データは XML リポジトリに読み込まれ、保持されます。ただし、データは計算や外部データベース アクセス (RDBMS や中央システムなど) の結果である場合もあります。

EBX® を使用すると、現在のデータセットのコンテキストで他のデータを考慮に入れることができます。

これは、*計算ルール* を定義することによって可能になります。

計算ルールは、`osd:function` エレメントを使用してデータモデルで指定されます (以下の例を参照)。

- `class` 属性の値は、Java インターフェイス `ValueFunction` を実装する Java クラスの修飾名にする必要があります。

- 追加のパラメーターはデータモデルレベルで指定できます。その場合、Java Bean 規則が適用されます。

例

```
<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>
```

検証の無効化

関数の実行に時間がかかる場合、計算値の検証を無効にすると便利ことがあります。実際、関数が N レコードのテーブルにアタッチされている場合、このテーブルを検証するときに N 回呼び出されます。データモデルで指定されたプロパティ `osd:disableValidation= "true"` を使用すると、計算値の検証を無効にすることができます (以下の例を参照)。

例

```
<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>

```

トリガー

データセットまたはテーブルレコードは、作成、更新、削除などの操作が実行されたときに自動的に実行されるメソッドに関連付けることができます。

データモデルでは、これらのトリガーは、`osd:trigger` エlementを使用して、`annotation/appinfo` Elementの下に宣言する必要があります。

データセットのトリガー

データセットトリガーの場合、抽象クラス `InstanceTrigger` を拡張する Java クラスをElement `osd:trigger` 内に宣言する必要があります。

データセットトリガーの場合は、データモデルの `root` Elementのすぐ下に `annotation/appinfo/osd:trigger` タグを定義することをお勧めします。

例

```

<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>

```

テーブルのトリガー

テーブルレコードトリガーの定義では、抽象クラス `TableTrigger` を拡張する Java クラスを `osd:trigger` エレメント内に定義する必要があります。関連するテーブルまたはテーブルタイプを記述するエレメントのすぐ下に `annotation/appinfo/osd:trigger` エレメントを定義することをお勧めします。

例

テーブルエレメントの場合

```

<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>

```

テーブル型エレメントの場合

```

<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>

```

追加のパラメーターを定義できるため、実装された Java クラスは Java Bean プロトコルに準拠する必要があります。上記の例では、Java クラスはメソッド `getParam1()`、`setParam1(String)`、`getParamX()`、`setParamX(String)`などを定義する必要があります。

自動インクリメントされた値

自動インクリメント値を定義することができます。自動インクリメント値はテーブル内でのみ許可され、`xs:int` または `xs:integer` 型にする必要があります。

自動インクリメントは、エレメント `annotation/appinfo` の下のエレメント `osd:autoIncrement` を使用してデータモデルに指定されます。

例

```
<xs:element name="autoIncrementedValue" type="xs:int">

  <xs:annotation>

    <xs:appinfo>

      <osd:autoIncrement />

    </xs:appinfo>

  </xs:annotation>

</xs:element>
```

また、オプションのエレメントとして、`start` と `step` の 2 つがあります。

- `start` 属性は、この自動増分の最初の値を指定します。この属性が指定されていない場合は、デフォルトで値 `1` が設定されます。
- `step` 属性は、自動インクリメントによって生成される次の値のステップを指定します。この属性が指定されていない場合は、デフォルトで値 `1` が設定されます。

例

```
<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>
```



```

        </xs:appinfo>

        </xs:annotation>

    </xs:element>

```

`osd:autoIncrement` を指定するフィールドの動作は次のとおりです。

- 新しいレコードが挿入され、フィールド値が未定義の場合は、フィールド値の計算と割り当てが実行されます。
- プログラムによる挿入ですでに `null` 以外の値が指定されている場合は、割り当ては実行されません。たとえば、アーカイブインポートまたは XML インポートで値が指定されている場合は、その値が保持されます。

したがって、非表示モードまたは上書きモードでのレコード挿入では割り当ては実行されません。

- 新しく割り当てられた値は、可能な限り、リポジトリのスコープ内で一意です。より正確には、割り当ての一意性は、データモデルのすべてのデータセットに及び、すべてのデータスペースにも及びます。後者の場合、`osd:autoIncrement` がレコードの主キーの一部である場合、競合が発生しないことが十分に保証された状態で、データスペースを親にマージできます。

この原則には非常に具体的な制限があります。値を指定する大量更新トランザクションが、同じフィールドに値を割り当てるトランザクションと同時に実行されると、後者のトランザクションが最初のトランザクションによって設定される値を割り当てる可能性があります (異なるデータスペース間ではロックはありません)。

内部的には、自動インクリメント値はリポジトリの「自動インクリメント」テーブルに保存されます。ユーザーインターフェイスでは、管理者は領域からこの値にアクセスできます。このフィールドは自動的に更新され、リポジトリ内の任意のインスタンスまたはデータスペースで、関連付けられた `osd:autoIncrement` フィールドに設定された最大値を定義します。この値は、更新されるテーブルで見つかった最大値を考慮して計算されます。

場合によっては、たとえば、自動インクリメント範囲が異なる複数の環境 (開発、テスト、実稼働) を管理する必要があるときは、この「最大値」チェックを回避する必要があります。この特定の動作は、`disableMaxTableCheck` プロパティを使用して実装できます。自動インクリメント値に競合が発生する可能性があるため、絶対に必要な場合を除き、このプロパティを有効にすることは一般的に推奨されません。ただし、このプロパティは次の方法で設定できます。

- ローカルで、自動インクリメント宣言のパラメーターエレメントを設定 (`<disableMaxTableCheck>true</disableMaxTableCheck>`)。
- データモデル全体で、データモデル宣言のエレメント `xs:appinfo` に `<osd:autoIncrement disableMaxTableCheck="true"/>` を設定。
- EBX® メイン構成ファイルで、プロパティ `ebx.autoIncrement.disableMaxTableCheck=true` をグローバルに設定。

TIBCO EBX® メイン構成ファイルを参照してください。

注意:

このオプションがグローバルに有効になっている場合、たとえば XML または CSV からインポートすることによって、自動インクリメントテーブルにレコードを作成できるようになります。このオプションを選択しない場合、リポジトリの整合性を確保するために、自動インクリメントテーブルにレコードを作成することは禁止されます。

ラベルとメッセージ

TIBCO EBX® を使用すると、データモデルのカスタムラベルとエラーメッセージをインターフェイスに表示できます。

- ラベルおよび説明
 - 動的なラベルと説明
- 列挙ラベル
- 必須のエラーメッセージ (`osd:mandatoryErrorMessage`)
- 変換エラーメッセージ
- 重大度のあるファセット検証メッセージ
 - XML スキーマ ファセット (`osd:validation`)
 - XML スキーマ列挙ファセット (`osd:enumerationValidation`)
 - EBX® ファセット (`osd:validation`)

ラベルおよび説明

データモデルの各ノードにラベルと説明を追加できます。EBX® では、各ノードがそのラベルとともに表示されます。ラベルが定義されていない場合は、エレメントの名前が使用されます。

2 つの異なる表記法を使用できます。

完全	ラベルと説明は、それぞれ <code><osd:label></code> エlementと <code><osd:description></code> Elementによって定義されます。
単純	ラベルはテキスト コンテンツから抽出され、最初のピリオド ('.') で終わり、最大 60 文字になります。説明にはテキストの残りの部分が使用されます。

説明には、外部ドキュメントへの標準 HTML `href` または EBX® 内の Adaptation の別のノードへのリンクのいずれかのハイパーリンクが含まれる場合もあります。

- `href` 表記またはその他の HTML を使用する場合は、適切にエスケープする必要があります。
- EBX® リンク表記はエスケープされず、ターゲットのパスを指定する必要があります。次に例を示します。

```
<osd:link path="../../../misc1">Adaptation 内の別のノードへのリンク</osd:link>
```

例

```
<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>

```

ノードが名前付きタイプを指している場合、ノードのラベルが名前付きタイプのラベルに置き換えられます。同じメカニズムがノードの説明(エレメント `osd:description`)にも適用されます。

注意:

空白の管理に関しては、ノードのラベルは、表示されるときに常に *折りたたまれます*。つまり、連続する空白のシーケンスは1つの空白に折りたたまれ、先頭と末尾の空白は削除されます。ただし、説明では、空白は常に *保持* されます。

動的なラベルと説明

各ノードのローカライズされたラベルと説明を静的に定義する代わりに、データモデルのノードのラベルと説明をプログラムで決定する Java クラスを指定することもできます。クラスを定義するには、データモデルに、属性 `class` を持つエレメント `osd:documentation` を含めます。ネストされたパラメーターエレメントを使用して、JavaBean プロパティを渡すこともできます。

例

```
<xs:schema ...>

  <xs:annotation>

    <xs:appinfo>

      <osd:documentation class="com.foo.MySchemaDocumentation">

        <param1>...</param1>

        <param2>...</param2>

      </osd:documentation>

    </xs:appinfo>

  </xs:annotation>

  ...

</xs:schema ...>
```

プログラムによって提供されるラベルと説明は、個々のノードでローカルに定義されたものよりも優先されます。

以下も参照してください。:

- `SchemaDocumentation`

列挙ラベル

列挙では、属性 `osd:label` を使用して、各列挙エレメントに単純な非ローカライズされたラベルを追加できます。

注意:

列挙エレメントに定義されたラベルは、表示されるときに常に `折りたたまれます`。

例

```
<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>
```

標準の `xs:documentation` エレメントを使用して、ラベルを完全にローカライズすることもできます。ローカライズされていないラベルとローカライズされたラベルの両方が列挙エレメントに追加された場合、ラベルが定義されていないロケールではローカライズされていないラベルが表示されます。

例

```
<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>

```

必須エラーメッセージ (osd:mandatoryErrorMessage)

ノードが属性 `minOccurs="1"` を指定している場合 (デフォルトの動作)、ユーザーがフィールドを完了しないと、提供する必要があるエラーメッセージが表示されます。このエラーメッセージは、エレメント `osd:mandatoryErrorMessage` を使用して、各ノードに対して個別に定義できます。

例

```

<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>

```

必須のエラーメッセージはローカライズできます。

```

<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>

```

注意:

空白の管理に関しては、列挙ラベルは表示されるときに常に *折りたたまれて* 表示されます。

変換エラーメッセージ

定義済みの XML スキーマエレメントごとに、ユーザー入力の形式が正しくない場合に特定のエラーメッセージを定義できます。

例

```

<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>

```

重大度のあるファセット検証メッセージ

フィールドの値が制約に準拠していない場合に表示される検証メッセージでは、カスタムの重大度、デフォルトのローカライズされていないメッセージ、およびローカライズされたメッセージのバリエーションを定義できます。重大度が指定されていない場合、デフォルトのレベルは **エラー** です。ブロッキング制約には、重大度 **エラー** が 必要です。

XML スキーマファセット (osd:validation)

検証メッセージは、ファセットの定義の下にある `annotation/appinfo` のエレメント `osd:validation` によって記述されます。

例

```

<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 スキーマ列挙ファセット (osd:enumerationValidation)

検証メッセージは、フィールドの定義の下にある `annotation/appinfo` のエレメント `osd:enumerationValidation` によって記述されます。

例

```

<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® ファセット (osd:validation)

検証メッセージは、ファセットの定義 (`annotation/appinfo/otherFacets` で定義) の下のエレメント `osd:validation` によって記述されます。

例

```
<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>
```

追加のプロパティ

- デフォルト値
- アクセスプロパティ
- 情報
- デフォルトビュー
 - デフォルトビューでのフィールドまたはテーブルの非表示
 - ビュー内のグループとフィールドの非表示
 - 構造化検索ツールでのフィールドの非表示
 - データサービスでのフィールドの非表示
 - 外部キーコンボボックスセレクトのビュー定義
 - デフォルトウィジェットのカスタマイズ
 - REST データサービスのカスタマイズ
- 比較モード
- 最終変更ポリシーの適用
- カテゴリ
 - データフィルター
 - 定義済みカテゴリ
 - 制限事項

デフォルト値

データモデルでは、属性 `default` を使用して、フィールドのデフォルト値を指定できます。このプロパティは、フィールドに値が定義されていない場合にデフォルト値を割り当てるために使用されます。

デフォルト値は、作成時にユーザー入力フィールドに表示されます。つまり、新しいレコードを作成するとき、または集約リストに新しい発生を追加するときに、デフォルト値が表示されます。

例

この例では、エレメントはデフォルトの文字列値を指定します。

```
<xs:element name="fieldWithDefaultValue" type="xs:string"
  default="aDefaultValue" />
```

アクセスプロパティ

属性 `osd:access` はアクセスモード、つまり特定のデータモデルノードのデータの読み取りおよび/または書き込みが可能かどうかを定義します。この属性には、`RW`、`R-`、`CC`、または `--` のいずれかの値が必要です。

各 XML スキーマノードに対して、次の 3 種類の Adaptation が可能です。

1. Adaptation ターミナルノード

このノードは、TIBCO EBX® で関連する値とともに表示されます。

メソッド `Adaptation.get()` を使用してアクセスすると、Adaptation 検索アルゴリズムが使用されます。

2. Adaptation 非ターミナルノード

このノードは、Adaptation ターミナルノードでもある子ノードが 1 つある場合にのみ EBX® に表示される複合タイプです。それ自体に値はありません。

メソッド `Adaptation.get()` を使用してアクセスすると、`null` が返されます。

3. Adaptation 不可ノード

このノードは Adaptation ターミナルノードではなく、子 Adaptation ターミナルノードはありません。このノードは EBX® には表示されません。

メソッド `Adaptation.get()` を使用してアクセスすると、定義されている場合はノードのデフォルト値が返され、そうでない場合は `null` が返されます。

以下も参照してください。:

- `Adaptation`

アクセスモード	動作
<code>RW</code>	Adaptation ターミナルノード: EBX® で値を読み書きできます。

アクセスモード	動作
R-	<i>Adaptation</i> ターミナルノード：値は EBX® でのみ読み取ることができます。
CC	<i>Cut</i> : これは <i>Adaptation</i> ターミナルノードではなく、その子ノードも <i>Adaptation</i> ターミナルノードではありません。この「指示」は、 <i>access</i> 属性の値に関係なく、どの子ノードよりも優先されます。
--	ノードが単純型の場合、アダプテーションは可能ではありません。ノードが複合型の場合、<i>Adaptation</i> ターミナルノードではなく、子ノードを定義しません。 データモデルの ルート ノードでは、このアクセス モードを指定する必要があります。
デフォルト	<i>access</i> 属性が定義されていない場合 - ノードが計算値である場合、それは R- と見なされます。 - ノードが単純型でその値が計算されていない場合は、RW と見なされます。 - ノードが集約リストである場合、それはターミナル値であり、RW と見なされます。 - それ以外の場合、<i>Adaptation</i> ターミナルノードではなく、その子ノードについては何も定義されません。

例

この例では、エレメントは *Adaptation* のターミナルノードであるため適応可能です。


```
<xs:element name="proxyIpAddress" type="xs:string" osd:access="RW"/>
```

情報

エレメント `osd:information` を使用すると、追加情報を指定できます。

以下も参照してください。:

- `SchemaNode.getInformation`

例

```
<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>
```

デフォルトビュー

デフォルトビューでのフィールドまたはテーブルの非表示

EBX® では、エレメント `osd:defaultView/hidden` を使用することで、テーブルまたはテーブル内のフィールドをデフォルトで非表示にすることができます。このプロパティは、特定のアクセス権限を定義せずに、データセットのデフォルトビューからエレメントを非表示にするために使用されます。つまり、デフォルトで非表示になっているエレメントは、関連付けられているデータ

モデルの構造から生成された、表形式または階層形式のいずれのデフォルトフォームおよびビューにも表示されません。

注意:

- エlementがデータセットのデフォルトビューで非表示になるように設定されている場合、このフィールドに関連付けられているアクセス権限は評価されません。
- ビューを定義することで、データセットのデフォルトビューで非表示になっているフィールドを表示できます。この場合のみ、このフィールドに関連付けられたアクセス権限が評価され、フィールドが表示されるかどうか決定されます。
- データセットのデフォルトビュー (ナビゲーションペイン) で非表示になっているテーブルを表示することはできません。

例

この例では、Elementはデータセットのデフォルトビューで非表示になっています。

```
<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>
```

ビュー内のグループとフィールドの非表示

エレメント `osd:defaultView/hiddenInViews` を使用すると、フィールドまたはグループをテーブルのすべてのビューで非表示にすることができます。このプロパティは、特定のアクセス権限を定義せずに、データセットの表形式 (デフォルトの表形式ビューを含む) および階層ビューからエレメントを非表示にするために使用されます。つまり、非表示のエレメントは、関連付けられたデータモデルの構造から作成された表形式または階層形式のどのビューにも表示されません。ただし、ビュー内の非表示のエレメントはフォームに表示されます。

すべてのビューでエレメントを非表示にするかどうかを指定するには、`osd:defaultView/hiddenInViews="true|false"` エレメントを使用します。

このプロパティが `true` に設定されている場合、カスタムビューを作成するときにエレメントは選択できません。その結果、データセット内のテーブルのすべてのビューにエレメントは表示されません。

グループがビューで非表示に設定されている場合、このグループの下にネストされているすべてのフィールドは、テーブルのビューにそれぞれ表示されません。

構造化検索ツールでのフィールドの非表示

構造化検索ツールでエレメントを非表示にするかどうかを指定するには、エレメント `osd:defaultView/hiddenInSearch="true|false|textSearchOnly"` を使用します。

このプロパティが `true` に設定されている場合、データセットのテキスト検索ツールと入力検索ツールでフィールドを選択できなくなります。

このプロパティが `textSearchOnly` に設定されている場合、フィールドはデータセットのテキスト検索でのみ選択可能ではなく、入力された検索で選択可能になります。

注意:

グループが検索ツールで非表示に設定されているか、テキスト検索でのみ非表示に設定されている場合は、このグループの下にネストされているすべてのフィールドは、検索ツールでは表示されず、テキスト検索でも表示されません。

いずれの場合も、フィールドはクイック検索ツールで検索可能なままになります。特定の検索戦略を定義することで、クイック検索を含むすべての検索ツールからフィールドを除外できます。

例

```

<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>

```

この例では、エレメントはデータセットのテキストおよび入力検索ツールで非表示になっています。

```

<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>

```

この例では、エレメントはデータセットのテキスト検索ツールでのみ非表示になります。

データサービスでのフィールドの非表示

データ サービスでエレメントを非表示にするかどうかを指定するには、エレメント

`osd:defaultView/hiddenInDataServices` を使用します。詳細については、[データモデルからのフィールドの無効化](#) を参照してください。

注意:

- グループが非表示として構成されている場合、このグループの下にネストされているすべてのフィールドはデータ サービスによって非表示と見なされます。

例

```
<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>
```

この例では、エレメントはデータサービスの選択操作で非表示になります。

外部キーコンボボックスセレクトのビュー定義

よりスムーズな選択のために、ターゲットテーブルまたは外部キーの階層ビューを表示するために使用される公開ビューを指定できます。ビューが定義されている場合、セレクトは、この外部キーのコンボボックスのユーザーインターフェイスに表示されます。ビューの定義は、XML スキーマエレメント `osd:defaultView/widget/viewForAdvancedSelection` を使用して行うことができます。

注意:

- このプロパティは外部キーフィールドでのみ定義できます。
- 公開されたビューは、外部キーのターゲットテーブルに関連付けられている必要があります。
- 公開されたビューが存在しない場合は、外部キーフィールドで高度な選択は使用できません。

例

この例では、公開されたビューの名前は、詳細選択で外部キーのターゲットテーブルを表示するように定義されています。

```
<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>

```

詳細については、[コンボボックスセレクト](#)を参照してください。

デフォルトウィジェットのカスタマイズ

ウィジェットは、データモデルアシスタントを使用して定義できます。詳細については、[デフォルトビュー > ウィジェット](#)を参照してください。

REST データサービスのカスタマイズ

デフォルトのビュー構成は、[セッションチャンネル](#)を通じて REST データサービスで管理されます。

比較モード

属性 `osd:comparison` をターミナルノードエレメントに含めることができます。このモードは、比較中にエレメントの相違点を検出する方法を制御します。属性に指定できる値は次のとおりです。

default	エレメントは、データの比較中に表示されます。
ignored	レコードまたはデータセット内のコンテンツの 2 つのバージョンを比較しても、変更は検出されません。

マージ中、無視されたエレメントのデータ値は、コンテンツが更新されても、値が異なってもマージされません。新しいコンテンツの場合、無視されたエレメントの値はマージされます。

アーカイブのインポート中、コンテンツが更新されても無視されたエレメントの値はインポートされません。新しいコンテンツの場合、無視されたエレメントの値はインポートされます。

注意:

- 比較中にグループが無視されるように設定されている場合、このグループの下にネストされているすべてのフィールドも無視されます。
- ターミナルフィールドに属性 `osd:comparison` が含まれていない場合、比較中にデフォルトで含まれます。

制限事項

- このプロパティは非ターミナルフィールドでは定義できません。
- 比較中に主キーフィールドを無視することはできません。

例

この例では、比較中に最初のエレメントは明示的に無視され、2 番目のエレメントは明示的に含まれます。

```
<xs:element name="fieldExplicitlyIgnoredInComparison"
  type="xs:string" minOccurs="0" osd:comparison="ignored"/>

<xs:element name="fieldExplicitlyNotIgnoredInComparison"
  type="xs:string" minOccurs="0" osd:comparison="default"/>
```


最終変更ポリシーの適用

属性 `osd:applyLastModification` を端末ノードエレメントに定義して、データセットのテーブルで実行できる「最終変更の適用」サービスにこのエレメントを含める必要があるかどうかを指定できます。

属性に指定できる値は次のとおりです。

default	このエレメントに最後の変更を適用できます。
ignored	このエレメントは、最終変更の適用サービスでは無視されます。つまり、このエレメントに対して実行された最終変更は、他のレコードには適用できません。

注意:

- グループが「最終変更を適用」サービスによって無視されるように設定されている場合、このグループの下にネストされているすべてのフィールドも無視されます。
- ターミナルフィールドに属性 `osd:applyLastModification` が含まれていない場合、最終変更の Adaptation サービスにデフォルトで含まれます。

制限事項

- このプロパティは非ターミナルフィールドでは定義できません。

例

この例では、最初のエレメントは「最後の変更を適用」サービスで明示的に無視され、2 番目のエレメントは明示的に含められます。

```
<xs:element name="fieldExplicitlyIgnoredInApplyLastModification"
  type="xs:string" minOccurs="0" osd:applyLastModification="ignored"/>

<xs:element name="fieldExplicitlyNotIgnoredApplyLastModification"
```

```
type="xs:string" minOccurs="0" osd:applyLastModification="default"/>
```

カテゴリ

カテゴリは、データモデルエレメントの表示を制限することで、「フィルタリング」に使用できます。

カテゴリを作成するには、データモデル XSD のテーブルノードに属性 `osd:category` を追加します。

データフィルター

以下の例では、*mycategory* という名前のカテゴリを作成するために、属性 `osd:category` がノードに追加されています。

```
<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>
```

ユーザーインターフェイスでデータセットに対して定義済みのカテゴリフィルターをアクティブにするには、ナビゲーションペインから > > **<category name>** を選択します。

定義済みカテゴリ

ローカライズされたラベルを持つ 2 つのカテゴリが事前定義されています。

- 非表示

テーブルノード自体を含むインスタンスノードは、デフォルトビューでは非表示になっていますが、ナビゲーションペインから >> を選択すると表示できます。

テーブルレコードノードは常に非表示になります。

- 制約 (非推奨)

制限事項

カテゴリは、「非表示」カテゴリを除き、テーブルレコードノードには適用されません。

データサービス

この章では、テーブルに関連する WSDL 操作の名前が TIBCO EBX® によってどのように定義および管理されるかについて詳しく説明します。

- 定義
- 構成
 - 埋め込みデータモデル
 - パッケージ化されたデータモデル
- パブリケーション
- WSDL とテーブル操作
 - WSDL ジェネレーター
 - SOAP 操作
- 制限事項

定義

EBX® は、[W3C Web Services Description Language 1.1](#) 標準に準拠する WSDL を生成します。デフォルトでは、WSDL 操作は、テーブルパスの最後のエレメントを使用してテーブルを参照します。WSDL 操作名は、アクション名 (プレフィックス) とテーブル名 (サフィックス) で構成されます。エンティティ名を使用して WSDL 操作でテーブルを参照することができます。エンティティ名では、テーブルのパスの最後のエレメントの代わりに一意の名前を使用できます。エンティティ名は、サフィックス操作の名前をオーバーライドします。

以下も参照してください。:

- Data Model Assistant を使用したエンティティマッピング

構成

埋め込みデータモデル

WSDL サフィックス操作の名前として使用されるエンティティ名は、EBX® のリポジトリに埋め込まれ、パブリケーションにリンクされます。つまり、埋め込みデータモデルを公開する場合、エンティティ名のリストをデータモデル定義の > テーブルの下で定義し、EBX® で管理できます。

パッケージ化されたデータモデル

WSDL サフィックス操作の名前として使用されるエンティティ名は、専用の XML ドキュメントファイルで定義され、データモデルとして指定され、キーワード `_entities` で終わる必要があります。たとえば、データモデルが `catalog.xsd` という名前の場合、オーバーライドされた WSDL 操作の名前の構成を含む XML ドキュメントは、`catalog_entities.xml` という名前になります。この XML ドキュメントも、データモデルと同じ場所に配置する必要があります。

XML ドキュメントは、データモデルのコンパイル時にこのパターンに一致するファイルが見つかった場合、EBX® によって自動的に読み込まれます。

パブリケーション

エンティティ名はコンパイル時に検証され、一意のテーブル名を持つ `Path` を含むカップルのリストが含まれます。チェックされる検証ルールは次のとおりです。

- `パス` は一意ではありません。
- テーブル名に構文エラーが含まれています。
- テーブル名が XML ドキュメント内で一意ではありません。

WSDL とテーブル操作

WSDL ジェネレーター

追加の検証ルールが追加されました。単一性チェックがテーブル名に体系的に適用されます。SOAP 操作名は、プレフィックスとしての操作タイプと、デフォルトではサフィックスとしてのテーブル名 (テーブルパスの最後のステップ) で構成されます。データセットには、パスが異なる複数の同一のテーブル名を含めることができます。単一性を保証するために、一意ではないテーブル名をオーバーライドすることができます。

SOAP 操作

テーブルに対する操作要求が SOAP コネクタから呼び出された場合、ターゲットテーブルは優先度によって取得されます。名前は次のものに対応します。

1. オーバーライドされたテーブル名
2. テーブルパスの最後のステップ

以下も参照してください。:

- データサービス

制限事項

WSDL 操作の名前は、**外部** データモデルでは使用できません。

ツールバー

この章では、TIBCO EBX® によってツールバーがどのように定義および管理されるかについて詳しく説明します。

- 定義
- ツールバーの使用

定義

ツールバーを使用すると、テーブルビュー、階層ビュー、またはレコードフォームにアクセスするときに表示するボタンとメニューをカスタマイズできます。

ツールバーは、データモデルアシスタントを使用してのみ作成および公開でき、**埋め込み** および **パッケージ** データモデルでのみ使用できます。

埋め込みデータモデルの場合、ツールバーは EBX® のリポジトリに埋め込まれ、パブリケーションにリンクされます。つまり、埋め込みデータモデルを公開する場合、データモデルで定義されたツールバーは、データモデルのパブリケーションに埋め込まれ、EBX® によって管理されます。

パッケージ化されたデータモデルの場合、ツールバーは専用の **XML** ドキュメントで定義され、データモデルとして名前を付け、キーワード **_toolbars** で終了する必要があります。たとえば、データモデルの名前が **catalog.xsd** の場合、ツールバーの定義を含む XML ドキュメントの名前は **catalog_toolbars.xml** にする必要があります。この **XML** ドキュメントも、データ

モデルと同じ場所に配置する必要があります。データモデルのコンパイル時にこのパターンに準拠するファイルが見つかった場合、ツールボードキュメントは EBX® によって自動的にロードされます。

以下も参照してください。:

- データモデルアシスタントを使用したツールバーの構成
- データモデルでのツールバーの使用
- ツールバー API

ツールバーの使用

ツールバーは、**テーブル** および **関連付け** で使用できます。

テーブルでは、ツールバーを表示する位置を、次のように指定できます。

- 表形式のビューの上部
- 表形式のビューの各行
- レコードフォームの上部
- 階層ビューの上部
- タイル ビューの上部
- タイルビューの各カード

関連付けでは、ツールバーを表示する位置を、次のように指定できます。

- 関連付けの表形式のビューの上部
- 関連付けの表形式のビューの各行

以下も参照してください。:

- ツールバーの使用
- 関連付け

カスタムフォーム

この章では、TIBCO EBX® によってカスタムフォームが定義および管理される方法について詳しく説明します。

関連する概念:

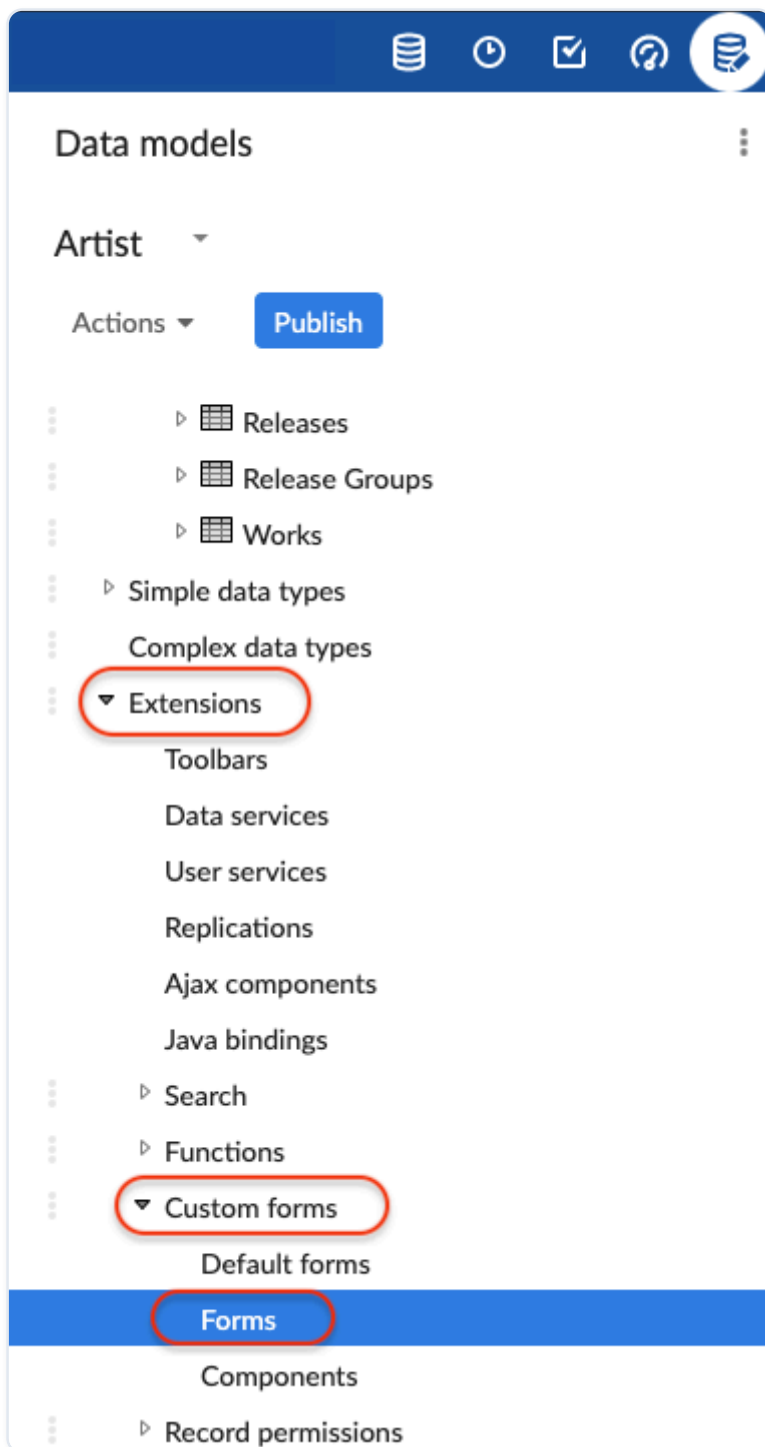
- インターフェイスのカスタマイズ
- アクセス
- フォームとコンポーネント
- エディター
- ブロック
 - 算術演算子
 - 配列
 - 配列項目
 - ブール演算子
 - チェックボックス
 - チェックボックスグループ
 - 子
 - 選択
 - 閉じるボタン
 - カラーピッカー
 - コンボボックス
 - コンパレーター
 - 比較
 - 配列の連結
 - 定数
 - コンテンツのタイトル
 - レコードの作成
 - カスタムアクションバー

- 日付時刻入力
- 宣言
- デフォルトのアクションバー
- デフォルトのツールバー
- レコードの重複
- 展開/折りたたみ
- フロー
- For each
- フォーム
- フォームの定義
- フォームグループ
- フォーム行
- フォームパラメーター
- From...get
- 関数
- 関数呼び出し
- 関数呼び出し引数
- 関数パラメーター
- グリッド
- グリッドエレメント
- If
- 入力パラメーター
- ロール
- ラベル
- ローカライズ済み
- メッセージ
- 次へボタン
- ノード
- 命名項目
- Not

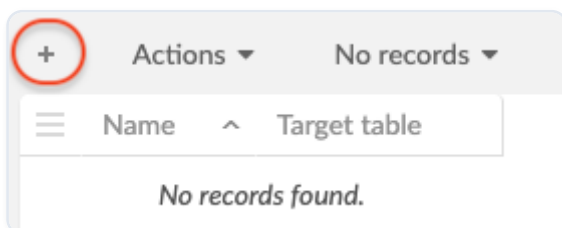
- オブジェクト
- それ以外の場合
- 段落タイトル
- パスワード
- 戻るボタン
- プロパティ
- ラジオボタン
- ラジオボタングループ
- 元に戻るボタン
- 保存ボタン
- 保存して閉じるボタン
- タブ
- タブコンテナ
- テキスト入力
- ツールバーの名前
- 変数
- When
- ウィジェット
- With

アクセス

アクセスするには、>>> に移動します。



ボタンをクリックしてフォームを作成します。



新しいフォームに名前を付け、そのフォームが使用できるテーブルを指定します。

Name AlbumForm

Target table * /root/artist

ボタンを押すと、レイアウトデザイナーにリダイレクトされます。

フォームとコンポーネント

フォームとコンポーネントの両方を データモデル拡張機能で作成できます。

- **フォームはレコードのレイアウトを記述します。** レコードや入力パラメーターなどのコンテキスト情報にアクセスできます。フォームを作成する際、ユーザーは対象テーブルを指定するよう求められます。作成されたら、拡張機能の テーブルまたは対象テーブルの「拡張機能」タブで、**テーブルのデフォルトフォームとして宣言**できます。テーブルのデフォルトではないフォームは使用されません。

- コンポーネントは、フォームと他のコンポーネント間で共有できる再利用可能なフラグメントです。フォームとは異なり、コンポーネントはコンテキスト情報にアクセスできません。そのような情報が必要な場合は、呼び出し元によって明示的に提供する必要があります。

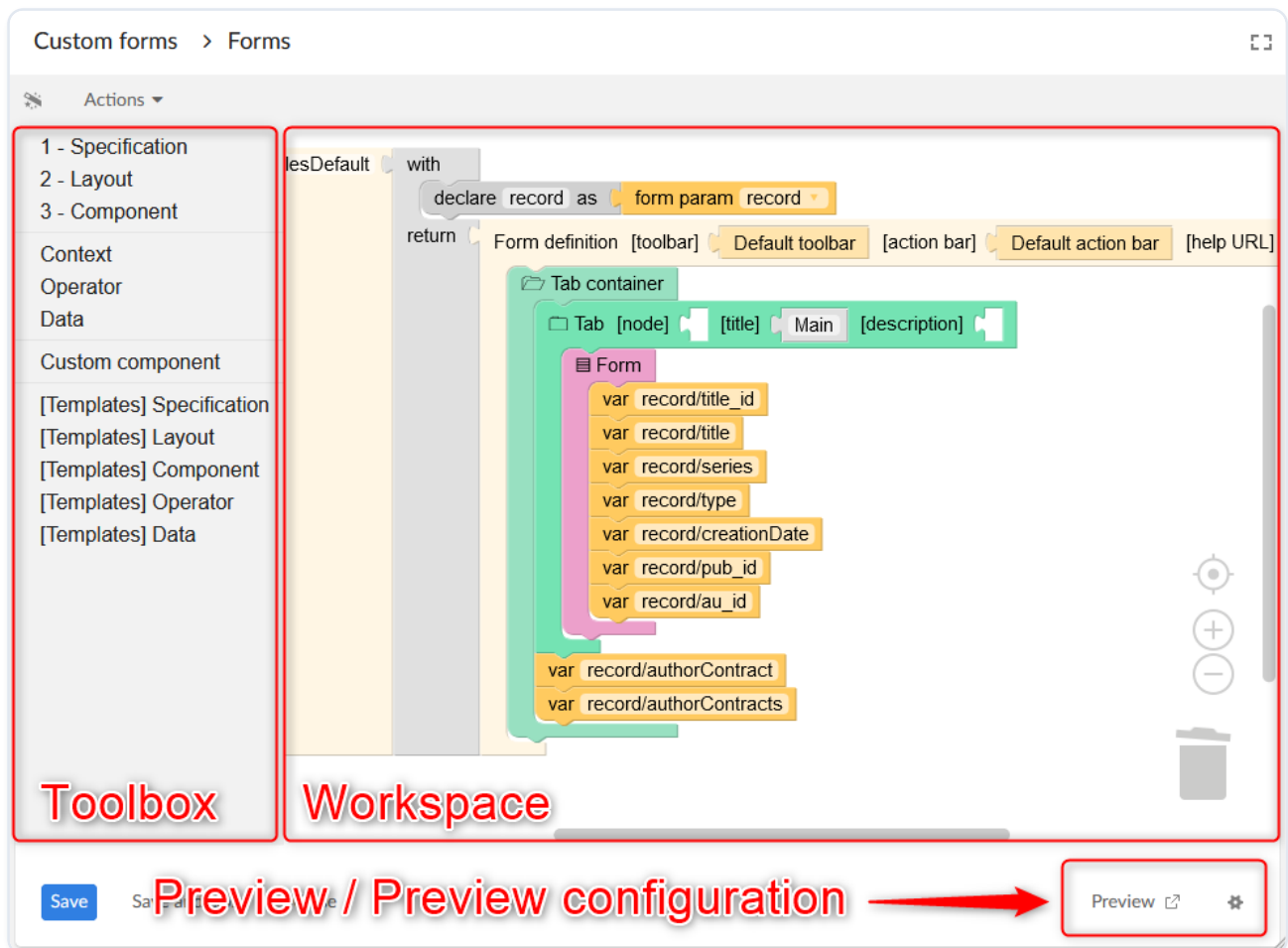
エディター

フォームまたはコンポーネントは、ブロックの重なり合いにすぎません。ブロックは具体的なグラフィックエレメントまたはロジックの一部を表すことができ、権限や言語などの条件に基づいて異なるレイアウトを設定できます。

ワークスペースには、定義済みのルートブロック内にフォームの説明が含まれています。このルートに接続されていないブロックは灰色表示され、非アクティブとしてマークされます。**ブロックを移動するには**、ブロックをドラッグアンドドロップする必要があります。ブロックをドラッグすると、その下に接続されているブロックもドラッグされます。移動する必要があるブロックが1つだけの場合は、そのブロックをクリックする前に Ctrl キーを押したままにします。ワークスペース内のブロックを **右クリック** すると、展開/折りたたみ、コメント、ヘルプなどのオプションの一覧も表示されます。

左側のツールボックスには、カテゴリのリストが表示されます。カテゴリをクリックすると、そのカテゴリに含まれるブロックが表示されます。一部のカテゴリには、関連するセクションがあります。このセクションでは、セクションのブロックのいくつかの組み合わせが提供されており、便利なショートカットやサンプルとして使用できます。

画面の右下には、**ボタン**と**ボタン**があります。これらを使用して、フォームがどのように表示されるかを確認できます。



ブロック

ツールボックスに表示されるすべてのブロックのリストを次に示します。

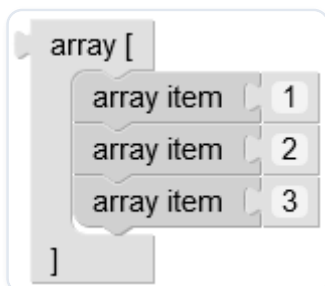
算術演算子

2つの整数に対する演算。



配列

項目のリスト。



以下も参照してください。:

- 配列項目

配列項目

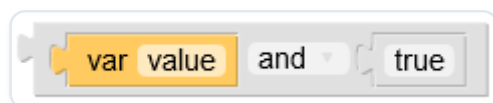
式ブロックを配列にするアダプタ。

以下も参照してください。:

- 配列

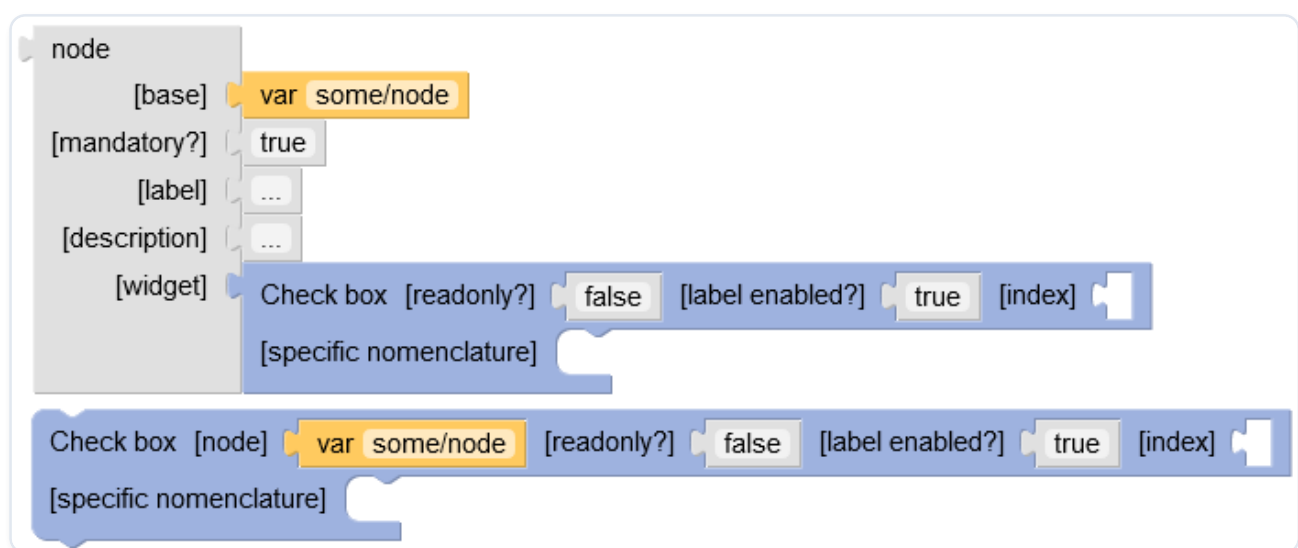
ブール演算子

2つのブール値に対する演算。



チェックボックス

チェックボックスウィジェットを表示します。

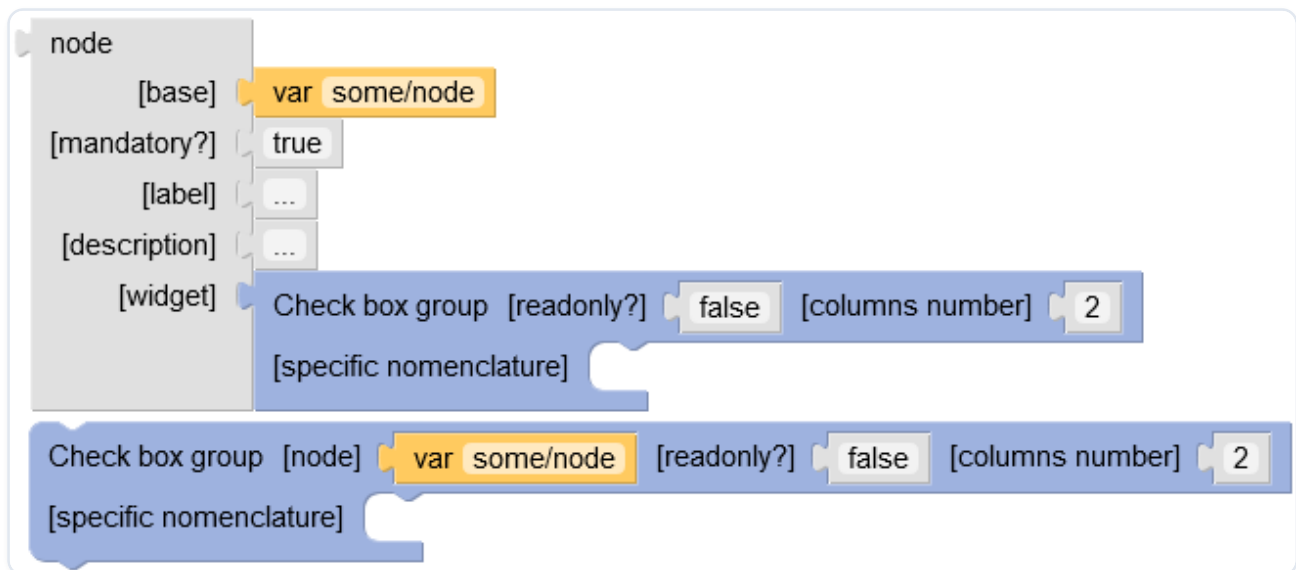


パラメーター

入力	説明
ノード	ウィジェットを表示するノード。
読み取り専用	ウィジェットを読み取り専用モードにするかどうかを示します。
ラベル有効	設定されている場合、ウィジェットの横に項目ラベルを追加するかどうかを示します。
インデックス	この列挙項目のインデックス。
特定の命名法	設定されている場合、モデル駆動型の命名法が上書きされます。

チェックボックスグループ

チェックボックスグループウィジェットを表示します。



パラメーター

入力	説明
ノード	ウィジェットを表示するノード。
読み取り専用	ウィジェットを読み取り専用モードにするかどうかを示します。

入力	説明
列数	設定されている場合、チェックボックスのレイアウトに使用する列の数を定義します。
特定の命名法	設定されている場合、モデル駆動型の命名法が上書きされます。

子

指定されたノードの子のリストを返します。

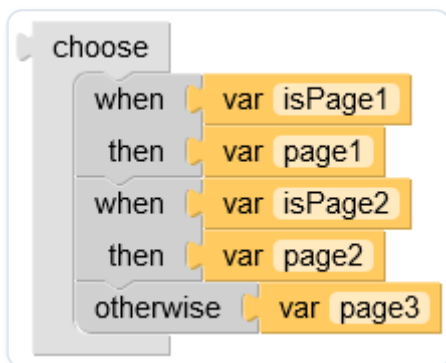


パラメーター

入力	説明
ノード	子を抽出する複合ノード。

選択

内部の「when」 / 「otherwise」ブロックのいずれかの内容を返すブロック。



パラメーター

入力	説明
content	「when」ブロックのリストと、オプションで最後の「otherwise」ブロック。

以下も参照してください。 :

- When
- それ以外の場合

閉じるボタン

標準の ボタン。

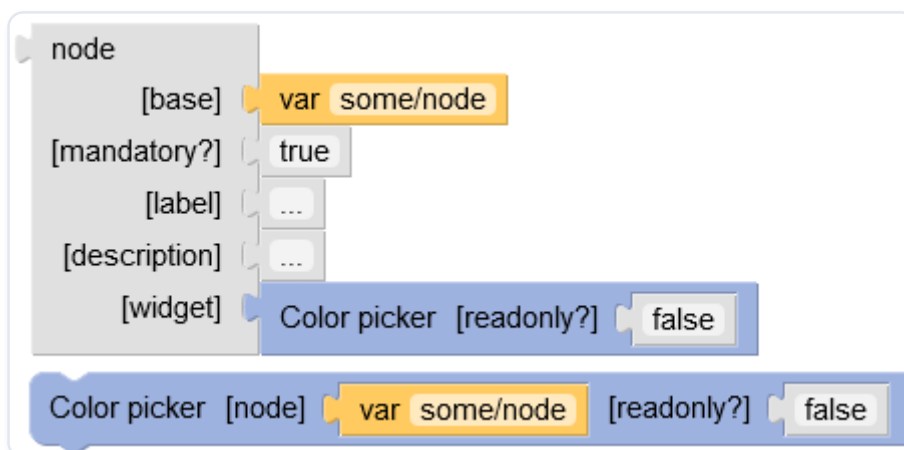
Close button [label] [default?] false

パラメーター

入力	説明
ラベル	設定されている場合、デフォルトのラベルが上書きされます。
デフォルト	これが Enter キーを押したときにトリガーされるアクションであるかどうかを示します。デフォルトのボタンは 1 つだけにする必要があります。

カラーピッカー

カラーピッカーウィジェットを表示します。

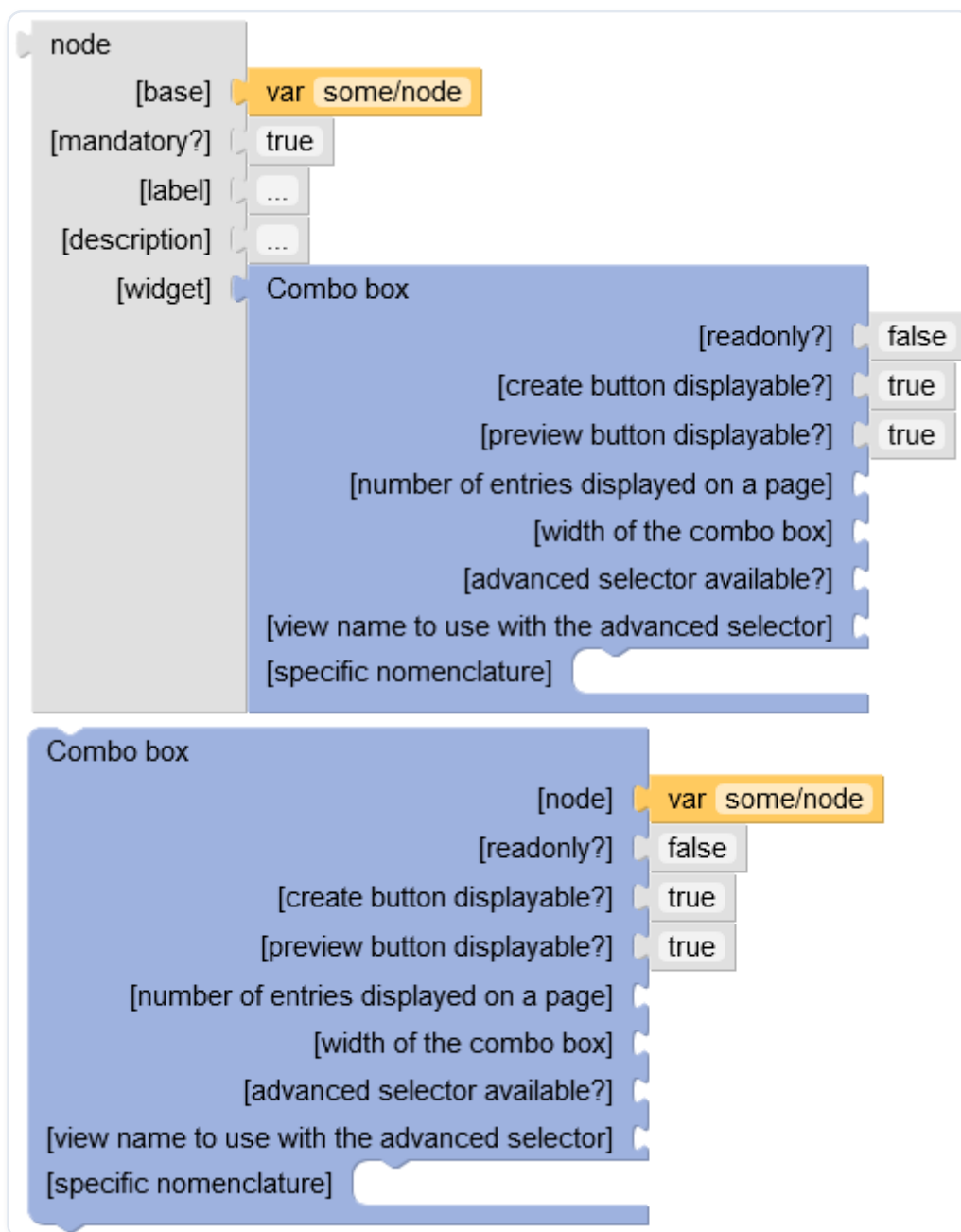


パラメーター

入力	説明
ノード	ウィジェットを表示するノード。
読み取り専用	ウィジェットを読み取り専用モードにするかどうかを示します。

コンボボックス

コンボボックスウィジェットを表示します。

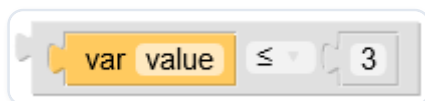


パラメーター

入力	説明
ノード	ウィジェットを表示するノード。
読み取り専用	ウィジェットを読み取り専用モードにするかどうかを示します。
表示可能なボタンの作成	設定されている場合、基になるノードが外部キーである場合に作成ボタンを表示するかどうかを定義します。
プレビューボタンを表示可能	設定されている場合、基になるノードが外部キーである場合にプレビューボタンを表示するかどうかを定義します。
ページに表示されるエントリの数	設定されている場合、ドロップダウン リストの各ページのエントリの数进行定義します。
コンボボックスの幅	設定されている場合、コンボボックスの幅を定義します。
高度なセレクトアーの利用	設定されている場合、基になるノードが外部キーである場合に高度なセレクトアーを表示するかどうかを定義します。
高度なセレクトアーで使用するビュー名	設定されている場合、関連付けられている外部キーフィールドのコンボ ボックス選択で使用する公開ビューの名前を定義します。
特定の命名法	設定されている場合、モデル駆動型の命名法が上書きされます。

コンパレーター

2 つの整数を比較します。



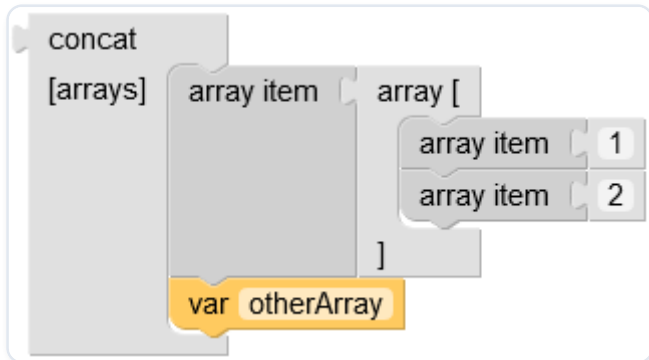
比較

文字列、数値、またはブール値を比較します。



配列の連結

指定された配列の連結を返します。

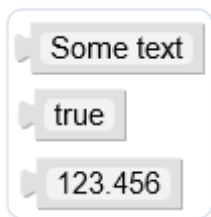


パラメーター

入力	説明
配列	配列のリスト。

定数

定数値。コンテキストに応じて、この値はテキスト、ブール値、または数値として解釈される場合があります。



パラメーター

入力	説明
値	テキスト、ブール値、または数値。

コンテンツのタイトル

ワークスペースコンテンツのタイトルを表示します。

Content title [label] Title to display [horizontal alignment] start

パラメーター

入力	説明
ラベル	表示するテキスト。
縦方向の配置	エレメントの水平方向の配置。設定されている場合、受け入れられる値は「start」、 「end」、「center」です。

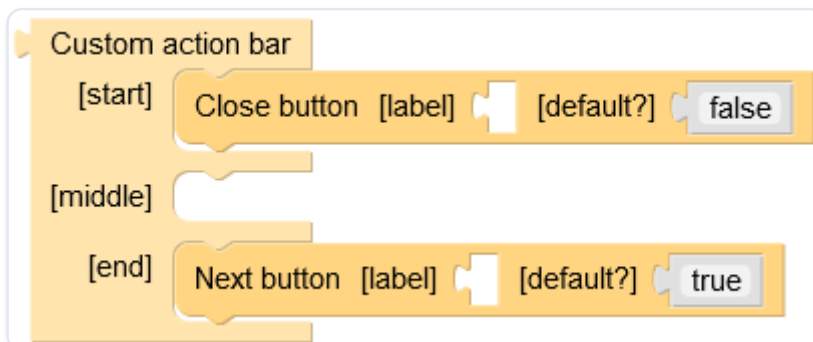
レコードの作成

フォームがレコードの作成または複製のコンテキストで表示されるかどうかを示すブール値を返します。

creating record?

カスタムアクションバー

カスタムアクションバー。



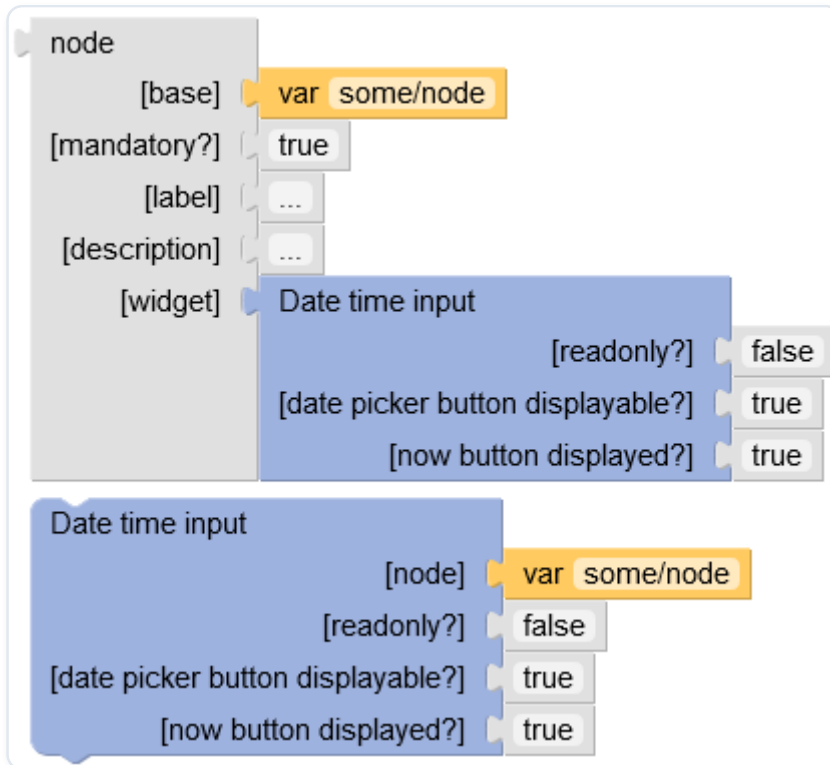
パラメーター

入力	説明
start	アクションバーの左側に表示される「ボタン」のリスト。
middle	アクションバーの中央に表示する「ボタン」のリスト。

入力	説明
end	アクションバーの右側に表示される「ボタン」のリスト。

日付時刻入力

入力された日付/時刻を表示します。



パラメーター

入力	説明
ノード	ウィジェットを表示するノード。
読み取り専用	ウィジェットを読み取り専用モードにするかどうかを示します。
日付選択ボタンの表示	日付ピッカーを開くボタンが表示されているかどうかを示します (読み取り/書き込みモードのみ)。
ボタンの表示	日付を現在の時刻に設定するボタンが表示されるかどうかを示します (読み取り/書き込みモードのみ)。

宣言

「with」ブロック内での変数の宣言。

```
declare isAllowed as in role? [administrator]
```

パラメーター

入力	説明
name	変数の名前。特定の 'with' ブロックに対して一意にする必要があります。
content	これを参照する 'var' ブロックによって返される変数の値。

以下も参照してください。:

- [With](#)

デフォルトのアクションバー

デフォルトのアクションバー。

```
Default action bar
```

デフォルトのツールバー

モデル駆動型ツールバー。

```
Default toolbar
```

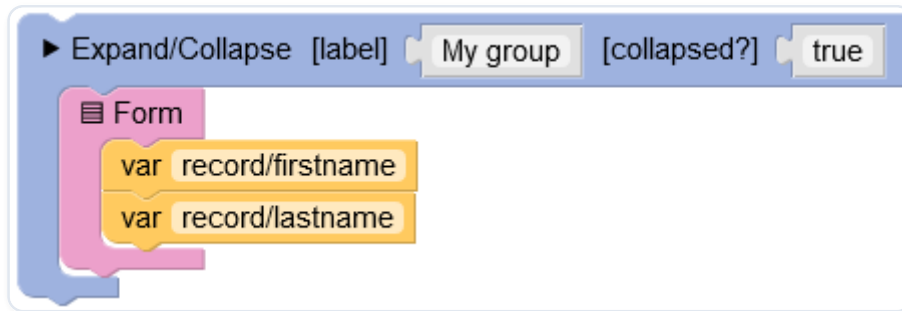
レコードの重複

フォームがレコードの複製のコンテキストで表示されるかどうかを示すブール値を返します。

```
duplicating record?
```

展開/折りたたみ

展開/折りたたみブロック内にコンテンツを表示します。

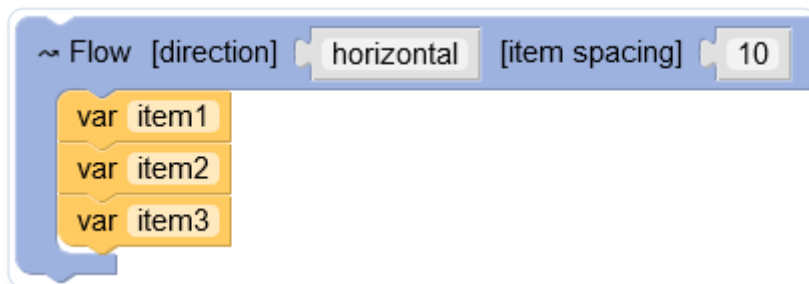


パラメーター

入力	説明
ラベル	矢印の横に表示されるラベル。
折りたたみ	グループが最初に折りたたまれているかどうかを示します。
content	折りたたみ可能なコンポーネントのリスト。

フロー

コンテンツを流動的に表示します。エレメントが1行または1列に収まらない場合は、折り返されて新しい行または列が始まります。



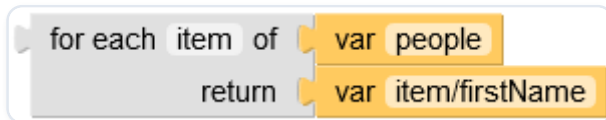
パラメーター

入力	説明
方向	「horizontal」または「vertical」は、コンテンツをキューに入れる方向を示します。
項目の間隔	エレメント間のスペース (ピクセル単位)。

入力	説明
content	水平または垂直に表示するエレメントのリスト。

For each

'of' 配列の各項目に適用された 'body' 関数の結果で構成される配列を返します。

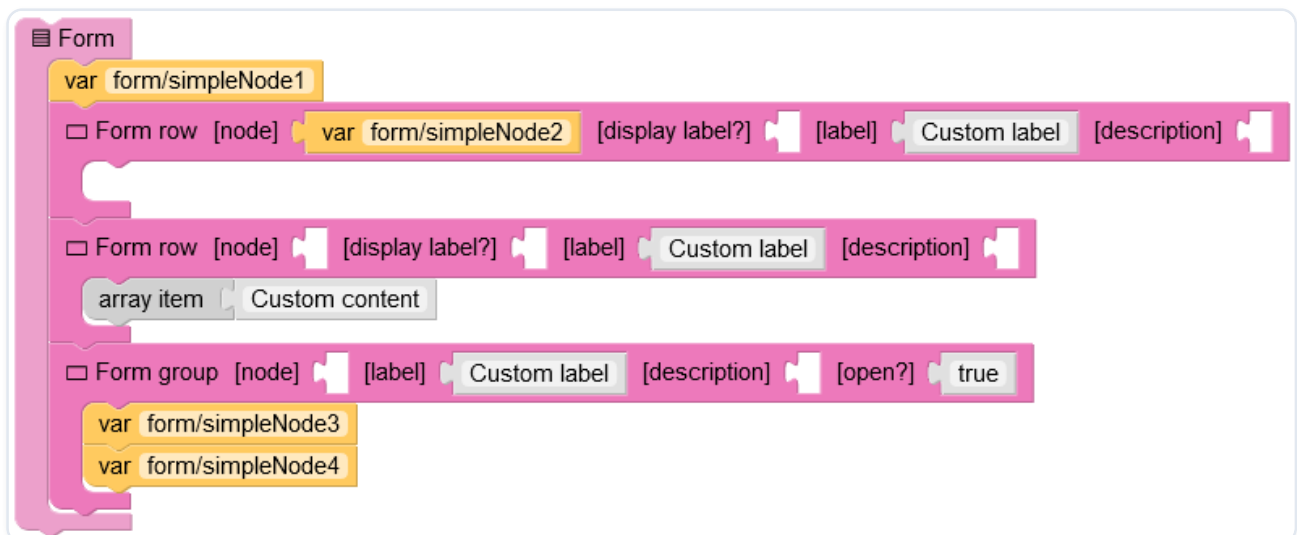


パラメーター

入力	説明
項目	現在の項目を含む変数の名前。
配列	変換する項目を含む配列。
戻り値	現在の項目に対する戻り値。

フォーム

左側にラベル、右側に値がある標準的な表のようなレイアウト。



パラメーター

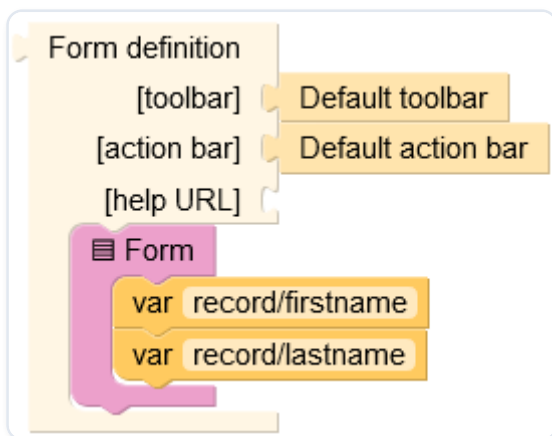
入力	説明
content	「フォーム行」、「フォームグループ」またはノードのリスト。ノードはモデル駆動型表示になり、「フォーム行」と「フォームグループ」はカスタムコンテンツを表示できます。

以下も参照してください。:

- [フォーム行](#)
- [フォームグループ](#)

フォーム定義

フォームの実際の定義。



パラメーター

入力	説明
ツールバー	フォームの上部に表示されるツールバーの定義。
アクションバー	フォームの下に表示されるアクションバーの定義。
ヘルプ URL	ヘルプページの URL。

入力	説明
内容	レイアウトの内容。

フォームグループ

「フォーム」内に、折りたたみ可能な項目のグループが表示されます。



パラメーター

入力	説明
ノード	設定されている場合、グループのラベル、説明、コンテンツはモデル駆動型になります。
ラベル	グループのラベル。ノードとラベルが設定されている場合は、モデル駆動型ラベルを上書きできます。
説明	グループの説明。ノードと説明が設定されている場合は、モデル駆動型の説明を上書きできます。
開く	グループが最初を開いているかどうかを示します。
内容	「フォーム行」、「フォームグループ」またはノードのリスト。ノードはモデル駆動型表示になり、「フォーム行」と「フォームグループ」はカスタムコンテンツを表示できます。

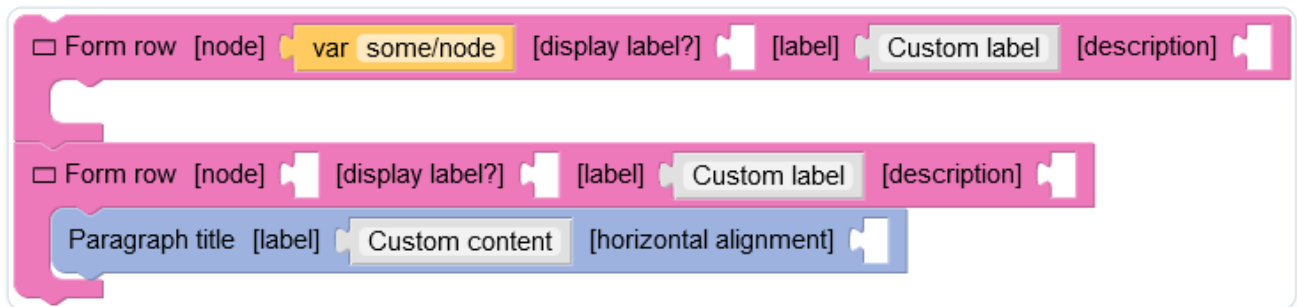
以下も参照してください。:

- フォーム

- フォーム行

フォーム行

「フォーム」または「フォームグループ」内に、基本的にラベルと値である行が表示されます。



パラメーター

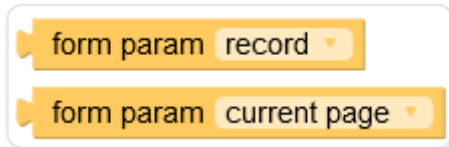
入力	説明
ノード	設定されている場合、ラベル、説明、コンテンツはモデル駆動型になります。
ラベルの表示	ラベルを表示するかどうかを示します。 「フォーム」で false に設定すると、コンテンツはラベルとコンテンツ領域の両方にまたがります。
ラベル	行のラベル。ノードとラベルが設定されている場合は、モデル駆動型ラベルを上書きできます。
説明	行の説明。ノードとラベルが設定されている場合は、モデル駆動ラベルを上書きできます。
内容	ラベルの横に表示されるコンテンツ。

以下も参照してください。:

- フォーム
- フォームグループ

フォームパラメーター

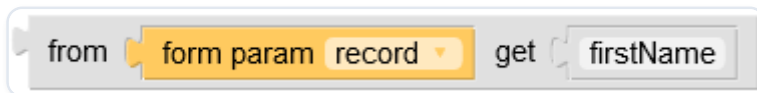
「レコード」は、表示されているレコードを表すノードです。「現在のページ」は、表示されているページを表す番号 (デフォルトでは 1) です。この値は、「前へ」および「次へ」ボタンで変更できます。



From...get

オブジェクトまたはノードから値を抽出します。

<from "node" get "path/to/value"> は <var "node/path/to/value"> と同等です。

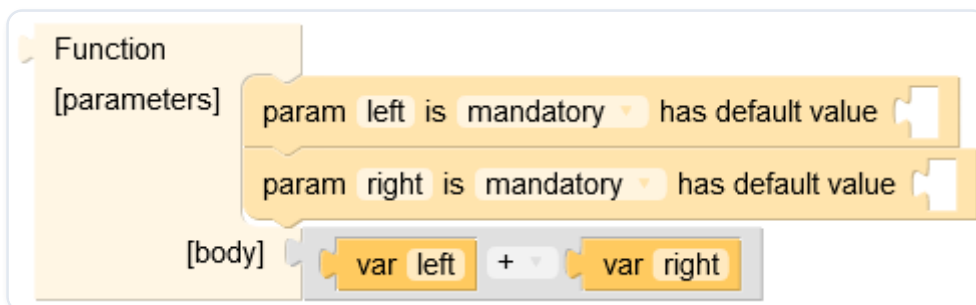


パラメーター

入力	説明
from	データを抽出するベースオブジェクトまたはノード。
get	'from' のオブジェクト内の、抽出するデータのパス。

関数

匿名関数の定義。関数は、その本体内のパラメーターを使用して結果を返します。



パラメーター

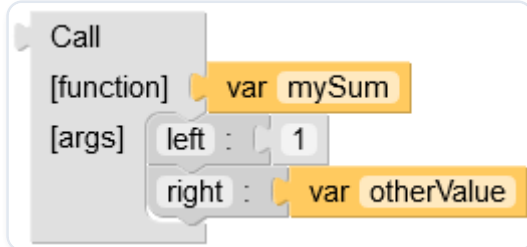
入力	説明
パラメーター	この関数が受け入れるパラメーターを定義する「param」ブロックのリスト。
本文	パラメーターを使用して関数の結果を記述する本体。

以下も参照してください。:

- 関数パラメーター
- 関数呼び出し

関数呼び出し

パラメーターに実際の値を設定して関数を呼び出します。



パラメーター

入力	説明
関数	呼び出す関数。
args	関数に渡す引数。

以下も参照してください。:

- 関数呼び出し引数

- 関数

関数呼び出し引数

呼び出された関数のパラメーターに渡す実際の値。

paramName : paramValue

パラメーター

入力	説明
名前	パラメーターの名前。
値	パラメーターに渡す実際の値。

以下も参照してください。:

- 関数呼び出し

関数パラメーター

これは関数のパラメーターの宣言です。

param optionalParam is optional has default value 0

パラメーター

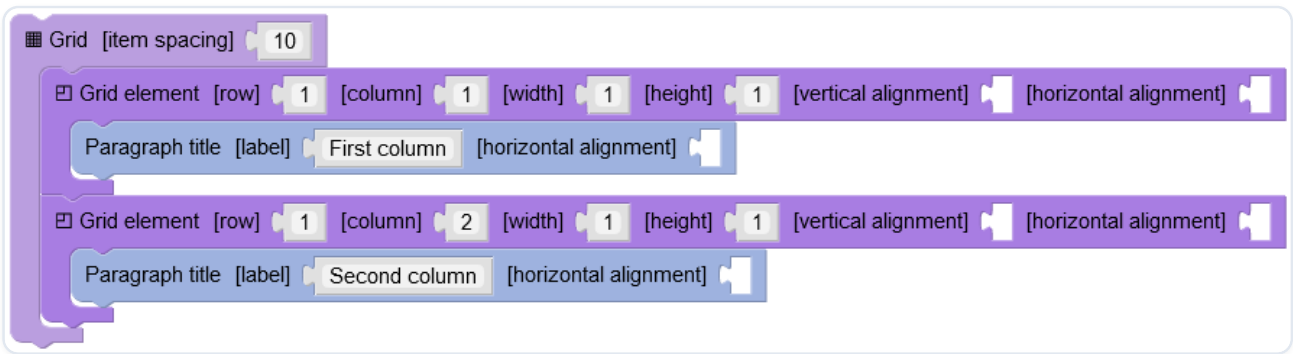
入力	説明
名前	パラメーターの名前。
必須	このパラメーターを呼び出し側が提供する必要があるかどうかを示します。
デフォルト値	設定されている場合、これは、呼び出し元が指定しない場合にこのパラメーターがとるデフォルト値です。

以下も参照してください。:

- 関数

グリッド

座標を使用してグリッド内にコンポーネントを配置できるレイアウト。



パラメーター

入力	説明
項目の間隔	エレメント間のスペース (ピクセル単位)。
内容	グリッドエレメントのリスト。コンテンツを表示する場所を指定します。

以下も参照してください。:

- グリッドエレメント

グリッドエレメント

「グリッド」内で、そのコンテンツの場所を指定します。

パラメーター

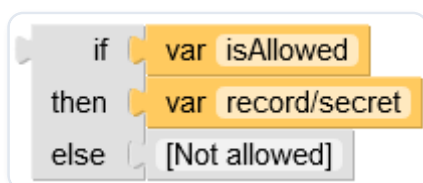
入力	説明
行	エレメントが始まる行。最小値は 1 です。
column	エレメントが始まる列。最小値は 1 です。
width	グリッドセル内のエレメントの幅。最小値は 1 です。
高さ	グリッドセル内のエレメントの高さ。最小値は 1 です。
縦方向の配置	エレメントの垂直方向の配置。設定されている場合、受け入れられる値は「start」、「end」、「center」です。
横方向の配置	エレメントの水平方向の配置。設定されている場合、受け入れられる値は「start」、「end」、「center」です。
内容	セル内のコンテンツ。

以下も参照してください。:

- グリッド

If

テストの結果に基づいて、「then」の内容または「else」の内容のいずれかを返すブロック。



パラメーター

入力	説明
テスト	返す値を決定するテスト。これはブール値にする必要があります。
then	テストが true の場合に返される値。
else	テストが false の場合に返される値。

入力パラメーター

入力パラメーターの生の値を返します。

 input parameter paramName

パラメーター

入力	説明
名前	パラメーター名。

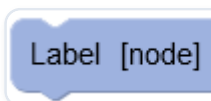
ロール

現在のユーザーが要求されたロールを持っているかどうかを示すブール値を返します。

 in role? [administrator]

ラベル

指定されたノードのラベルを表示します。

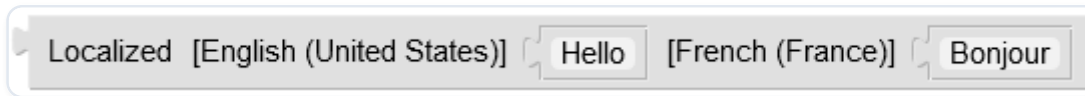
 Label [node]  var record/firstName

パラメーター

入力	説明
ノード	ラベルを表示するノード。

ローカライズ済み

ユーザーのロケールに応じて結果が決まるブロック。



パラメーター

入力	説明
[locale]	各パラメーターはロケールに関連しています。ブロックの結果は、現在のユーザーのロケールに対応するものになります。

メッセージ

指定されたインデックスの引数で、Java のような {0}、{1} などのプレースホルダーを置き換えて、指定されたテキストにフォーマットを設定します。



パラメーター

入力	説明
パターン	指定された引数に置き換えられるプレースホルダー ({0}、{1}など) を含むテキスト。
引数	パターン内のプレースホルダーを置き換える値のリスト。

次へボタン

「現在のページ」の値を 1 増やすボタン。

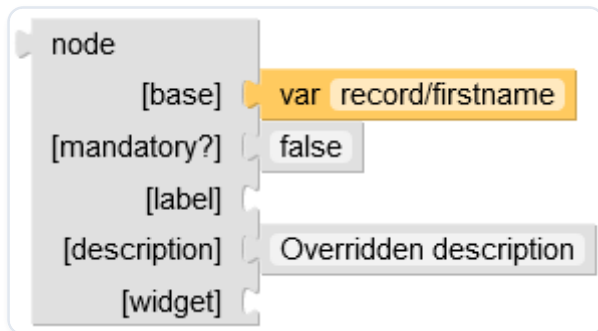


パラメーター

入力	説明
ラベル	設定されている場合、デフォルトのラベルが上書きされます。
デフォルト	これが Enter キーを押したときにトリガーされるアクションであるかどうかを示します。デフォルトのボタンは 1 つだけにする必要があります。

ノード

既存のノードの定義を上書きします。

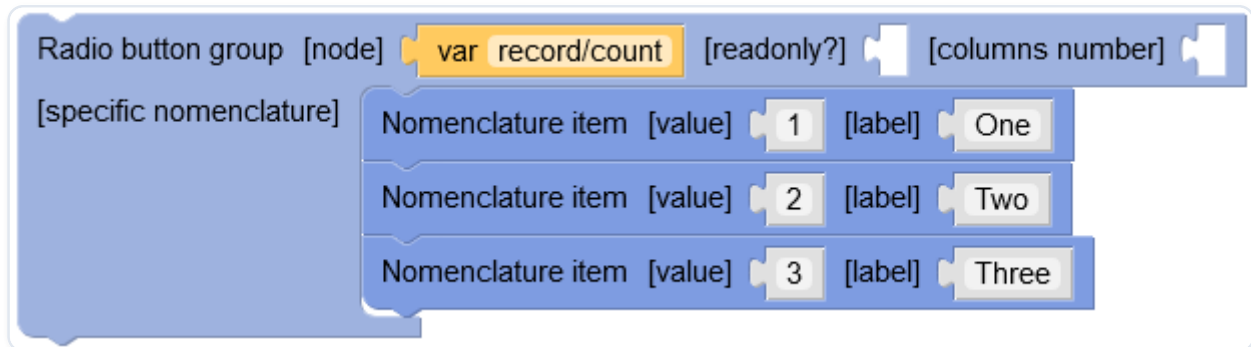


パラメーター

入力	説明
ベース	上書きするノード。
必須	設定されている場合、ベースノードの必須インジケータを上書きします。
ラベル	設定されている場合、ノードのラベルが上書きされます。
説明	設定されている場合、ノードのラベルが上書きされます。
ウィジェット	設定されている場合、ノードのデフォルトのウィジェットを上書きします。

命名項目

命名項目は (キー、ラベル) のペアです。

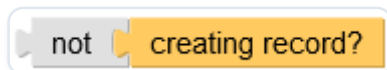


パラメーター

入力	説明
値	項目の値。
ラベル	項目のラベル。

Not

指定されたブール値の逆を返します。



オブジェクト

(キー、値) ペアのリストで構成されるオブジェクト。



パラメーター

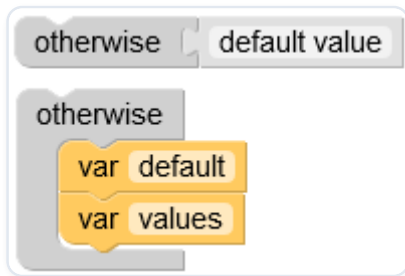
入力	説明
内容	「property」ブロックのリスト。

以下も参照してください。:

- プロパティ

それ以外の場合

「choose」ブロックの最終選択。一致する「when」ケースがない場合に返されます。



パラメーター

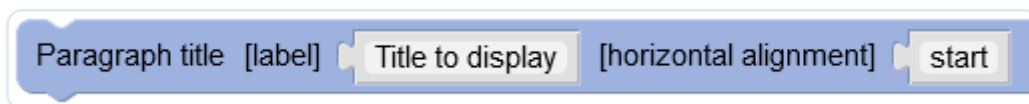
入力	説明
内容	'when' ケースが一致しなかった場合に返される値。

以下も参照してください。:

- 選択

段落タイトル

段落タイトルを表示します。



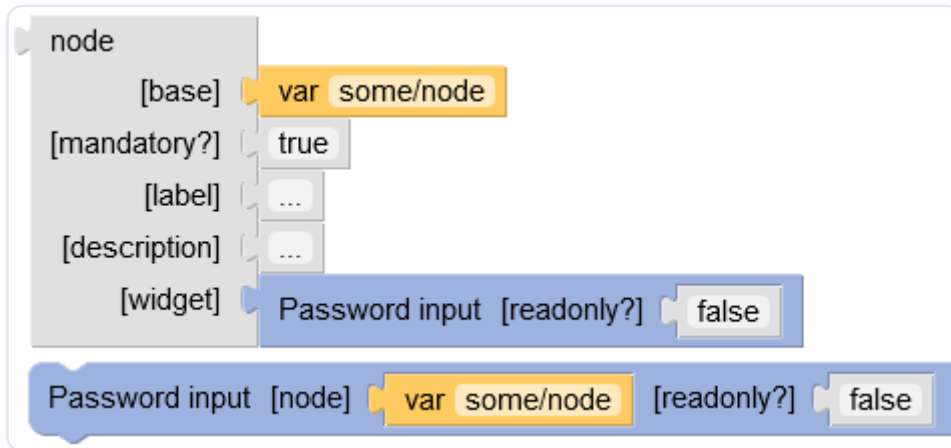
パラメーター

入力	説明
ラベル	表示するテキスト。

入力	説明
縦方向の配置	エレメントの水平方向の配置。設定されている場合、受け入れられる値は「start」、 「end」、「center」です。

パスワード

入力パスワードウィジェットを表示します。



パラメーター

入力	説明
node	ウィジェットを表示するノード。
読み取り専用	ウィジェットを読み取り専用モードにするかどうかを示します。

前のボタン

「現在のページ」の値を 1 減らすボタン。



パラメーター

入力	説明
ラベル	設定されている場合、デフォルトのラベルが 上書きされます。

入力	説明
デフォルト	これが Enter キーを押したときにトリガーされるアクションであるかどうかを示します。デフォルトのボタンは 1 つだけにする必要があります。

プロパティ

オブジェクトのプロパティ。

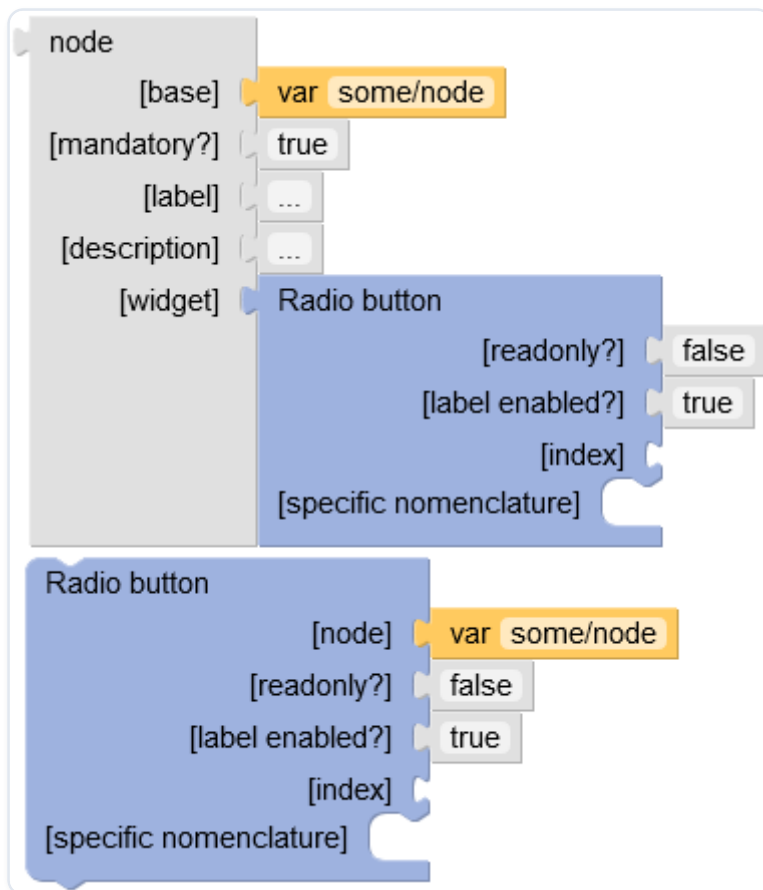


パラメーター

入力	説明
名前	プロパティの名前。特定のオブジェクトに対して一意にする必要があります。
値	プロパティの値。

ラジオボタン

ラジオボタンウィジェットを表示します。

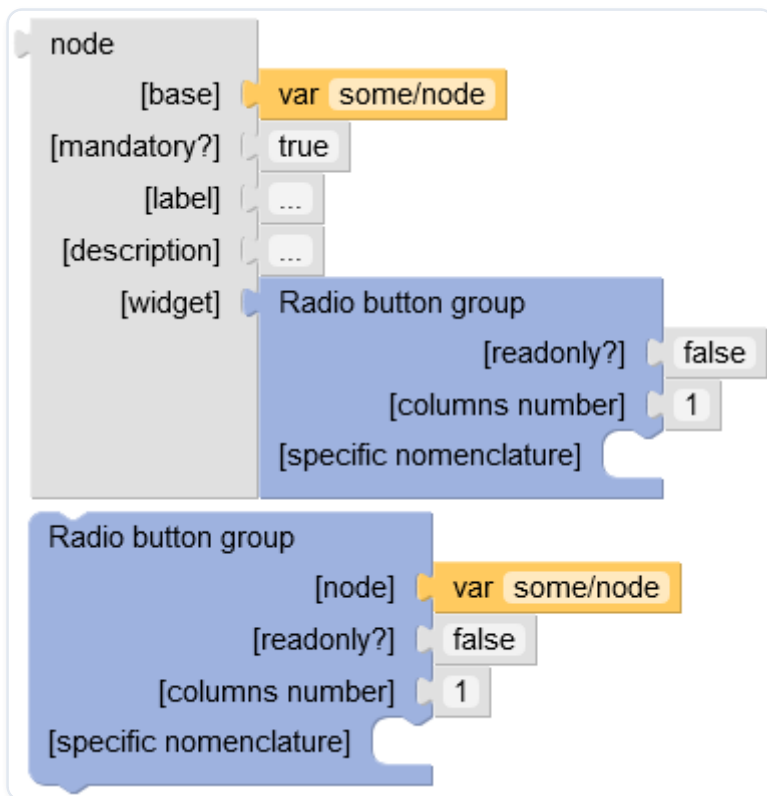


パラメーター

入力	説明
node	ウィジェットを表示するノード。
読み取り専用	ウィジェットを読み取り専用モードにするかどうかを示します。
ラベル有効	設定されている場合、ウィジェットの横に項目ラベルを追加するかどうかを示します。
インデックス	この列挙項目のインデックス。
特定の命名法	設定されている場合、モデル駆動型の命名法が上書きされます。

ラジオボタングループ

ラジオボタングループウィジェットを表示します。



パラメーター

入力	説明
node	ウィジェットを表示するノード。
読み取り専用	ウィジェットを読み取り専用モードにするかどうかを示します。
列数	設定されている場合、ラジオ ボタンのレイアウトに使用する列の数を定義します。
特定の命名法	設定されている場合、モデル駆動型の命名法が上書きされます。

元に戻すボタン

標準の「元に戻す」ボタン。

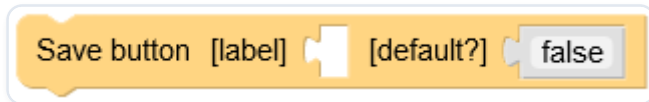


パラメーター

入力	説明
ラベル	設定されている場合、デフォルトのラベルが上書きされます。
デフォルト	これが Enter キーを押したときにトリガーされるアクションであるかどうかを示します。デフォルトのボタンは 1 つだけにする必要があります。

保存ボタン

標準の「保存」ボタン。

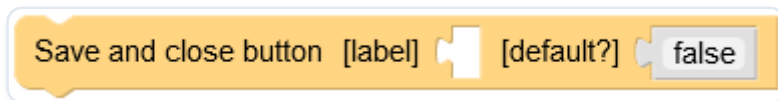


パラメーター

入力	説明
ラベル	設定されている場合、デフォルトのラベルが上書きされます。
デフォルト	これが Enter キーを押したときにトリガーされるアクションであるかどうかを示します。デフォルトのボタンは 1 つだけにする必要があります。

保存して閉じるボタン

標準の「保存して閉じる」ボタン。

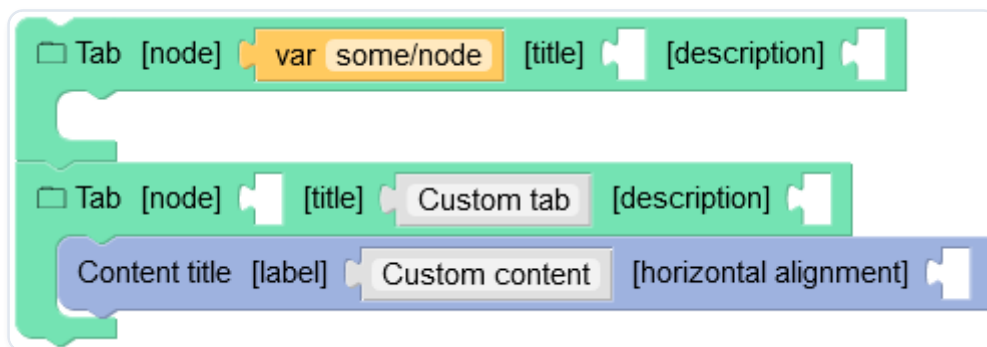


パラメーター

入力	説明
ラベル	設定されている場合、デフォルトのラベルが上書きされます。
デフォルト	これが Enter キーを押したときにトリガーされるアクションであるかどうかを示します。デフォルトのボタンは 1 つだけにする必要があります。

タブ

「タブコンテナ」内のタブ。



パラメーター

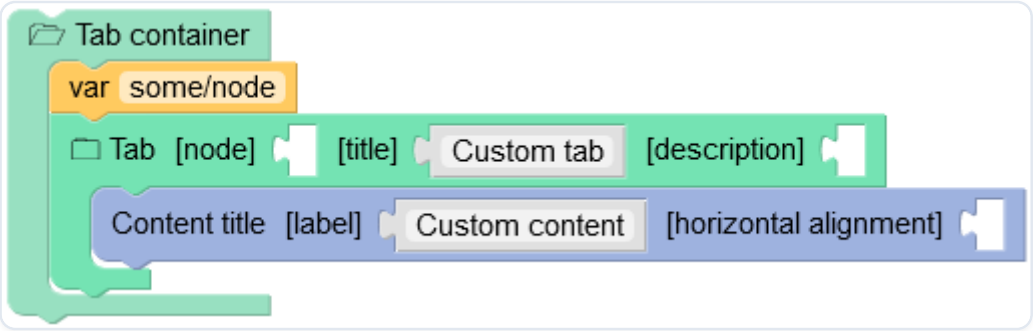
入力	説明
ノード	存在する場合、タブのタイトル、説明、コンテンツはモデル駆動型になります。
タイトル	タブのタイトル。ノードとタイトルが設定されている場合、モデル駆動型タイトルを上書きできます。
説明	タブの説明。ノードと説明が設定されている場合は、モデル駆動型の説明を上書きできます。
内容	タブのコンテンツ。ノードとコンテンツが設定されている場合は、モデル駆動型コンテンツを上書きできます。

以下も参照してください。:

- タブコンテナ

タブコンテナ

タブ名のリストと 1 つのアクティブなタブを表示するタブのコンテナ。



パラメーター

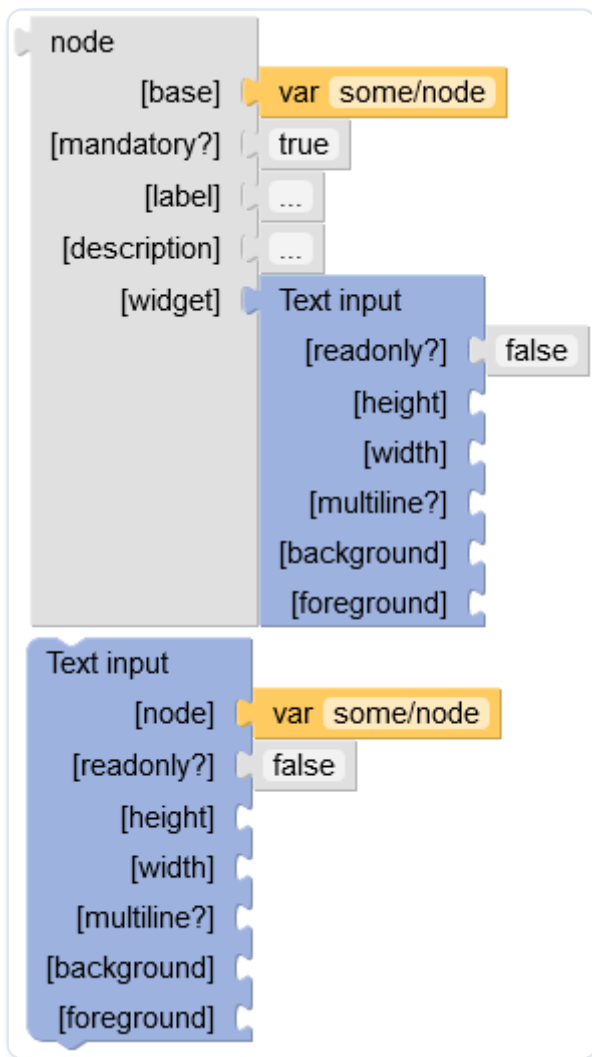
入力	説明
内容	「タブ」のリスト。

以下も参照してください。:

- タブ

テキスト入力

テキスト入力ウィジェットを表示します。



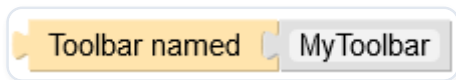
パラメーター

入力	説明
ノード	ウィジェットを表示するノード。
読み取り専用	ウィジェットを読み取り専用モードにするかどうかを示します。
高さ	設定されている場合、ウィジェットの高さをピクセル単位で定義します。
幅	設定されている場合、ウィジェットの幅をピクセル単位で定義します。
複数行	設定されている場合、ウィジェットが複数の行にまたがるかどうかを定義します。

入力	説明
背景	設定されている場合、背景色を 16 進形式で定義します。
前景	設定されている場合、前景色を 16 進形式で定義します。

ツールバーの名前

指定された名前を持つツールバー。

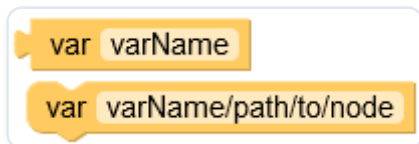


パラメーター

入力	説明
名前	表示するツールバーの名前。

変数

'with...declare' ブロックによって宣言された変数への参照。変数がオブジェクトまたはノードである場合は、パスも受け入れられます。

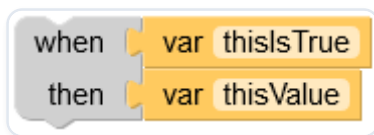


パラメーター

入力	説明
式	'with...declare' ブロックによって宣言された変数への参照。変数がオブジェクトまたはノードである場合は、パスも受け入れられます。

When

「choose」ブロック内の選択肢。テストが true の場合は、「then」コンテンツが返され、それ以外の場合は次のブロックがテストされます。



パラメーター

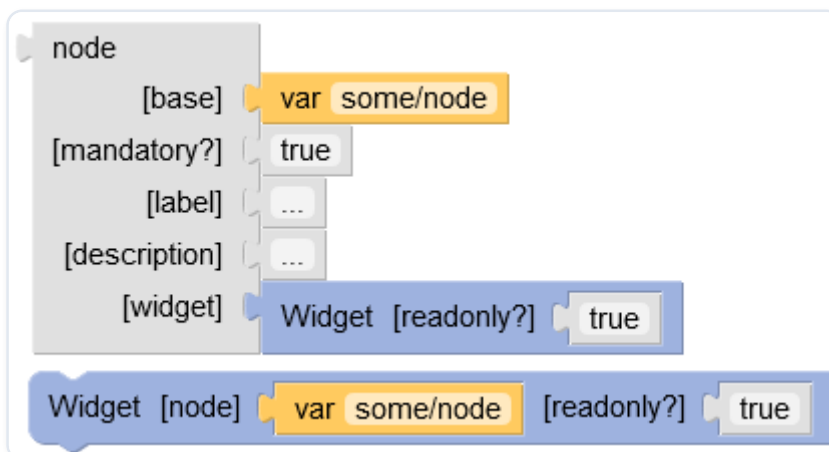
入力	説明
when	このケースを解決する必要があるかどうかを示すブールテスト。
then	テストが true の場合、この値が返されます。

以下も参照してください。:

- [選択](#)

ウィジェット

デフォルトのウィジェットを表示します。

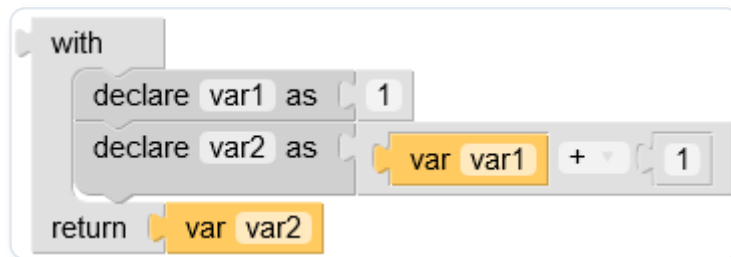


パラメーター

入力	説明
ノード	ウィジェットを表示するノード。
読み取り専用	ウィジェットを読み取り専用モードにするかどうかを示します。

With

「return」ステートメントで使用する変数を宣言します。



パラメーター

入力	説明
変数	「宣言」ブロックのリスト。
戻り値	返される値。'variables' ステートメントで宣言された変数は、ここで 'var' ブロックを使用して使用できます。

以下も参照してください。:

- [宣言](#)

パブリケーション REST API

この章では、データモデルのパブリッシュに使用される REST API について詳しく説明します。

DMA データモデルのパブリッシュ

このサービスは、Data Model Assistant (DMA) で定義されたデータモデルをパブリッシュするために設計されています。これにより、以下を実行できます。

- DMA で定義されているすべてのデータモデルをリスト表示
- データモデルの情報を取得
- DMA で定義された構成に従って、既存のデータモデルをパブリッシュ
- DMA で定義された構成を上書きして、既存のデータモデルを特定の XML スキーマドキュメントにパブリッシュ
- データモデルのすべてのバージョンをリスト表示

- データモデルの特定のバージョンを取得
- データモデルのすべての埋め込みパブリケーションを取得

サービスの完全な説明は、以下の URL を使用して OpenAPI 仕様にアクセスすることで利用できます。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/datamodel/v1/dma/
```

ワークフローモデル

ワークフローには、「ライブラリ」または「特定」と「スクリプト」の 3 種類のステップがあります。

「ライブラリ」は `module.xml` で定義され、再利用可能な Bean です。「ライブラリ」Bean を使用すると、人間工学が向上し、パラメーターが定義画面に動的に表示されます。

「特定の」オブジェクトは、クラス名のみで定義された Bean です。この場合、表示は動的ではありません。

「スクリプト」は、カスタムタスクを定義するために使用される DSL (ドメイン固有言語) に基づく手続き型言語です。EBX スクリプト IDE は、スクリプトの作成と編集に使用されます。ワークフローコンテキストのパラメーターは、定義画面に動的に表示されます。

- Bean カテゴリ
- ScriptTask のサンプル
 - Java コード
- ScriptTaskBean のサンプル
 - Java コード
 - module.xml による構成
- UserTask のサンプル
 - module.xml によるサービス宣言
 - より複雑な UserTask
- 条件のサンプル
 - Java コード
- ConditionBean のサンプル

- Java コード
- module.xml による構成
- SubWorkflowsInvocationBean のサンプル
 - Java コード
 - module.xml による構成
- WaitTaskBean のサンプル
 - Java コード
 - module.xml による構成
- ActionPermissionsOnWorkflow のサンプル
 - Java コード
 - module.xml による構成
- WorkflowTriggerBean のサンプル
 - Java コード
 - module.xml による構成
- プロセスインスタンスを開始するトリガーのサンプル
 - 例

Bean カテゴリ

STEP	ライブラリ	特定
スクリプト	ScriptTaskBean	ScriptTask
条件	ConditionBean	条件
ユーザータスク	UserTask	

ScriptTask のサンプル

Java コード

スクリプトタスクは、次の例のようにメソッド `execute` を上書きする必要があります。

```
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());

    }

}

```

以下も参照してください。:

- [com.orchestranetworks.workflow.ScriptTask](#)

ScriptTaskBean のサンプル

Java コード

スクリプト タスク Bean は、次の例のようにメソッド `executeScript` を上書きする必要があります。

```

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());

    }

}

```

以下も参照してください。:

- [com.orchestranetworks.workflow.ScriptTaskBean](#)

module.xml による構成

スクリプト タスク Bean は `module.xml` で宣言する必要があります。

```

<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>

```

UserTask のサンプル

module.xml によるサービス宣言

ビルトインサービスは、`module.xml` で宣言して、ユーザータスク定義で使用できます。

```

<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>
    </service>
</services>

```

```

        <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>

```

より複雑な UserTask

GUI は上記の例と非常に似ています。呼び出す 'UserTask' を拡張するクラスを定義するには、フィールド 'Rule' を入力する必要があります。

```

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");

    }

}

```

以下も参照してください。:

- [com.orchestranetworks.workflow.UserTask](#)

条件のサンプル

Java コード

メソッド `evaluate` を上書きする必要があります。

```

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;

    }

}

```

```
}
```

以下も参照してください。:

- [com.orchestranetworks.workflow.Condition](#)

ConditionBean のサンプル

Java コード

次のサンプルのように、メソッド `evaluateCondition` を上書きする必要があります。

```
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;

    }

}

```

以下も参照してください。:

- [com.orchestranetworks.workflow.ConditionBean](#)

module.xml による構成

条件 Bean は `module.xml` で宣言する必要があります。

```
<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>

```

SubWorkflowsInvocationBean のサンプル

Java コード

```

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 RuntimeException

```

```

{

    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").e
quals("true")

        &&
validateProductDirection.getDataContext().getVariableString("Accepted").eq
uals(

            "true"))

        aContext.setVariableString("validation", "ok");

    }

}

```

以下も参照してください。:

- [com.orchestranetworks.workflow.SubWorkflowsInvocationBean](#)

module.xml による構成

`SubWorkflowsInvocationBean` Bean は `module.xml` で宣言する必要があります。

```
<module>

    <beans>

        <bean
class="com.orchestranetworks.workflow.test.MySubWorkflowsInvocationBean"/>

    </beans>

</module>
```

WaitTaskBean のサンプル

Java コード

```
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"));

    }

}

```

以下も参照してください。:

- [com.orchestranetworks.workflow.WaitTaskBean](#)

module.xml による構成

`WaitTaskBean` Bean は `module.xml` で宣言する必要があります。

```

<module>

    <beans>

        <bean
className="com.orchestranetworks.workflow.test.MyWaitTaskBean"/>

```

```
</beans>

</module>
```

ActionPermissionsOnWorkflow のサンプル

Java コード

```
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();

    }

}

```

以下も参照してください。:

- [com.orchestranetworks.workflow.ActionPermissionsOnWorkflow](#)

module.xml による構成

`ActionPermissionsOnWorkflow` Beanは、`module.xml` で宣言する必要があります。

```

<module>

    <beans>

        <bean
        className="com.orchestranetworks.workflow.test.MyDynamicPermissions"/>

    </beans>

</module>

```


WorkflowTriggerBean のサンプル

Java コード

```
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 RuntimeException

{

    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.T0, "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.T0, "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);

}

}

```

以下も参照してください。:

- [com.orchestranetworks.workflow.WorkflowTriggerBean](#)

module.xml による構成

WorkflowTriggerBean Bean は、 module.xml で宣言する必要があります。

```
<module>

    <beans>

        <bean
className="com.orchestranetworks.workflow.test.MyWorkflowTriggerBean"/>

    </beans>

</module>
```

プロセスインスタンスを開始するトリガーのサンプル 例

```
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();

    }

    //...

}

```

ユーザーインターフェイス

- インターフェイスのカスタマイズ
- ユーザーサービス
 - 概要
 - クイックスタート
 - ユーザーサービスの実装

。ユーザーサービスの宣言

- 開発の推奨事項

インターフェイスのカスタマイズ

TIBCO EBX® グラフィカルインターフェイスは、さまざまな EBX® API を通じてカスタマイズできます。

- EBX® の Web コンポーネントとしての使用
- ユーザーサービスの追加
- フォームのカスタマイズ
- ウィジェットのカスタマイズ
- テーブルフィルターのカスタマイズ
- レコードラベルのカスタマイズ
- CSS と JavaScript の組み込み

EBX® の Web コンポーネントとしての使用

EBX® は、Web コンポーネント APIのおかげで、サポートされている Web ブラウザーからアクセスできるあらゆるアプリケーションに統合できます。

典型的な使用法は、EBX® ビューを組織のイントラネットフレームワークに統合することです。Web コンポーネントは、ユーザーサービスを使用して EBX® ユーザーインターフェイスから呼び出すこともできます。

EBX® の全部または一部を Web ページに埋め込むには、HTML タグ `<iframe>` を使用して EBX® への URL を指定する必要があります。この URL は手動で指定することも、`UIHttpManagerComponent` API を使用して指定することもできます。1 つの Web ページには、EBX® を統合する複数の iframe を含めることができます。これにより、EBX® からテーブル、フォーム、階層ビューなどで構成されるポータルを作成できます。

以下も参照してください。:

- TIBCO EBX® の Web コンポーネントとしての使用

ユーザーサービスの追加

ユーザーサービスは EBX® の拡張機能であり、ユーザーが特定の機能や高度な機能にアクセスできるようにするグラフィカルユーザーインターフェイス (GUI) を提供します。

標準の EBX® ユーザーインターフェイスと同じビジュアルコンポーネントとデータ検証メカニズムを使用して、強力なカスタムユーザーサービスを開発できます。

以下も参照してください。:

- [ユーザーサービスの概要](#)

フォームのカスタマイズ

さまざまなツールと API を使用して、ユーザーインターフェイス内のフォームのデフォルトのレイアウトと動作を上書きできます。

カスタムフォームエディター	DMA の「カスタムフォーム」拡張機能を使用すると、ユーザーインターフェイスのフォームの既定のレイアウトを上書きできます。この拡張機能はグラフィカルエディターを提供し、デフォルトのレイアウトを使用するレコードフォームと同じコンポーネントと標準の動作を備えながら、フォームのレイアウトをカスタマイズできます。 以下も参照してください。: • カスタムフォーム
プログラムによるフォームレイアウト	<code>UIForm</code> API を使用して、フォームのデフォルトレイアウトを上書きすることもできます。この API は、EBX® の標準入力コンポーネントを提供します。これにより、デフォルトレイアウトを使用するレコードフォームと同じコンポーネントと標準動作

	を持ちながら、フォームのレイアウトをカスタマイズできるようになります。 以下も参照してください。: • <code>UIForm</code> • <code>UIFormPaneWriter</code> • <code>UIWidget</code> • <code>UIFormHeader</code> • <code>UIFormBottomBar</code>
フォームレイアウトとしてのユーザーサービス	ユーザーサービスを使用して、フォームのデフォルトのレイアウトを上書きすることもできます。この API を使用すると、デフォルトのレイアウトを使用するレコードフォームと同じコンポーネントを持ちながら、カスタマイズ可能な動作でフォームのレイアウトをカスタマイズできます。 以下も参照してください。: • <code>UserServiceRecordFormFactory</code> • 概要

ウィジェットのカスタマイズ

Java API にはカスタムウィジェットが含まれており、フィールドまたはフィールドグループのユーザーインターフェイスコンポーネントの開発が可能です。カスタムウィジェット (`UICustomWidget`) を使用すると、特定のノードの入力コンポーネントまたは表示コンポーネントが配置される領域を制御できます。これにより、HTML で完全にカスタマイズ可能な入力コンポーネントと表示コンポーネントを持つことができます。標準コンポーネント (`UIWidgets`) が利用可能で、使用できます。カスタムウィジェットは、フォーム内の入力コンポーネント、フォーム内の表示、テーブルセル内の表示など、いくつかの表示側面を実装できます。カスタムウィジェットが独自の HTML コンポーネントを作成すると、フォームの送信時にデータベースに値を保存できます。

以下も参照してください。:

- `UICustomWidget`

テーブルフィルターのカスタマイズ

テーブルフィルターを使用すると、特定のテーブルに対して、テーブルビューにフィルターを適用するための条件入力フォームを作成できます。`UITableFilter` API は、カスタム UI でテーブルフィルターを実装するために使用されます。この API は、基になるデータ形式に自動的に適応する UI を作成するメソッドを提供します (たとえば、該当する場合はコンボ ボックスを表示するなど)。

以下も参照してください。:

- `UITableFilter`
- データモデルエレメントのプロパティ
- `UILabelRendererForHierarchy`

レコードレーベルのカスタマイズ

EBX® は、ラベルを使用して、特定のレコードへの参照 (外部キーなど) を表示します。ラベルは、レコードフォームのタイトルや階層ビューでも使用されます。このラベルは、式を使用してモデル内でカスタマイズできます。`UILabelRenderer` API を使用してラベルをカスタマイズすることもできます。

以下も参照してください。:

- `UILabelRenderer`

CSS と JavaScript の組み込み

登録モジュールで宣言することで、各 EBX® ページに CSS ファイルと JavaScript ファイルを統合することができます。JavaScript ファイルの組み込みは、コンテキストに応じて開発を通じて条件が課される場合があります。

以下も参照してください。:

- [モジュール登録](#)
- [開発の推奨事項](#)
- `UIDependencyRegisterer`

ユーザーサービス

- [概要](#)
- [クイックスタート](#)
- [ユーザーサービスの実装](#)
- [ユーザーサービスの宣言](#)

概要

ユーザーサービスは TIBCO EBX® の拡張機能であり、ユーザーが特定の機能または高度な機能にアクセスできるようにするグラフィカルユーザーインターフェイス (GUI) を提供します。

API が利用可能であり、標準の EBX® ユーザーインターフェイスと同じビジュアルコンポーネントとデータ検証メカニズムを使用して、強力なカスタムユーザーサービスを開発できます。

- [ネイチャー](#)
- [宣言](#)
- [表示](#)
- [レガシーユーザーサービス](#)

ネイチャー

ユーザーサービスには、ネイチャーと呼ばれるさまざまなタイプがあります。ネイチャーは、サービスを実行するために選択する必要がある最小限の要素 (データスペース、データセット、テーブル、レコード...) を定義します。次の表に、使用可能なネイチャーを示します。

ネイチャー	説明
<code>データスペース</code>	データスペース (ブランチまたはスナップショット) のアクションメニューから、または現在の選択がデータスペースの選択を意味する

ネイチャー	説明
	任意のコンテキストから起動できるユーザーサービスのネイチャー。
データセット	データセットのアクションメニューから、または現在の選択がデータセットの選択を意味する任意のコンテキストから起動できるユーザーサービスのネイチャー。
テーブルビュー	選択したビューに関係なく、テーブルのツールバーから、または現在の選択がテーブルの選択を意味するコンテキストから起動できるユーザーサービスのネイチャー。
レコード	レコードフォームのツールバーから、または現在の選択が 単一レコードの選択を意味する任意のコンテキストから起動できるユーザーサービスのネイチャー。
階層	階層ビューが選択されたときにテーブルのツールバーから起動できるユーザーサービスのネイチャー。
階層ノード	テーブル階層ビューノードのメニューから起動できるユーザーサービスのネイチャー。現在、 <i>record</i> 階層ノードのみがサポートされています。
関連付け	関連付けのターゲットテーブルビューから、または現在の選択が関連付けのターゲットテーブルビューの選択を意味するコンテキストに対して起動できるユーザーサービスのネイチャー。
関連付けレコード	関連付けノードのターゲットレコードの形式から、または現在の選択が 単一関連付けターゲットレコードの選択を意味する任意のコンテキストから起動できるユーザーサービスのネイチャー。

宣言

ユーザーサービスは、次の 2 つのレベルで宣言できます。

- モジュール
- データモデル

データモデルによって宣言されたサービスは、現在の選択にこのモデルのデータセットが含まれている場合にのみ起動できます。ユーザーサービスを **データスペース** のネイチャーにすることはできません。

モジュールによって宣言されたサービスは、任意のデータスペースまたはデータセットに対して起動できます。

宣言により、ユーザーサービスに有効な選択に制限を追加できます。

表示

次の図は、ユーザーサービスの機能領域を示しています。

The screenshot shows a user service form for 'John Doe'. The interface includes a header with the user name, a top toolbar with icons for help, messages, and close, and a navigation bar with 'Actions' and navigation buttons. The main content area is divided into 'Main' and 'Groups' tabs. The 'Main' tab contains a form with fields for Last Name, First Name, Start Date, Country, Job, Email, Phone, and Mobile Phone. At the bottom of the form are buttons for 'Save', 'Save and close', 'Revert', and 'Close'. The interface is annotated with red callouts: 1 points to the user name, 2 to the messages icon, 3 to the 'Actions' dropdown, 4 to the navigation buttons, 5 to the help icon, 6 to the close icon, 7 to the 'Main' tab, 8 to the 'Groups' tab, and 9 to the 'Save' button. Labels 'A' and 'B' are placed on the interface: 'A' is in the header area, and 'B' is in the main form area.

A. ヘッダー

B. フォーム

1. パンくずリスト
2. メッセージボックスボタン
3. トップツールバー
4. ナビゲーションボタン

5. ヘルプボタン
6. 閉じるボタン (ポップアップのみ)
7. タブ
8. フォームペイン (タブごとに 1 つ)
9. 下部ボタン

ほとんどの領域はオプションであり、カスタマイズ可能です。詳細については、[クイックスタート](#)、[ユーザーサービスの実装](#)、[ユーザーサービスの宣言](#) を参照してください。

レガシーユーザーサービス

5.8.0 より前のバージョンでは、ユーザーサービスは XML で宣言され、Servlet/JSP に基づいていました。このタイプの宣言は使用するべきではありませんが、[レガシードキュメント](#) は引き続き利用できます。

クイックスタート

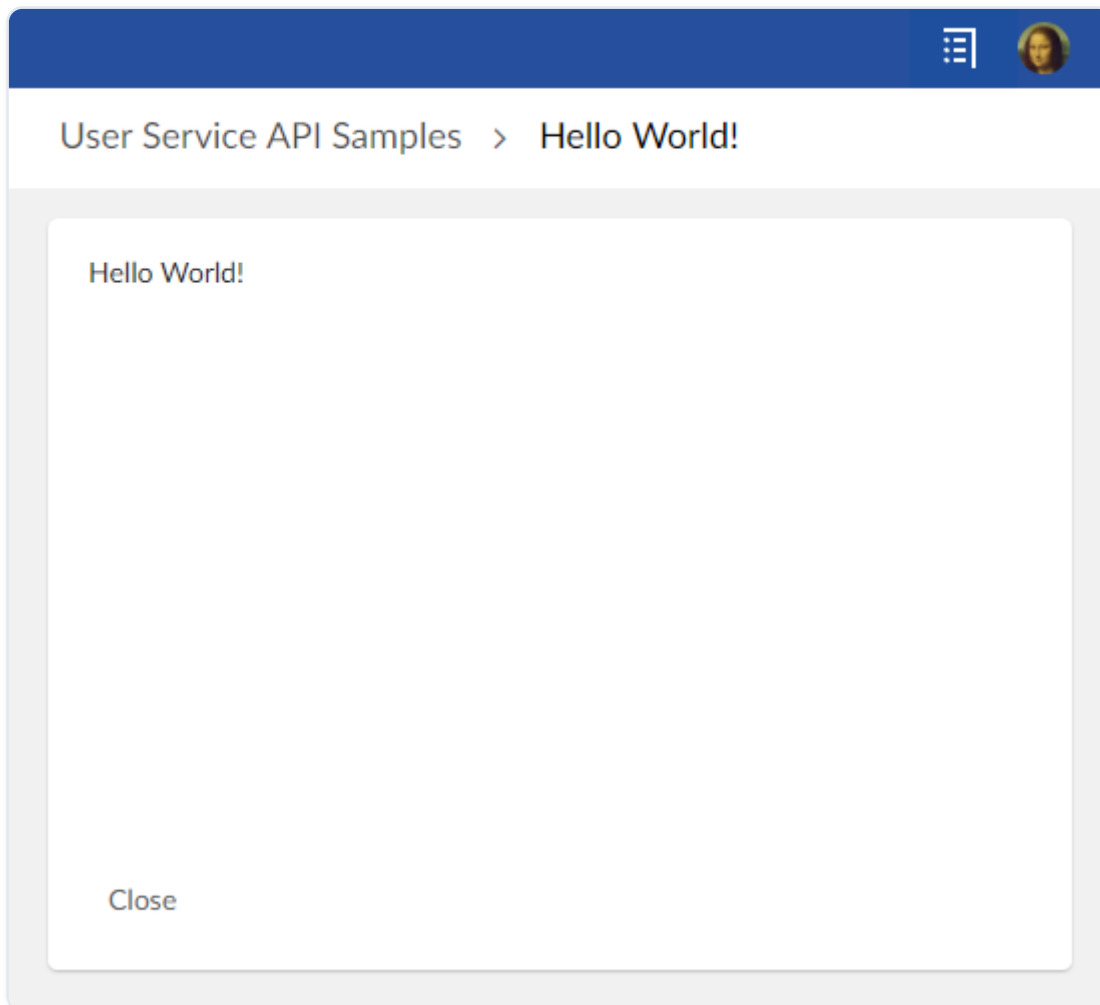
- [メインクラス](#)
- [Hello world](#)

メインクラス

最小要件は、2つのクラスを実装することです。1 つはサービス宣言用で、もう 1 つは実装自体用です。

Hello world

サンプルは、単に「Hello world」メッセージを表示するデータセットユーザーサービスであり、データセットのアクションメニューから起動できます。



サービス実装クラスは、インターフェイス `UserService<DatasetEntitySelection>` を実装する必要があります。

```
/**  
  
 * This service displays hello world!  
  
 */  
  
public class HelloWorldService implements  
    UserService<DatasetEntitySelection>  
{  
  
    public HelloWorldService()  
  
    {  
  
    }  
}
```

```

@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;

    }

}

```

宣言クラスは、インターフェイス `UserServiceDeclaration.OnDataset` を実装する必要があります。

```

/**

 * 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 HelloWorldService();

}

@Override

public void defineActivation(ActivationContextOnDataset aContext)

{

    // The service is activated for all datasets instanciaded 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)

    {

    }

}

```

このサンプルでは、データモデルによってユーザサービスが登録されています。データモデルでは、以下のコードを実装したスキーマ拡張を定義する必要があります。

```

public class CustomSchemaExtensions implements SchemaExtensions

{

    @Override

    public void defineExtensions(SchemaExtensionsContext aContext)

    {

        // Register the service.

```

```

        aContext.registerUserService(new HelloWorldServiceDeclaration());
    }
}

```

スキーマ拡張宣言の詳細については、[SchemaExtensions](#) を参照してください。

ユーザーサービスの実装

- 実装インターフェイス
- ライフサイクルとスレッドモデル
- オブジェクトコンテキスト
- 表示設定
- データベースの更新
- Ajax
- REST データサービス
- ファイルのアップロード
- ファイルのダウンロード
- 表示なしのユーザーサービス
 - 既知の制限事項

実装インターフェイス

次の表は、ネイチャーごとに実装するインターフェイスを示しています。

ネイチャー	インターフェイス
Dataspace	UserService<DataspaceEntitySelection>
Dataset	UserService<DatasetEntitySelection>
TableView	UserService<TableViewEntitySelection>
Record	UserService<RecordEntitySelection>
Hierarchy	UserService<HierarchyEntitySelection>

ネイチャー	インターフェイス
HierarchyNode	UserService<HierarchyNodeEntitySelection>
Association	UserService<AssociationEntitySelection>
AssociationRecord	UserService<AssociationRecordEntitySelection>

ライフサイクルとスレッドモデル

ユーザーサービスの実装クラスは次のとおりです。

- 宣言 `createUserService()` メソッドを呼び出すことにより、最初の HTTP リクエストでインスタンス化されます。
- 現在のページがスコープ外になるか、セッションがタイムアウトすると破棄されます。

このクラスへのアクセスは TIBCO EBX® によって同期され、一度に 1 つの HTTP リクエストのみが処理されるようにします。したがって、クラスはスレッドセーフである必要はありません。

ユーザーサービスには属性がある場合があります。これらの属性の状態は、HTTP リクエスト間で保持されます。ただし、開発者は、EBX® サーバーに過負荷をかけないように、これらの属性でメモリなどのリソースを適度に使用する必要があることに注意する必要があります。

オブジェクトコンテキスト

オブジェクトコンテキストは、ユーザーサービスによって管理されるオブジェクトのコンテナです。このコンテキストは、ユーザーサービスによるメソッド

`UserService.setupObjectContext` の実装によって初期化および変更されます。

オブジェクトコンテキストのオブジェクトは、オブジェクトキーによって識別されます。

```
ObjectKey customerKey = ObjectKey.forName("customer");
```

オブジェクトには、次のものがあります。

- レコード
- データセット
- 永続化前の新しいレコード
- 動的オブジェクト

オブジェクトコンテキストは HTTP リクエスト間で維持され、通常は最初のリクエスト時にのみ設定する必要があります。

永続化されると、新しいレコードオブジェクトは自動的に プレーンレコードオブジェクトに変更されます。

[Adaptation](#) と同様に、[path](#) 式はオブジェクトのサブエレメントを参照するために使用されます。

次の例では、ペインライターがオブジェクトの属性にマップされたフォーム入力を追加します。

```
// Add an input field for customer's last name.

aWriter.setCurrentObject(customerKey);

aWriter.addFormRow(Path.parse("lastName"));
```

次の例では、イベントコールバックがオブジェクトの属性の値を取得します。

```
// Get value of customer's last name.

ValueContext customerValueContext =
aValueContext.getValueContext(customerKey);

String lastName = customerValueContext.getValue(Path.parse("lastName"));
```

動的オブジェクトは、スキーマがユーザーサービス自体によって定義されるオブジェクトです。プログラムでスキーマを定義するための API が提供されています。この API では、インスタンスエレメント (インスタンスノード) のみを定義できます。テーブルの定義はサポートされていません。タイプ、ラベル、カスタムウィジェット、列挙型、プログラムによる制約など、標準の EBX® データモデルで利用できる他のほとんどの機能をサポートします。

次の例は、同じスキーマを持つ 2 つのオブジェクトを定義しています。

```
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 family 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);

}

...

}

```

表示設定

表示は、ユーザーサービスによるメソッド `UserService.setupDisplay` の実装によって設定されます。

このメソッドはリクエストごとに呼び出され、以下を設定できます。

- タイトル (デフォルトはユーザーサービス宣言で指定されたラベル)
- コンテキストヘルプ URL
- ブレッドクラム
- ツールバー
- 下部ボタン

必要に応じて、ヘッダーと下部ボタンを非表示にすることができます。

表示設定は永続化されず、HTTP リクエストごとに、メソッド `UserService.setupDisplay` を呼び出す前にデフォルトにリセットされます。

下部ボタン

ボタンには、アクションと送信の2種類があります。

ボタンは、フォームを送信せずに *action* イベントをトリガーします。デフォルトでは、ユーザーは、ページを離れることにより、最後の変更が失われることを認める必要があります。この動作はカスタマイズできます。

ボタンは、常にフォームを送信する *submit* イベントをトリガーします。

イベントの詳細については、次のセクションを参照してください。

コンテンツコールバック

このコールバックは通常、インターフェイス `UserServicePane` を実装して、プレーンな EBX® フォームをレンダリングします。コールバックは、タブ付きの EBX® フォームをレンダリングするための `UserServiceTabbedPane` のインスタンスにすることもできます。

特定のケースでは、コールバックは `UserServiceRawPane` を実装できます。このインターフェイスには制限がありますが、EBX® によって管理されていない HTML フォームを実装する場合に役立ちます。

ツールバー

ツールバーはオプションであり、2つのフレーバーがあります。

フォーム スタイル：

テーブルビュー スタイル：

スタイルは自動的に選択されます。レコードに定義されたツールバーはフォームスタイルであり、テーブルに定義されたツールバーはテーブルビュースタイルです。

例

次の例は、データの保存が成功した場合にのみ、現在のユーザーサービスを閉じ、ユーザーを現在の選択にリダイレクトするボタンを実装しています。

```
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 succeeded. Close the service.

    return UserServiceNext.nextClose();

```

```

    }

}

```

次の例は、Java 6 構文と互換性があります。前のコードとの違いのみが示されています。

```

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);

            }

        })
    )
    }
}

```

```

        }));

    }

}

```

次の例は、サービスを閉じて現在のユーザーを別のユーザーサービスにリダイレクトする URL を実装しています。

```

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);

}

}

```

次のコードは、メソッド `UserService.processEventOutcome` の実装であり、単純なユーザーサービスには十分です。

```

public class HelloWorldService implements UserService<...>

{

    @Override

    public UserServiceEventOutcome processEventOutcome(

        UserServiceProcessEventOutcomeContext<DatasetEntitySelection>
aContext,

        UserServiceEventOutcome anEventOutcome)

    {

        // By default do not modify the outcome.

        return anEventOutcome;

    }

}

```

次の例は、3つのステップを含む、より複雑な「ウィザード」サービスであり、各ステップには独自の `UserService.setupDisplay` メソッドがあります。

```

// 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;

}

}
```

データベースの更新

イベントコールバックはデータベースを更新する場合があります。

次の例は、1つのトランザクションを使用して2つのオブジェクトを保存します。

```

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();
    }
}

```

次の例は、[プロシージャ](#) を使用してデータベースを更新します。

```

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;

}
```

Ajax

ユーザーサービスは Ajax コールバックを実装できます。Ajax コールバックは、インターフェイス `UserServiceAjaxRequest` を実装する必要があります。

クライアントは、`UserServiceResourceLocator.getURLForAjaxRequest` によって生成された URL を使用して Ajax コールバックを呼び出します。

Ajax コンポーネントの使用を容易にするために、EBX® は、リクエストを送信して応答を処理するための JavaScript プロトタイプ `EBX_AJAXResponseHandler` を提供します。

`EBX_AJAXResponseHandler` の詳細については、`UserServiceAjaxRequest` を参照してください。

次の例は、部分的な HTML を返す Ajax コールバックを実装しています。

```
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>");

    }

}

```

REST データサービス

ユーザーサービスは、HTTP リクエストを介して REST データサービスにアクセスできます。

クライアントは、`UIResourceLocator.getURLForRest` によって生成された URL を使用する必要があります。この URL には、ユーザー認証に必要な情報が含まれています。

REST データサービスの詳細については、[ビルトイン RESTfulサービス](#) を参照してください。

次の例は、応答が `textarea` に出力される REST データサービス呼び出しを実装しています。

```

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("}");

    }

}

```

ファイルのアップロード

ユーザーサービスは、ファイル入力フィールドを使用してフォームを表示できます。

次の例は、タイトルとファイルの 2 つの入力フィールドを持つフォームを表示します。

```

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();

}

}
```

詳細については、`UIWidgetFileUpload` を参照してください。

ファイルのダウンロード

ユーザーサービスは、ファイルをダウンロードするための URL またはボタンを表示できます。ファイルの実際のダウンロードは、ユーザーサービスの制御下にあります。

次の例は、ファイルをダウンロードするための URL を表示します。

```
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();

    }

}

```

表示なしのユーザーサービス

ユーザーサービスは、表示せずにタスクを実行して前の画面に戻るか、ユーザーを別の画面にリダイレクトするように設計されている場合があります。

このタイプのサービスは、インターフェイス [UserServiceExtended](#) とメソッド `UserServiceExtended.initialize` を実装する必要があります。

次の例は、現在のテーブルビューで選択したレコードを削除します。

```

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;

    }

}

```

既知の制限事項

このようなサービスが Web コンポーネント、関連付け、パースペクティブアクション、または階層ノードのコンテキストで呼び出された場合、サービスは起動、初期化、および閉じられますが、サービスのターゲットエンティティは引き続き表示されます。

ユーザーサービスの宣言

- [宣言インターフェイス](#)
- [ライフサイクルとスレッドモデル](#)
- [登録](#)
- [サービスプロパティ](#)
- [サービス有効化範囲](#)
- [Web コンポーネントの宣言](#)

- ワークフローとパースペクティブでのパラメーターの宣言と可用性
- ユーザーサービス拡張
- ユーザーサービスグループ
 - ビルトインユーザーサービスグループ
 - ユーザーサービスグループの宣言
 - グループへのサービスの関連付け

宣言インターフェイス

次の表は、ユーザーサービスの宣言クラスが実装する必要があるインターフェイスをネイチャーごとに示しています。

ネイチャー	宣言 インターフェイス
Dataspace	UserServiceDeclaration.OnDataspace
Dataset	UserServiceDeclaration.OnDataset
TableView	UserServiceDeclaration.OnTableView
Record	UserServiceDeclaration.OnRecord
Hierarchy	UserServiceDeclaration.OnHierarchy
HierarchyNode	UserServiceDeclaration.OnHierarchyNode
Association	UserServiceDeclaration.OnAssociation
AssociationRecord	UserServiceDeclaration.OnAssociationRecord

ライフサイクルとスレッドモデル

ユーザーサービス宣言クラスは TIBCO EBX® の起動時にインスタンス化され、スレッドセーフになるようにコーディングする必要があります。ほとんどの実装は不変クラスであるため、これは通常問題にはなりません。

登録

ユーザーサービス宣言は、モジュールまたはデータモデルによって登録する必要があります。

モジュールによる登録は、次のようなコードによってモジュール登録サーブレットによって実行されます。

```
public class CustomRegistrationServlet extends ModuleRegistrationServlet

{

    @Override

    public void handleServiceRegistration(ModuleServiceRegistrationContext
aContext)

    {

        // Register custom user service declaration.

        aContext.registerUserService(new CustomServiceDeclaration());

    }

}
```

モジュール登録サーブレットの詳細については、[モジュール登録](#) および `ModuleRegistrationServlet` を参照してください。

データモデルによる登録は、次のようなコードによって実現されます。

```
public class CustomSchemaExtensions implements SchemaExtensions

{

    @Override

    public void defineExtensions(SchemaExtensionsContext aContext)

    {

        // Register custom user service declaration.

        aContext.registerUserService(new CustomServiceDeclaration());

    }

}
```



```
}
```

データモデル拡張の詳細については、[SchemaExtensions](#) を参照してください。

サービスプロパティ

ユーザーサービスのプロパティには、、、およびサービスを所有するが含まれます。すべてオプションですが、少なくともラベルを定義することをお勧めします。

詳細については、[UserServiceDeclaration.defineProperties](#) を参照してください。

サービス有効化範囲

有効化範囲は、どの選択でサービスが利用可能かを定義します。

サービス有効化範囲定義の例

```
public class CustomServiceDeclaration implements
UserServiceDeclaration.OnTableView

{

    ...

    @Override

    public void defineActivation(ActivationContextOnTableView aContext)

    {

        // activates the service in all dataspace 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());

    }

}

```

有効化範囲の宣言の詳細については、`UserServiceDeclaration.defineActivation` を参照してください。

ユーザーサービスの可用性の解決の詳細については、[サービスの権限の解決](#) を参照してください。

Web コンポーネント宣言

ワークフローとパースペクティブでのパラメーター宣言と可用性

ユーザーサービスは、サービスのネイチャーに応じて、一連のビルトインパラメーターを持つ Web コンポーネントとして自動的に使用可能になります。ワークフローユーザータスク、パースペクティブメニューアクション、またはツールバー Web コンポーネントアクションを構成するときにカスタムパラメーターを定義したり、サービス Web コンポーネントを使用可能として設定したりするには、`UserServiceDeclaration.declareWebComponent` を使用する必要があります。

Web コンポーネント宣言の例

```
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);

    }

}

```

詳細については、[TIBCO EBX® の Web コンポーネントとしての使用](#) を参照してください。

ユーザーサービス拡張

既存のユーザーサービス (ビルトインまたはカスタム) を拡張して、これらのサービスを Web コンポーネントとして使用するときに入力/出力パラメーターを追加することができます。

そのためには、まずユーザーサービス拡張をモジュールまたはデータモデルによって登録する必要があります。

モジュールによる登録は、次のようなコードでモジュール登録サーブレットによって実行されます。

```

public class CustomRegistrationServlet extends ModuleRegistrationServlet

{

    ...

    @Override

```

```

    public void handleServiceRegistration(ModuleServiceRegistrationContext
aContext)

    {

        // Register user service extension declaration.

        aContext.registerUserServiceExtension(new
ServiceExtensionDeclaration());

    }

}

```

モジュール登録サーブレットの詳細については、[モジュール登録](#) および `ModuleRegistrationServlet` を参照してください。

データモデルによる登録は、次のようなコードによって実現されます。

```

public class CustomSchemaExtensions implements SchemaExtensions

{

    ...

    @Override

    public void defineExtensions(SchemaExtensionsContext aContext)

    {

        // Register user service extension declaration.

        aContext.registerUserServiceExtension(new
ServiceExtensionDeclaration());

    }

}

```

データモデル拡張の詳細については、[SchemaExtensions](#) を参照してください。

ユーザーサービスグループ

ユーザーサービスグループは、メニューおよび権限管理画面でのユーザーサービスの表示を整理するために使用されます。

次のタイプのサービスグループを利用できます。

- EBX® が提供する [ビルトインユーザーサービスグループ](#)
- モジュールで宣言された [カスタムユーザーサービスグループ](#)

グループとサービス間のリンクは、サービスの宣言時に作成されます。[グループへのサービスの関連付け](#)を参照してください。

ビルトインユーザーサービスグループ

利用可能なビルトインサービスグループは、次のとおりです。

サービスグループキー	説明
<code>@ebx-importExport</code>	EBX® によって提供されるすべてのビルトインインポートおよびエクスポートサービスを含むグループ。デフォルトのメニューでは、これらのサービスは [インポート]/[エクスポート] サブメニューに表示されます。
<code>@ebx-views</code>	[表示] メニューに表示されるサービスを含むグループ。他のサービスグループとは異なり、このグループに関連付けられたサービスは、デフォルトのメニューには表示されず、テーブルトップツールバーのカスタマイズできない部分に表示される [表示] メニューにのみ表示されます。これらのサービスは、カスタムツールバーに手動で追加することもできます。

ユーザーサービスグループの宣言

ユーザーサービスグループは、モジュールを登録するときに、メソッド

`ModuleServiceRegistrationContext.registerServiceGroup` を使用して宣言する必要があります。

```
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..."));

    }

}
```

グループへのサービスの関連付け

サービスとグループの関連付けは、[declaration](#) で、メソッド

`UserServicePropertiesDefinitionContext.setGroup` を使用して行われます。

```
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);

    }

}
```

サービスは、ビルトインサービスグループまたはカスタムサービスグループのいずれかに関連付けることができます。後者の場合、このサービスは、このグループに属する他のビルトインサービスと同様に、このビルトイングループに表示されます。

開発の推奨事項

- [HTML](#)
 - [XHTML](#)
 - [iFrame](#)
- [CSS](#)

- パブリック CSS クラス
- 高度な CSS
- JavaScript
 - パブリック JS 関数
 - Java でのページ生成中の JavaScript の呼び出し
 - 高度な JavaScript

HTML

TIBCO EBX® のデフォルトスタイルをカスタムインターフェイスに適用できるように、特定の HTML スタイルとタグの包含を最小限に抑えることをお勧めします。API のアプローチは、開発者の実装プロセスを簡素化しながら、HTML ページのすべてのエレメントに標準化されたスタイルを自動的に適用することです。

XHTML

EBX® は、XHTML 1.0 Transitional で開発されたリッチインターネットアプリケーションです。つまり、HTML の構造は厳密な XML ファイルであり、"br" タグを含むすべてのタグを閉じる必要があります。この構造により、CSS ルールをより細かく制御でき、ブラウザのレンダリングの違いが少なくなります。

iFrame

EBX®では、特に `UIHttpManagerComponent` の URL と連携して iFrame を使用できます。技術的な理由から、JavaScript のみを使用して iFrame の `src` 属性を設定することをお勧めします。この方法では、ページが完全にレンダリングされ、すべてのビルトイン HTML コンポーネントが準備できた時点で iFrame が読み込まれます。

例

任意の `UIComponentWriter` から開発された次の例では、`UIHttpManagerComponent` を使用して iFrame の URL を構築し、正しい方法で設定します。

```
// 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("\"");");
```

CSS

パブリック CSS クラス

定数カタログ `UICSSClasses` は、ソフトウェアでコンポーネントのスタイルを設定するために使用される主要な CSS クラスを提供します。これらの CSS クラスは、背景色、境界線、管理内のカスタマイズ可能なテキストに従うため、ソフトウェアへの適切な長期的な統合が保証されます。フローティングマージンとパディングは、可変密度、アイコンのスタイルなどに応じて変動します。

以下も参照してください。:

- `UICSSUtils`

高度な CSS

EBX® では、すべてのページに 1 つ以上の外部カスケーディング スタイル シートを統合できます。これらの外部 CSS はリソースとして見なされ、[モジュール登録](#)で宣言する必要があります。

ソフトウェアを変更せずに CSS ルールとプロパティが適切に機能するようにするには、次の推奨事項に従う必要があります。これらのルールを守らないと、次のような結果を招く可能性があります。

- ソフトウェアが美観的にも機能的にも適切に動作しない：一部のデータの表示が失われるリスクがあり、一部の入力コンポーネントが機能しなくなる可能性があります
- ネイティブ CSS ルールが CSS 実装に影響するため、CSS ルールとプロパティが適切に機能しません

CSS 識別子とクラス名の予約済みプレフィックス

次のプレフィックスは、CSS #ids および .classes を作成するために使用しないでください。

ebx_	Internal built-in
yui	Yahoo User Interface global
ygtv	Yahoo User Interface tree view
layout-doc	Yahoo User Interface layout
cke_	CK editor (HTML エディターウィジェットで使用)
fa	Font Awesome (パースペクティブやツールバーで使用されるアイコン)

EBX® 内部で使用する CSS クラス

次の CSS クラスは、コンテキストセレクトーのないルールセットには含めないでください。

以下のいずれかの CSS クラスを使用して CSS セレクターにプレフィックスを付けないと、競合が発生し、EBX®の UI が破損します。

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

競合を避けるための例

避けるべき例

```
.selected {  
  
    background-color: red;  
  
}
```

正しい例

```
#myCustomComponent li.selected {

    background-color: red;

}
```

JavaScript

パブリック JS 関数

JavaScript 関数のカタログ `JavaScriptCatalog` には、JS ファイルで直接 (コピーと貼り付けで) 使用できる関数のリストが用意されています。

Java でのページ生成中の JavaScript の呼び出し

Java コンポーネントの HTML を生成するときに、API `UIJavaScriptWriter` を使用して特定の JavaScript コードを追加できます。

この JavaScript はページ全体がロードされると実行されます。 `UIBodyWriter.add` で記述された HTML エlement を即座に管理できます。実行コンテキストが on-load イベントの後に来るため、on-load 関数 (`window.onload = myFunctionToCallOnload;` など) の設定はサポートされていません。

高度な JavaScript

EBX® では、1 つ以上の外部 JavaScript ファイルを含めることができます。これらの外部 JavaScript ファイルはリソースと見なされ、[モジュール登録](#) で宣言する必要があります。パフォーマンス上の理由から、JavaScript リソースは必要な場合にのみ含めることをお勧めします (たとえば、ユーザーサービスや特定のフォーム内など)。API `UIDependencyRegisterer` を使用すると、開発者はコンテキストに応じて特定のページに JavaScript リソースを統合する条件を指定できます。

ソフトウェアを変更せずに JavaScript リソースが適切に機能するようにするには、次の推奨事項に従う必要があります。これらを守らないと、次のような結果を招く可能性があります。

- ソフトウェアの不適切な機能：ソフトウェアの機能またはグローバル変数が消去されると、一部の入力または表示コンポーネント (画面全体を含む) が機能しなくなる可能性があります
- グローバル変数または関数名が消去される可能性があるため、JavaScript 命令が適切に機能しません

予約済みの JS プレフィックス

次のプレフィックスは予約されており、変数、関数、メソッド、クラスなどの作成には使用しないでください。

ebx_	内部ビルトイン API
EBX_	内部ビルトイン API
YAHOO	Yahoo ユーザーインターフェイス API

SOAP データサービス

- [はじめに](#)
- [WSDL の生成](#)
- [SOAP 操作](#)

はじめに

- [概要](#)
- [アクティベーションと構成](#)
- [インタラクション](#)
 - [入力および出力メッセージのエンコード](#)
 - [追跡情報](#)
 - [セッションパラメーター](#)
 - [例外処理](#)
 - [JMS の使用](#)
- [セキュリティ](#)
 - [認証](#)
 - [グローバル許可](#)
 - [SOAP セキュリティヘッダーのオーバーライド](#)
 - [ルックアップメカニズム](#)
- [監視](#)
- [SOAP と REST の比較](#)

- 制限事項
 - 日付、時刻、日時の形式
 - SOAP 命名規則

概要

データサービスを使用すると、外部システムは、SOAP/Web サービス記述言語 (WSDL) 標準を使用して、TIBCO EBX® リポジトリで管理されているデータと対話できます。

SOAP 操作を呼び出すには、統合のユースケースで、データモデルから WSDL を生成する必要があります。次のような操作を実行できます。

- レコードの選択、挿入、更新、削除、またはカウント
- データセット値の選択
- データスペースまたはスナップショット間、または同じデータモデルに基づく 2 つのデータセット間のテーブルの違いの取得
- レコードの認証情報の取得

他の汎用 WSDL を生成して、次のような操作を実行できます。

- データスペースの作成、マージ、またはクローズ
- スナップショットの作成または終了
- データセット、データスペース、またはスナップショットの検証
- データワークフローの開始、再開、または終了
- UI またはシステム情報へのアクセスを管理するための管理操作

注意:

SOAP と REST の比較を参照してください。

アクティベーションと構成

データ サービスは、`ebx-dataservices` Web アプリケーションを他の EBX® モジュールとともに展開することで有効になります。詳細については、[Jakarta EE の展開](#) を参照してください。

リバースプロキシモードを使用するなど、特定の展開の場合、詳細については、[URL コンピューティング](#)を参照してください。

データサービスにアクセスするためのデフォルトの方法は HTTP 経由ですが、SOAP 操作に JMS を使用することもできます。詳細については、[JMS 構成](#)および [JMS の使用](#)を参照してください。

インタラクション

入力および出力メッセージのエンコード

UTF-8 では、すべての入力メッセージを *排他的*にする必要があります。すべての出力メッセージは UTF-8 です。

追跡情報

呼び出されるデータサービス操作によっては、セッション追跡情報を指定できる場合があります。

- SOAP 操作の例では、リクエストヘッダーには次のものが含まれます。

```
<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>
```

詳細については、Java API の `Session.getTrackingInfo` を参照してください。

セッションパラメーター

呼び出されるデータサービス操作に応じて、セッション入力パラメーターを指定できます。それらはリクエスト本文で定義されます。

入力パラメーターは、トリガー、アクセスルール、[カスタム](#) Web サービスなどのセッションオブジェクトを使用するカスタム Java コンポーネントで使用できます。これらは、データワークフロー操作でも使用できます。

- SOAP 操作の例では、オプションのリクエストヘッダーには次のものが含まれます。


```

<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>

```

詳細については、Java API の `Session.getInputParameterValue` を参照してください。

例外処理

以下の実行時に予期しないサーバーエラーが発生した場合

- SOAP 操作である SOAP 例外応答は、`soap:Fault` エレメントを介して呼び出し元に返されます。例を次に示します。

```

<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>

```

JMS の使用

HTTP の代わりに JMS を使用して SOAP 操作にアクセスすることが可能です。JMS アーキテクチャは、1 つの JMS リクエストキュー (必須)、1 つの JMS 障害キュー (オプション)、および JMS 応答キューに依存しています。[JMS 構成](#)を参照してください。必須のキューは入力キューです。リクエストメッセージは入力キューに配置する必要があり、応答メッセージは EBX® によって JMS リクエストの `replyTo` キューに配置されます。オプションのキューは、必要に応じて入力メッセージを再生できる障害キューです。構成ファイルでキューが設定およびアクティブ化され、リクエストメッセージの処理中に例外が発生した場合、この入力メッセージは障害キューにコピーされます。

リクエストと応答の関係は、JMS リクエストの `messageId` メッセージ識別子フィールドを応答の `correlId` 関連識別子フィールドにコピーすることによって作成されます。

生成された WSDL を特殊化するには、リネージ管理で JMS ロケーションポイントを定義する必要があります。特定のロケーションポイントが指定されていない場合、デフォルト値は `jms:queue:jms/EBX_QueueIn` になります。

セキュリティ

認証

コンテンツへのアクセスには認証が必須です。いくつかの認証方法が利用可能で、以下で説明されています。説明は優先度順に並べられています (EBX® は最も優先度の高い認証方式を最初に適用します)。

- 「基本認証スキーム」方式は、`RFC 2617 (基本認証スキーム)` で説明されているように、base64 エンコードの HTTP ヘッダー `Authorization` に基づいています。

ユーザーエージェントがユーザー ID「Alibaba」とパスワード「opensesame」を送信したい場合は、

次のヘッダーフィールドを使用します。

```
> Authorization: Basic QWxpYmFiYTpvGVCuIHNlc2FtZQ==
```

- 「ベアラー認証スキーム」方式は、サードパーティプロバイダーから発行される可能性のあるベアラー トークンで機能します。

```
> Authorization: bearer <accessToken>
```

この認証スキームの詳細については、[ベアラー認証の構成](#)を参照してください。

- 「標準認証スキーム」は、HTTP `リクエスト` に基づいています。ユーザーとパスワードはリクエストパラメーターから抽出されます。リクエストパラメーターの詳細については、[パラメーターセクション](#)を参照してください。

この認証スキームの詳細については、`Directory.authenticateUserFromLoginPassword` を参照してください。

- 「SOAP セキュリティヘッダー認証スキーム」メソッドは、[Web サービスセキュリティ UsernameToken プロファイル 1.0](#) 仕様に基づいています。

デフォルトでは、タイプ `PasswordText` がサポートされています。これは、WSDL で定義されている次の SOAP ヘッダーを使用して実行されます。

```

<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>

```

注意:

SOAP 操作でのみ使用できます。

- 「特定の認証スキーム」は、HTTP リクエストに基づいています。このメソッドの実装は、たとえば、HTTP リクエストからパスワードダイジェストまたはチケットを抽出できます。詳細については、`Directory.authenticateUserFromHttpRequest` を参照してください。
- 「SOAP 固有のヘッダー認証スキーム」。

詳細については、[SOAP セキュリティヘッダーのオーバーライド](#)を参照してください。

グローバル許可

グローバルアクセス許可は、SOAP および WSDL コネクタアクセスに対して個別に定義できます。

以下も参照してください。:

- [グローバル許可](#)

SOAP セキュリティヘッダーのオーバーライド

別のセキュリティ認証メカニズムを定義するために、デフォルトの WSS ヘッダーをオーバーライドすることができます。このようなオーバーライドは、HTTP と JMS の両方で考慮されます。定義およびオーバーライドするには、> の下の 構成設定を使用します。これには、次のフィールドが含まれます。

スキーマの場所	WSDL にインポートするセキュリティ XML スキーマの URI
ターゲット名前空間	スキーマ内のエレメントのターゲット名前空間
名前空間のプレフィックス	ターゲット名前空間のプレフィックス
メッセージ名	WSDL で使用するメッセージ名
ルートエレメント名	セキュリティヘッダーのルートエレメント名。名前は、スキーマで宣言されている名前と同じにする必要があります。
wsdl:part エレメント名	メッセージの <code>wsdl:part</code> の名前

デフォルトのセキュリティヘッダーをオーバーライドする目的は、セキュリティヘッダーに一致する WSDL メッセージの宣言を変更して、次の内容が含まれるようにすることです。

```
<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>

```

対応する XML スキーマヘッダー宣言は次のようになります。

```

<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>

```

上記の XML スキーマと構成を使用する SOAP メッセージには、次のヘッダーがあります。

```

<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>

```

このデフォルト以外のヘッダーを処理するには、メソッド

`Directory.authenticateUserFromSOAPHeader` を実装する必要があります。

注意:

SOAP 操作でのみ使用できます。

ルックアップメカニズム

EBX® はいくつかの認証方法を提供するため、条件に基づくルックアップメカニズムが設定され、特定のリクエストにどの方法を適用する必要があるかが分かります。メソッドの適用条件は、認証方式の優先度に従って評価されます。条件が満たされない場合、サーバーは次のメソッドを評価します。次の表に、サポートされている各プロトコルで使用可能な認証方法とその適用条件を示します。それらは、優先度の高いものから低いものの順に並べられます。

操作/プロトコル	認証方法と適用条件
SOAP / JMS	SOAP セキュリティヘッダー - SOAP リクエストは、JMS プロトコルを介して受信されます - SOAP ヘッダーコンテンツには、Security エlementが含まれている必要があります SOAP 固有のヘッダー - SOAP リクエストは、JMS プロトコルを介して受信されます - SOAP ヘッダーコンテンツには、Security Elementを含めることはできません
SOAP / HTTP	基本 - HTTP リクエストは Authorization ヘッダーを保持する必要があります Authorization ヘッダー値は Basic という単語で始まる必要があります - URL パラメーターに login は指定されていません

操作/プロトコル	認証方法と適用条件
	ベアラー • HTTP リクエストは <code>Authorization</code> ヘッダーを保持する必要があります • <code>Authorization</code> ヘッダー値は <code>Bearer</code> という単語で始まる必要があります • URL パラメーターに <code>login</code> は指定されていません 標準 • HTTP リクエストは <code>Authorization</code> ヘッダーを保持できません • <code>login</code> と <code>password</code> は URL パラメーターで提供されます SOAP セキュリティヘッダー • SOAP ヘッダーコンテンツには、<code>Security</code> エlementが含まれている必要があります • HTTP リクエストは <code>Authorization</code> ヘッダーを保持できません • URL パラメーターに <code>login</code> は指定されていません 特定 • HTTP リクエストを、以前の認証方法の条件を満たすものにすることはできません SOAP 固有のヘッダー • SOAP ヘッダーコンテンツには、<code>Security</code> エlementを含めることはできません

操作/プロトコル	認証方法と適用条件
	• HTTP リクエストは <code>Authorization</code> ヘッダーを保持できません • URL パラメーターに <code>login</code> は指定されていません
WSDL / HTTP	基本 • HTTP リクエストは <code>Authorization</code> ヘッダーを保持できません • <code>Authorization</code> ヘッダー値は <code>Basic</code> という単語で始まる必要があります • URL パラメーターに <code>login</code> は指定されていません 標準 • HTTP リクエストは <code>Authorization</code> ヘッダーを保持できません • <code>login</code> と <code>password</code> は URL パラメーターで提供されます 特定 • HTTP リクエストを、以前の認証方法の条件を満たすものにすることはできません

同じリクエストに複数の認証方法が存在する場合、EBX® は HTTP コード `401 Unauthorized` を返します。

監視

データサービスイベントは、EBX® メイン構成ファイルで宣言されているログカテゴリ `ebx.dataServices` を介して監視できます。たとえば、`ebx.log4j.category.log.dataServices= INFO`、`ebxFile:dataservices` です。

以下も参照してください。:

- [EBX® ログの構成](#)
- [TIBCO EBX® メイン構成ファイル](#)

SOAP と REST の比較

操作	SOAP	REST
データ		
メタモデルを選択		X
レコードを選択またはカウント (フィルターおよび/またはパブリケーションの表示を使用)	X	X
関連値を持つレコードを選択 (第 1 レベル)	X	
ビジネスオブジェクトの選択		X
レコードを更新または削除 (フィルターおよび/またはビューのパブリケーションを使用)		X
可能な列挙値のセクター (フィルター付き)		X
作成または複製の準備		X
レコードの挿入、更新、または削除	X	X
履歴レコードを選択またはカウント (フィルターおよび/またはパブリケーションの表示を使用)		X

操作	SOAP	REST
データセットからノード値を選択	X	X
データセットからノード値を更新		X
データスペースまたはスナップショット間のテーブルまたはデータセットの変更を取得	X	
レプリケーションユニットを更新	X	
レコードの認証情報を取得	X	
サービス契約を生成	WSDL	OpenAPI
ビュー		
公開されたテーブルビューを検索		X
データセット		
データセット情報を選択		X
ルートまたは子データセットを選択		X
データスペース		
データスペースまたはスナップショット情報を選択		X
ルートまたは子のデータスペースを選択、またはスナップショットを選択		X
データスペースを作成、閉じる、マージ	X	X

操作	SOAP	REST
スナップショットを作成、閉じる	X	X
データスペースまたはスナップショットを検証	X	
データセットの検証	X	
データスペースのロック、ロック解除	X	X
ワークフロー		
ワークフローの開始、再開、または終了	X	
管理		
デフォルトのディレクトリコンテンツ「ユーザー」、「ロール」... テーブルの管理	X	X
ユーザーインターフェイスを開く、閉じる	X	X
管理データセットを選択、挿入、更新、削除操作		X
システム情報を選択	X	X
ステージング API		X
ヘルス操作		X
その他		
Java API からカスタムサービスを開発 (*)		X
カスタムサービス契約の生成 (*)		OpenAPI

(*) 詳細については、[REST Toolkit](#) を参照してください。

制限事項

日付、時刻、日時の形式

データサービスは、次の日付と時刻の形式のみをサポートします。

タイプ	形式	例
xs:date	yyyy-MM-dd	2007-12-31
xs:time	HH:mm:ss または HH:mm:ss.SSS	11:55:00
xs:dateTime	yyyy-MM-ddTHH:mm:ss また は yyyy-MM- ddTHH:mm:ss.SSS	2007-12-31T11:55:00

SOAP 命名規則

データサービス操作の命名規則により、データモデル内で定義された各テーブルには、WSDL 生成の一意の名前が必要です。

WSDL の生成

- サポートされている標準
- 操作の種類
- WSDL のダウンロード方法
 - ユーザーインターフェイスからの WSDL のダウンロード
 - HTTP プロトコルからの WSDL のダウンロード
- HTTP の例

サポートされている標準

TIBCO EBX®は、[W3C Webサービス記述言語 1.1](#) 標準に準拠する WSDL を生成します。

操作の種類

WSDL は、さまざまな種類の操作に対して生成できます。

操作の種類	WSDL の説明
custom	EBX® アドオンの WSDL。
dataset	データセットおよびレプリケーション操作の WSDL。
directory	デフォルト EBX® ディレクトリ操作の WSDL。 <code>tablePaths</code> または <code>operations</code> パラメーターを使用してデータをフィルタリングすることもできます。
repository	データスペースまたはスナップショット管理操作の WSDL。
tables	特定のデータセットのテーブルに対する操作の WSDL。
userInterface	バージョン 5.8.1 以降非推奨。この操作タイプは、 <code>administration</code> に置き換えられました。下位互換性の理由から、ユーザーインターフェイス管理操作は引き続き使用できますが、このタイプは使用しないことをお勧めします。 ユーザーインターフェイス管理操作の WSDL (これらの操作には管理者のみがアクセスできます)。
administration	次のような管理操作の WSDL • ユーザーインターフェイス管理 • システム情報検索 これらの操作には、管理者のみがアクセスできます。
workflow	EBX® ワークフロー管理操作の WSDL。

WSDL のダウンロード方法

EBX® は次の方法をサポートしています。

- ユーザーインターフェイスから
- HTTP プロトコルから

WSDL は、許可されたプロファイルによってのみダウンロードできます。

操作の種類	アクセス権限
custom	少なくとも 1 つの Web サービスが登録されている場合は、すべてのプロファイル。
dataset	すべてのプロファイル。
directory	次の条件が有効な場合のすべてのプロファイル • 特定のディレクトリ実装が使用されていない場合 (ビルトインの管理者ロールのみが、この条件の対象になります)。 • 管理用にグローバルアクセス許可が定義されている場合。 • 「directory」データセットのアクセス権限が、現在のプロファイルへの書き込み許可を所有する場合。
repository	すべてのプロファイル。
tables	すべてのプロファイル。
userInterface	バージョン 5.8.1 以降非推奨。この操作タイプは、administration に置き換えられました。下位互換性の理由から、ユーザーインターフェイス管理操作は引き続き使用できますが、このタイプは使用しないことをお勧めします。

操作の種類	アクセス権限
	すべての条件が有効な場合は、ビルトインの管理者ロールまたは委任された管理者プロファイル • 管理用にグローバルアクセス許可が定義されている場合。 • 「ユーザーインターフェイス」データセットアクセス権限が、現在のプロファイルへの書き込み権限を所有する場合。
administration	すべての条件が有効な場合は、ビルトインの管理者ロールまたは委任された管理者プロファイル • 管理用にグローバルアクセス許可が定義されている場合。 • [管理] データセットアクセス権限が、現在のプロファイルへの書き込み権限を所有する場合。
workflow	すべてのプロファイル。

ユーザーインターフェイスからの WSDL のダウンロード

許可されたユーザーは、データサービス管理領域から EBX® WSDL をダウンロードできます。

注意:

詳細については、『ユーザーガイドの』 [データスペース操作の WSDL の生成](#)を参照してください。

HTTP プロトコルからの WSDL ダウンロード

アプリケーションは、`GET` または `POST` HTTPメソッドを使用して、EBX® WSDL をダウンロードできます。アプリケーションは、適切な権限を持つプロファイルを使用して認証される必要があります。

URL フォーマット

```
http[s]://<host>[:<port>]/<ebx-dataservices>/<type>[/<dataspace>[/<dataset>]]?<queryParameters>
```

説明

- `<ebx-dataservices>` は、'ebx-dataservices.war' Web アプリケーションのパスに対応します。このパスは、複数の URI セグメント (またはなし) と Web アプリケーションの名前の順序で構成されます。
- `<type>` は、[操作タイプ](#)に対応します。
- `<dataspace>` は、データスペースまたはスナップショット識別子に対応します。
- `<dataset>` は、データセット名に対応します。
- `<queryParameters>` は、URL で渡される共通または専用の操作パラメーターに対応します。

パラメーター

リクエストパラメーターは、次のいずれかの方法で指定できます。

- URL からのパスセグメントに対応する `PathParam` (推奨)
- 値を持つ標準の HTTP パラメーターに対応する `QueryParam`

パラメーター名	<code>PATHPARAM</code>	<code>QUERYPARAM</code>	必須	説明
<code>WSDL</code>	いいえ	はい	はい	WSDL のダウンロード操作を指定します。 空の値。

パラメーター 名	PATHPARAM	QUERYPARAM	必須	説明
login	いいえ	はい	いいえ	ユーザー ID を指定します。 標準の認証方法を使用する場合に必要です。 文字列 型の値。
password	いいえ	はい	いいえ	ユーザーパスワードを指定します。 標準の認証方法を使用する場合に必要です。 文字列 型の値。
type	はい	いいえ	はい	操作タイプを指定します。 custom 、 dataset 、 directory 、 administration 、 userInterface 、 repository 、 tables 、 workflow の

パラメーター 名	PATHPARAM	QUERYPARAM	必須	説明
				いずれかの値を指定します。 <code>文字列</code> 型の値。
<code>branch</code> <code>version</code>	はい	はい	(*)	データスペースまたはスナップショット識別子を指定します。 (*) <code>tables</code> および <code>dataset</code> タイプに必要です。それ以外の場合は無視されます。 <code>文字列</code> 型の値。
<code>instance</code>	はい	はい	(*)	データセット名を指定します。 <code>文字列</code> 型の値。

パラメーター 名	PATHPARAM	QUERYPARAM	必須	説明
<code>tablePaths</code>	いいえ	はい	いいえ	テーブルパスのリストを指定します。 <code>tables</code> または <code>directory</code> タイプの場合はオプション。それ以外の場合は無視されます。 定義されていない場合、すべてのテーブルが選択されます。 各テーブルパスはカンマで区切られます。 <code>文字列</code> 型の値。
<code>operations</code>	いいえ	はい	いいえ	操作のサブセットの WSDL を生成できます。 <code>tables</code> または <code>directory</code> の操作タイプ

パラメーター 名	PATHPARAM	QUERYPARAM	必須	説明
				ではオプションです。それ以外の場合は無視されます。定義されていない場合、指定されたタイプのすべての操作が生成されます。 このパラメーターの値は、次の 1 つ以上の文字を連結したものです。 • C = レコードのカウント • D = レコードの削除 • E = 資格情報の取得 • G = 変更の取得 • I = レコードの挿入 • U = レコードの更新

パラメーター 名	PATHPARAM	QUERYPARAM	必須	説明
				• R = 読み取り操作 (CEGS と同等) • S = レコードの選択 • W = 書き込み操作 (DIU と同等) 文字列型の値。
namespaceURI	はい	はい	(**)	カスタム Web サービスの一意の名前空間 URI を指定します。 (**) type パラメーターが custom に設定されている場合は必須であり、それ以外の場合は無視されます。 URI タイプの値。

パラメーター 名	PATHPARAM	QUERYPARAM	必須	説明
attachmentFilename	いいえ	はい	(***)	添付ファイル名を指定します。 (***) isContentInAttachment パラメーターが定義され、true に設定されている場合はオプション、それ以外の場合は無視されます。 文字列 型の値。
isContentInAttachment	いいえ	はい	いいえ	WSDL を添付ファイルとしてダウンロードするかどうかを指定します。 ブール値 型の値。 デフォルト値は false です。
targetNamespace	いいえ	はい	いいえ	WSDL のターゲット名前空間 URI をオー

パラメーター名	PATHPARAM	QUERYPARAM	必須	説明
				バーライドします。 <code>URI</code> タイプの値。 デフォルト値は <code>urn:ebx:ebx-dataservices</code> です。

メッセージ本文

メッセージ本文は必要ありません。

HTTP コード

HTTP コードが常に返されます。エラーは 400 を超えるコードで示されます。

ステータスコード	情報
<code>200</code> (OK)	WSDL コンテンツは正常に生成され、リクエストによって返されます (オプションで添付ファイル)。
<code>400</code> (Bad request)	リクエストが正しくありません。これは次の場合に発生します。 - リクエストエレメントが正しくありません。 - テーブル名の単一性チェックに、少なくとも 1 つのエラーが含まれています。 注意:

ステータスコード	情報
	詳細については、 WSDL およびテーブル操作 を参照してください。
401 (Unauthorized)	リクエストには認証されたユーザーが必要です。
403 (Forbidden)	認証されたユーザーに対する要求は許可されていません。
405 (メソッドは許可されていません)	この構成ではリクエストは許可されていません。
500 (Internal error)	リクエストはエラーを生成します (スタックトレースと詳細なエラーメッセージが返されます)。

応答本文

応答本文は、返されたステータスコードと要求された WSDL コンテンツによって異なります。

- 200 (OK) : HTTP ヘッダー `Content-Type` が `text/xml; charset=UTF-8` に設定されています。

コンテンツが添付ファイルの場合、HTTP ヘッダー `Content-Disposition` は `attachment; filename*=UTF-8''<filename.wsdl>` に設定されます。

- 4xx : 詳細なメッセージが本文に返されます。HTTPヘッダー `Content-Type` は `text/html; charset=utf-8` に設定されます。

HTTP の例

次の例の一部は、`PathParam` と `QueryParam` の 2 つのメソッドで表示されます。

- WSDL には、標準の認証方法を使用したすべてのリポジトリ操作が含まれます。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/repository?
WSDL&login=<login>&password=<password>
```

- WSDL には、すべてのワークフロー操作が含まれます。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/workflow?WSDL
```

- WSDL には、'dataspace1' データスペース内の 'dataset1' データセットのすべてのテーブル操作が含まれます。

```
PathParam
```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables/dataspace1/dataset1?WSDL
```

```
QueryParam
```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables?
```

```
WSDL&branch=<dataspace1>&instance=<dataset1>
```

- WSDL には、'dataspace1' データスペースの 'dataset1' データセットに対して選択された 2 つのテーブル操作が含まれます。

```
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
```

- WSDL には、'dataspace1' データスペースの 'dataset1' データセットに対して選択された 2 つのテーブル操作が含まれます。

```
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
```

- WSDL には、専用 URI のカスタム Web サービス操作が含まれます。

```
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`

SOAP操作

- データモデルから生成された操作
 - コンテンツポリシー
 - 一般的なリクエストパラメータ
 - 操作の選択
 - 削除操作
 - カウント操作
 - 更新操作
 - 挿入操作
 - 変更操作を取得
 - クレデンシャルの取得操作
 - 複数の連鎖操作
 - オプティミスティックロック
- データセットとデータスペースの操作
 - データスペースを検証する
 - データセットを検証する
 - データスペースを作成する
 - スナップショットを作成する
 - データスペースのロック
 - データスペースのロックを解除する
 - データスペースをマージする
 - データスペースまたはスナップショットを閉じる
 - レプリケーションの更新
- データワークフローの操作
 - ワークフローを開始する

- ワークフローを再開する
- ワークフローを終了する
- 管理サービス
 - ディレクトリ サービス
 - ユーザー インターフェースの操作
 - システム情報操作

データモデルから生成された操作

TIBCO EBX®リポジトリで使用されるデータモデルの場合、その操作を定義する対応するWSDLを動的に生成できます。このWSDLを使用すると、EBX®リポジトリの読み取り/書き込みが可能になります。たとえば、パス `/root/XX/exampleTable` にあるテーブルの場合、生成されるリクエストは基になるデータモデルの構造に従い、テーブル名 `<m:{operation}_exampleTable xmlns:m="urn:ebx-schemas:dataservices_1.0">` を含みます。

注意:

WSDLおよびSOAP操作はデータモデル構造に密接に依存しているため、データモデルを変更した後は、最新のWSDLを再配布することが重要です。

コンテンツポリシー

レコードのコンテンツへのアクセス、XML 要素の有無は、認証されたユーザー セッションの [解決された権限](#) によって異なります。以下に詳述する追加の側面がコンテンツに影響を与える可能性があります。

データモデルからのフィールドの無効化

データモデルで定義されている `hiddenInDataServices` プロパティを使用すると、ユーザープロフィールに関係なく、データサービスのフィールドを常に非表示にできます。このパラメーターは、生成されたWSDLに影響を与えます。非表示のフィールドまたはグループは、要求および応答の構造に含まれません。

`hiddenInDataServices` パラメータ値を変更すると、以前のWSDLを引き続き使用するクライアントに次の影響があります。

- リクエストに応じて、データモデルプロパティが `true` に変更され、関連するフィールドが WSDL リクエストに存在する場合、例外がスローされます。
- 応答時に、スキーマプロパティが `false` に変更されている場合、WSDL 検証がアクティブ化されると、エラーが返されます。

この「デフォルトビュー」の設定は、データモデル内で定義されています。

も参照してください:

- データ サービスでフィールドを非表示にする
- 権限

関連付けフィールド

テーブルレコードへの読み取りアクセスにより、UIManager に表示される関連付けフィールドをエクスポートできます。この機能は、「hiddenInDataServices」モデルパラメーターと組み合わせることができます。

注:

制限事項：変更および更新操作では、関連付けフィールドは管理されません。また、select 操作は、第1レベルの関連付け要素のみをエクスポートします（関連付けられたオブジェクトのコンテンツに関連付け要素を含めることはできません）。

一般的なリクエストパラメータ

いくつかのパラメータはいくつかの操作に共通であり、以下で詳しく説明します。

要素	説明	必須
ブランチ	データセットが属するデータスペースの識別子。	このパラメーターまたは「バージョン」パラメーターのいずれかを定義する必要があります。'挿入'、'更新'、および '削除' 操作に必要です。

要素	説明	必須
version	データセットが属するスナップショットの識別子。	このパラメーターまたは「ブランチ」パラメーターのいずれかを定義する必要があります
実例	クエリするテーブルを含むデータセットの一意の名前。	はい
述語	XPath述語は、リクエストが適用されるレコードを定義します。空の場合、このフィールドが必須である 「削除」操作を除いて、すべてのレコードが取得されます。 も参照してください: <code>includesMetadata</code> パラメーター	削除操作にのみ必要
データ	データモデルの構造内で表される、変更されるレコードが含まれます。操作全体は、XMLインポートと同等です。データコンテンツに対して実行される操作の詳細は、 セクションインポート で指定されています。	挿入および更新操作にのみ必要です
viewPublication	このパラメーターは、論理的な <code>AND</code> 操作として <code>predicate</code> パラメーターと組	番号

要素	説明	必須
	み合わせることができません。 このパラメーターの動作については、Web コンポーネントとしての EBX® セクションに説明が記載されています。 「viewId」パラメーターが使用されている場合は使用できず、階層ビューまたはタイルビューでは使用できません。	
viewId	バージョン5.2.3以降非推奨。このパラメーターは、パラメーター 'viewPublication' に置き換えられました。下位互換性のために利用可能なままですが、将来のバージョンで最終的に削除される予定です。 'viewPublication' パラメーターが使用されている場合、このパラメーターは使用できません。	番号
blockingConstraintsDisabled	このプロパティは、すべてのテーブル更新データサービス操作で使用できます。 <code>true</code> の場合、検証プロセスはデータモデルで定義さ	番号

要素	説明	必須
	れたブロッキング制約を無効にします。 このパラメータが存在しない場合、デフォルトは <code>false</code> です。 詳細については、ブロッキング制約と非ブロッキング制約を参照してください。	
詳細	<code>details</code> 要素は、次のオプションを指定します。 オプションの属性 <code>locale</code> (デフォルトは 'en-US') は、検証メッセージを返す必要がある <code>blockingConstraintsDisabled</code> パラメーターの言語を定義します。	番号
最後のブロードキャストへのリダイレクトを無効にする	このプロパティは、すべてのデータ サービス操作で使用できます。 <code>true</code> の場合、D3 プライマリノード上の配信データスペースへのアクセスは、最後のブロードキャスト スナップショットにリダイレクトされません。それ以外の場合、そのようなデータスペースへのアクセスは常に最後のスナップショットブロードキャストにリダイレクトされます。	番号

要素	説明	必須
	このパラメータが存在しない場合は、構成プロパティ <code>ebx.dataservices.disableRedirectionToLastBroadcast.default</code> が設定されていない限り、デフォルトは <code>false</code> (D3 マスターでのリダイレクトが有効)になります。 指定されたデータスペースが D3 プライマリ ノード上の配信データスペースでない場合、このパラメータは無視されます。	

操作の選択

テーブルでリクエストを選択

```
<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}>

```

と：

要素	説明	必須
ブランチ	共通パラメータの説明を参照してください。	
version	共通パラメータの説明を参照してください。	
実例	共通パラメータの説明を参照してください。	
述語	共通パラメータの説明を参照してください。 このパラメーターは、論理的な AND 操作として viewPublication パラメーターと組み合わせることができます。	
viewPublication	共通パラメータの説明を参照してください。	
includesMetadata	応答には、指定されたメタデータが含まれています。 返された各レコードには、ebx-metadata ルートエレメントの下にステップと呼	番号

要素	説明	必須
	ばれる追加エレメントが含まれています。 たとえば、「system」ステップ値には技術データが含まれます。楽観ロックセクションも参照してください。 <pre> <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> </pre> 文字列値のステップ (デフォルト値は空) はカンマで区切る必要があります (system, teamUp)。 <code>ebx-all</code> 値を使用して、すべてのステップを返すことができます。	
includesTechnicalData	バージョン 6.1.0 以降非推奨。 <code>includesMetadata</code> リクエストパラメーターに置き換えられました。 <code>true</code> の場合、レスポンスには技術データが含まれま	番号

要素	説明	必須
	す。楽観ロックセクションも参照してください。 返される各レコードには、この技術情報の追加の属性が含まれます。次に例を示します。 <pre>...<table ebxd:lastTime="2010-06-28T10:10:31.046" ebxd:lastUser="Uadmin ebxd:uuid="9E7D0530-828C-11DF-B733-0012D01B6E76">... .</pre>	
exportCredentials	<code>true</code> の場合、selectは各レコードのクレデンシャルも返します。	番号
ページ付け	ページネーション構成を指定します。以下の子要素を参照してください。	番号
pageSize (<code>pagination</code> 要素の下にネストされています)	1ページあたりのレコードの最大数を指定します。 <code>整数</code> タイプ。デフォルト値は10です。 も参照してください: <code>ebx.dataservices.pagination.pageSize.default</code> 値 0 は、推奨される最大ページサイズを選択します。デフォルトは 10000 です。 も参照してください:	番号

要素	説明	必須
	<code>ebx.dataservices.pagination.maxSize.default</code> 注: 指定またはデフォルトのページサイズ値が推奨される最大ページサイズを超える場合は、最大ページサイズが使用されます。	
previousPageLastRecordPredicate (ページ付け 要素の下にネストされています)	ページ付けが有効になっている場合、ページをフェッチする必要があるレコードを定義するXPath述語。この値は、前の応答によって要素 <code>lastRecordPredicate</code> として提供されます。渡されたレコードが見つからない場合は、最初のページが返されます。	番号
最後のブロードキャストへのリダイレクトを無効にする	共通パラメータの説明を参照してください。	

テーブルで応答を選択します

```
<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>

```

と：

要素	説明
データ	テーブルパスの後に表示されるレコードのコンテンツ。

要素	説明
資格情報	各レコードの各ノードのアクセス権が含まれています。
lastRecordPredicate	ページネーションが有効になっている場合にのみ返されます。これは、要素 <code>previousPageLastRecordPredicated</code> の次のリクエストで使用される最後のレコードを定義します。

オブティミスティックロックセクションも参照してください。

データセットでリクエストを選択

この操作は、テーブルなしのデータセットコンテンツを返します。

```
<m:selectInstance xmlns:m="urn:ebx-schemas:dataservices_1.0">

  <branch>String</branch>

  <version>String</version>

  <instance>String</instance>

</m:selectInstance>
```

と：

要素	説明	必須
ブランチ	共通パラメータの説明を参照してください。	
version	共通パラメータの説明を参照してください。	
実例	共通パラメータの説明を参照してください。	
最後のブロードキャストへのリダイレクトを無効にする	共通パラメータの説明を参照してください。	

データセットの応答を選択

```
<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>
```

と：

要素	説明
データ	テーブルのないデータセットコンテンツ。

削除操作

レコードを削除するか、子データセットの場合は、レコードのコンテキストに応じてレコードの状態を「隠蔽」または「継承」として定義します。レコードは、 `predicate` パラメーターによって選択されます。

リクエストの削除

```
<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}>
```

と：

要素	説明	必須
ブランチ	共通パラメータの説明を参照してください。	
実例	共通パラメータの説明を参照してください。	
述語	共通パラメータの説明を参照してください。	
インクルードオカルティング	オカルティングモードのレコードが含まれます。デフォルト値は <code>false</code> です。	番号

要素	説明	必須
継承IfInOccultingMode	オカルティングモードの場合、親からレコードを継承します。デフォルト値は <code>false</code> です。	番号
occultIfInherit	バージョン5.7.0以降非推奨レコードが継承モードの場合、レコードを非推奨にします。デフォルト値は <code>false</code> です。	番号
checkNotChangedSinceLastTime	タイムスタンプは、最後の読み取り以降にレコードが変更されていないことを確認するために使用されます。 オプティミスティックロックセクション も参照してください。	番号
blockingConstraintsDisabled	共通パラメータ の説明を参照してください。	
詳細	共通パラメータ の説明を参照してください。	
最後のブロードキャストへのリダイレクトを無効にする	共通パラメータ の説明を参照してください。	

応答を削除

提供されたパラメーターの1つが不正である場合、必須パラメーターが欠落している場合、アクションが許可されていない場合、またはレコードが選択されていない場合は、例外が返されます。それ以外の場合は、特定の応答が返されます。

```
<ns1:delete_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
```

```
    <status>String</status>
```

```
    <blockingConstraintMessage>String</blockingConstraintMessage>
```

```
</ns1:delete_{TableName}Response>
```

と：

要素	説明
status	「00」は、操作が正常に実行されたことを示します。「95」は、少なくとも1つの操作がブロック制約に違反し、その結果、操作全体が中止されたことを示します。
blockingConstraintMessage	この要素は、ローカライズされたメッセージでステータスが「95」に等しい場合に存在します。メッセージのロケールは、要求パラメーターまたはユーザーセッションから取得されます。

カウント操作

カウントリクエスト

```
<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}>
```

と：

要素	説明
ブランチ	共通パラメータの説明を参照してください。
version	共通パラメータの説明を参照してください。
実例	共通パラメータの説明を参照してください。
述語	共通パラメータの説明を参照してください。
最後のブロードキャストへのリダイレクトを無効にする	共通パラメータの説明を参照してください。

応答をカウント

```
<ns1:count_{TableName}Response xmlns:ns1="urn:ebx-
schemas:dataservices_1.0">

    <count>Integer</count>

</ns1:count_{TableName}Response>
```

と：

要素	説明
count	要求内の述部に対応するレコードの数。

更新操作

更新リクエスト

```
<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}>

```

と：

要素	説明	必須
ブランチ	共通パラメータの説明を参照してください。	
実例	共通パラメータの説明を参照してください。	

要素	説明	必須
updateOrInsert	<code>true</code> でレコードが現在存在しない場合、操作によってレコードが作成されます。 <code>boolean</code> タイプ。デフォルト値は <code>false</code> です。	番号
byDelta	<code>true</code> であり、受信メッセージに要素が現在存在しない場合、ターゲット値は変更されません。 <code>false</code> で、ノードが <code>hiddenInDataServices</code> と宣言されている場合、ターゲット値は変更されません。完全な動作については、 挿入および更新操作のセクション で説明しています。	番号
blockingConstraintsDisabled	共通パラメータ の説明を参照してください。	
詳細	共通パラメータ の説明を参照してください。	
最後のブロードキャストへのリダイレクトを無効にする	共通パラメータ の説明を参照してください。	
データ	共通パラメータ の説明を参照してください。	

も参照してください:

- [オプティミスティックロック](#)

応答を更新

```
<ns1:update_{TableName}Response xmlns:ns1="urn:ebx-
schemas:dataservices_1.0">

  <status>String</status>

  <blockingConstraintMessage>String</blockingConstraintMessage>

</ns1:update_{TableName}Response>
```

と：

要素	説明
status	「00」は、操作が正常に実行されたことを示します。「95」は、少なくとも1つの操作がブロック制約に違反し、その結果、操作全体が中止されたことを示します。
blockingConstraintMessage	この要素は、ローカライズされたメッセージでステータスが「95」に等しい場合に存在します。メッセージのロケールは、要求パラメータまたはユーザーセッションから取得されます。

挿入操作

リクエストを挿入

```
<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}>
```

と :

要素	説明	必須
ブランチ	共通パラメータの説明を参照してください。	
実例	共通パラメータの説明を参照してください。	
byDelta	<code>true</code> であり、受信メッセージに要素が現在存在しない場合、ターゲット値は変更されません。 <code>false</code> で、ノードが	番号

要素	説明	必須
	<code>hiddenInDataServices</code> と宣言されている場合、ターゲット値は変更されません。完全な動作については、 挿入および更新操作のセクション で説明しています。	
<code>blockingConstraintsDisabled</code>	共通パラメータ の説明を参照してください。	
詳細	共通パラメータ の説明を参照してください。	
最後のブロードキャストへのリダイレクトを無効にする	共通パラメータ の説明を参照してください。	
データ	共通パラメータ の説明を参照してください。	

応答を挿入

```
<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>
```

と：

要素	説明
status	「00」は、操作が正常に実行されたことを示します。「95」は、少なくとも1つの操作がブロック制約に違反し、その結果、操作全体が中止されたことを示します。
blockingConstraintMessage	この要素は、ローカライズされたメッセージでステータスが「95」に等しい場合に存在します。メッセージのロケールは、要求パラメータまたはユーザーセッションから取得されます。
述語	挿入されたレコードの主キーに一致する述部。複数のレコードが挿入されると、述部は入力メッセージ内のレコードの宣言順序に従います。

変更操作を取得する

[コンテンツポリシー](#)に従って変更を返します。

変更リクエストを取得する

2つのデータセット間の変更：

```
<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}>

```

2つのテーブル間の変更：

```

<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}>

```

と：

要素	説明	必須
ブランチ	共通パラメータの説明を参照してください。	
version	共通パラメータの説明を参照してください。	

要素	説明	必須
実例	共通パラメータの説明を参照してください。	
compareWithBranch	比較するデータスペースの識別子。	このパラメーターまたは「compareWithVersion」パラメーターのいずれかを定義する必要があります。
compareWithVersion	比較するスナップショットの識別子。	このパラメーターまたは「compareWithBranch」パラメーターのいずれかを定義する必要があります。
compareWithInstance	比較するデータセットの識別子。未定義の場合は、instanceパラメータが使用されます。	番号
resolveMode	差が解決モードで計算されるかどうかを定義します。デフォルトは true です。詳細については、解決モードを参照してください。	番号
includeInstanceUpdates	データセットのコンテンツ更新を含めるかどうかを定義します。デフォルトは false です。	番号
ページ付け	操作 getChanges および getChangesOnDataSet のページ付けコンテキストを有効にします。クライアントがページ付けコンテキストサイズを定義できるようにします。各ページには、最大サイズに応じて、テーブルの挿入、更新、および/または削除されたレコードのコレクションが含まれています。変更の取得永続化コンテキスト	番号

要素	説明	必須
	は、リクエストのページサイズパラメータに従って最初の呼び出しで構築されます。 ページネーション コンテキストは サーバー ファイル システム上 に保持され、最後のページまで、またはタイムアウトに達するまで次のページを呼び出すことができます。 作成の場合： <code>pageSize</code> パラメーターを定義します。次のステップ：前の応答の <code>識別子</code> を使用して <code>context</code> 要素を定義します。ページ付けを有効にします。以下の子要素を参照してください。	
<code>pageSize</code> (<code>pagination</code> 要素の下にネストされています)	各ページのレコードの最大数を定義します。最小サイズは <code>50</code> です。	いいえ（作成のみ）
<code>context</code> (<code>ページ付け</code> 要素の下にネストされています)	ページ付けコンテキストのコンテンツを定義します。	いいえ（次のみ）
<code>identifier</code> (<code>context</code> 要素の下にネストされています)	ページ付けコンテキスト識別子。	はい
最後のブロードキャストへのリダイレクトを無効にする	共通パラメータ の説明を参照してください。	

注:

`compareWithBranch` または `compareWithVersion` パラメーターのいずれも指定されていない場合、親との比較が行われます。

- 現在のデータスペースまたはスナップショットがデータスペースの場合、最初のスナップショットとの比較が行われます（データスペースで行われたすべての変更が含まれます）。
- 現在のデータスペースまたはスナップショットがスナップショットの場合、その親データスペースとの比較が行われます（現在のスナップショットが作成されてから親データスペースで行われたすべての変更が含まれます）。
- 現在のデータスペースが「参照」データスペースの場合、例外を返します。

も参照してください:

- [差異ヘルパー](#)

変更の応答を取得する

2つのデータセット間の変更

```
<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>

```

2 つのテーブル間の変更

```

<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>

```

と：

要素	説明	必須
挿入	選択肢 <code>compareWithBranch</code> または <code>compareWithVersion</code> から 挿入されたレコードが含まれ ます。挿入されたレコードの XMLエクスポートに対応する この要素の下コンテンツ。	番号
更新済み	更新されたレコードまたは データセットのコンテンツが 含まれます。	番号
changes (<code>updated</code> エレメ ント下にネスト)	フィールドのグループのみが 更新されました。	はい
change (<code>changes</code> エレメン ト下にネスト)	フィールドのグループは、レ コードの独自の <code>XPath述語</code> 属性で更新されました。	はい
path (<code>change</code> エlement下 にネスト)	レコード内のパス。	はい
path (<code>changes</code> Element 下にネスト)	データセット内のパス。	はい
data (<code>updated</code> Element 下にネスト)	データセットまたは更新済み レコードの XML エクスポート に対応するこのElement 下のコンテンツ。	いいえ
deleted	レコードはリクエストのコン テキストから削除されまし た。レコードの XPath 述語 を含む <code>predicate</code> エレメ ントのリストに対応するコン テンツ。	いいえ
pagination	リクエストでページ機能が有 効の場合。変更の永続化コン	いいえ

要素	説明	必須
	テキストを取得すると、最後のページまで、またはコンテキストタイムアウトに達するまで、次のページを呼び出すことができます。次のページが含まれます： <code>identifier</code> を使用して <code>context</code> エレメントを定義します。最後のページ： <code>identifier</code> を使用せずに <code>context</code> エレメントを定義します。ページネーションを有効にします。以下の子エレメントを参照してください。	
<code>context</code> (<code>pagination</code> エレメント下にネスト)	ページ付けコンテキストのコンテンツを定義します。	はい (next および last のみ)
<code>identifier</code> (<code>context</code> エレメント下にネスト)	ページ付けコンテキスト識別子。最後に返されたページでは定義されていません。	いいえ
<code>pageNumber</code> (<code>context</code> エレメント下にネスト)	ページ付けコンテキストの現在のページ番号。	はい
<code>totalPages</code> (<code>context</code> エレメント下にネスト)	ページ付けコンテキストの合計ページ数。	はい

ページ機能を有効にした getChanges 操作

`pagination` エレメントとサブエレメントのみが説明されています。

作成用

リクエストの抜粋

...

```
<pagination>
```

```
<!-- on first request for creation -->
```

```

        <pageSize>Integer</pageSize>

    </pagination>

    ...

```

応答の抜粋

```

    ...

    <pagination>

        <!-- on next request to continue -->

        <context>

            <identifier>String</identifier>

            <pageNumber>Integer</pageNumber>

            <totalPages>Integer</totalPages>

        </context>

    </pagination>

    ...

```

次のステップ

リクエストの抜粋

```

    ...

    <pagination>

        <context>

            <identifier>String</identifier>

        </context>

```

```
    </pagination>

    ...

```

応答の抜粋

```
    ...

    <pagination>

        <!-- on next request to continue -->

        <context>

            <identifier>String</identifier>

            <pageNumber>Integer</pageNumber>

            <totalPages>Integer</totalPages>

        </context>

    </pagination>

    ...

```

最後のステップ

リクエストの抜粋

```
    ...

    <pagination>

        <context>

            <identifier>String</identifier>

        </context>

    </pagination>

```

...

応答の抜粋

...

```
<pagination>
```

```
  <context>
```

```
    <pageNumber>Integer</pageNumber>
```

```
    <totalPages>Integer</totalPages>
```

```
  </context>
```

```
</pagination>
```

...

getCredentials 操作

getCredentials リクエスト

```
<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}>
```

と：

エレメント	説明	必須
branch	共通パラメーターの説明を参照してください。	
version	共通パラメーターの説明を参照してください。	
instance	共通パラメーターの説明を参照してください。	
predicate	共通パラメーターの説明を参照してください。	
viewPublication	共通パラメーターの説明を参照してください。	

getCredentials 応答

```
<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>

```

次の可能な値を使用します。

- R：読み取り専用
- W：読み取り/書き込み用
- H：隠し用

複数連鎖操作

複数操作のリクエスト

一貫した応答を確保しながら、データセット内のテーブル間で複数の操作を実行することが可能です。操作は、クライアント側で定義された順序に従って順番に実行されます。

すべての操作は、**SERIALIZABLE** 分離レベルの単一トランザクションで実行されます。複数操作のすべての要求が読み取り専用である場合、同じデータスペース内であっても、他の読み取り専用トランザクションと同時に完全に実行できます。

シーケンスの 1 つの操作中にエラーが発生すると、すべての更新がロールバックされ、クライアントは詳細を含む **StandardException** エラーメッセージを受け取ります。

[同時実行](#)を参照してください。

```

<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_>

```

と：

エレメント	説明	必須
branch	共通パラメーターの説明を参照してください。	
version	共通パラメーターの説明を参照してください。	
instance	共通パラメーターの説明を参照してください。	
blockingConstraintsDisabled	共通パラメーターの説明を参照してください。	
詳細	共通パラメーターの説明を参照してください。	

エレメント	説明	必須
disableRedirectionToLastBroadcast	共通パラメーターの説明を参照してください。	
request	このエレメントには、ブランチ、バージョン、インスタンスのパラメーターを含まない単一の操作のように、1つの操作が含まれます。このエレメントは、追加の操作のために複数回繰り返すことができます。各リクエストは、「id」属性で識別できます。応答では、この「id」属性が識別目的で返されます。 count、select、getChanges、getCredentials、insert、delete、update などの操作。	はい

注意：

- request エレメントの数の制限を受け入れません。
- リクエストの id 属性は、複数操作のリクエストで一意である必要があります。
- すべての操作が読み取り専用の場合 (count、select、getChanges、または getCredentials)、パフォーマンスを考慮して、トランザクション全体が読み取り専用設定されます。

制限事項

- 複数 操作は、1つのモデルと1つのデータセット (パラメーター instance) に適用されます。

以下も参照してください。：

- プロシージャ
- リポジトリ

複数操作の応答

詳細については、各応答操作を参照してください。

```
<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>
```

と：

エレメント	説明
response	このエレメントには、1つの操作の応答が含まれます。追加の操作のために複数回繰り返されます。各応答は、要求で設定された、または自動的に生成された「id」属性によって識別されます。エレメントの内容は、 <code>count</code> 、 <code>select</code> 、 <code>getChanges</code> 、 <code>getCredentials</code> 、 <code>insert</code> 、 <code>delete</code> 、 <code>update</code> などの単一の操作の応答に対応するものです。

楽観ロック

以前に読み取られたが、その間に変更された可能性のあるレコードの Update または Delete 操作を防ぐために、楽観ロックメカニズムが提供されています。

エレメント `includesMetadata` を `system` 値とともに追加することにより、Select リクエストに技術情報を含めることができます。

```
<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}>
```

次の更新リクエストでは、`ebx-metadata/system/update_time` エレメントの値を使用します。指定した時刻以降にレコードが変更された場合、更新はキャンセルされます。変更されたレコードが更新されないようにするには、`ebx-metadata/system/update_time` エレメントの値を追加します。

```
<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}>

```

lastTime 属性の値を使用して、変更されたレコードの削除を防ぐこともできます。

```

<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}>
```

注意:

`checkNotChangedSinceLastTime` エレメントは複数回使用できますが、同じレコードに対してのみ使用できます。これは、`predicate` エレメントが複数のレコードを返す場合、リクエストが失敗することを意味します。

データセットとデータスペースの操作

データスペースとスナップショットの操作のパラメーターは次のとおりです。

エレメント	説明	必須
branch	操作が適用されるターゲットデータスペースの識別子。指定しない場合、必要なデータスペースのマージ操作を除いて、「参照」データスペースが使用されます。	このパラメーターまたは「version」パラメーターのいずれかを定義する必要があります。データスペースのマージ、ロック、ロック解除、およびレプリケーションの Update 操作に必要です。
version	操作が適用されるターゲットスナップショットの識別子。	このパラメーターまたは「branch」パラメーターのいずれかを定義する必要があります。
versionName	作成するスナップショットの識別子。空の場合、サーバー側で定義されます。	いいえ
childBranchName	作成するデータスペースの子の識別子。空の場合、サーバー側で定義されます。	いいえ

エレメント	説明	必須
instance	操作が適用されるデータセットの一意の名前。	レプリケーションの更新操作に必要です。
ensureActivation	このインスタンスがアクティブ化されているかどうかを検証で確認する必要があるかどうかを定義します。	はい
details	検証が詳細を返すかどうかを定義します。オプションの属性 <code>severityThreshold</code> は、返すメッセージの最低の重大度レベルを定義します。可能な値は、重大度の降順で、「fatal」、「error」、「warning」、または「info」です。たとえば、値を「error」に設定すると、エラーおよび致命的な検証メッセージが返されます。この属性が定義されていない場合、デフォルトですべてのレベルのメッセージが返されます。オプションの属性 <code>locale</code> (デフォルトは「en-US」) は、検証メッセージが返される言語を定義します。	いいえ。指定しない場合、詳細は返されません。
owner	所有者を定義します。 <code>Profile.format</code> によって返される内部フォーマットを使用する必要があります。	いいえ
branchToCopyPermissionFrom	権限のコピー元となるデータスペースの識別子を定義します。	いいえ
documentation	専用言語のドキュメント。複数の <code>documentation</code> エレメントを複数の言語で使用できます。	いいえ

エレメント	説明	必須
locale (<code>documentation</code> エレメント下にネスト)	ドキュメントのロケール。	<code>documentation</code> エレメントが使用されている場合にのみ必要です。
label (<code>documentation</code> エレメント下にネスト)	言語のラベル。	番号
description (<code>documentation</code> エレメント下にネスト)	言語の説明。	番号

データスペースを検証する

データスペースリクエストを検証する

```
<m:validate xmlns:m="urn:ebx-schemas:dataservices_1.0">

  <branch>String</branch>

  <version>String</version>

</m:validate>
```

データスペースの応答を検証する

```
<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>

```

データセットの検証

データセットリクエストを検証する

```

<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>

```

データセットの応答を検証する

```

<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>

```

データスペースの作成

データスペースリクエストの作成

```

<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>

```

データスペース応答を作成する

```

<ns1:createBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

    <status>String</status>

    <childBranchName>String</childBranchName>

</ns1:createBranch_Response>

```

と：

要素	説明
status	「00」は、操作が正常に実行されたことを示します。

スナップショットの作成

スナップショットリクエストの作成

```

<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>

```

スナップショット応答を作成する

```

<ns1:createVersion_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

    <status>String</status>

    <versionName>String</versionName>

</ns1:createVersion_Response>

```

と：

要素	説明
status	「00」は、操作が正常に実行されたことを示します。

データスペースのロック

データスペースリクエストをロックする

```

<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>

```

と：

要素	説明	必須
durationToWaitForLock	このパラメーターは、操作が中止する前にロックを待機する最大期間（秒単位）を定義します。	いいえ、デフォルトでは待機しません
message	ロックのユーザーメッセージ。複数の <code>message</code> 要素を使用できます。	番号
locale（ <code>message</code> 要素の下にネストされています）	ユーザーメッセージのロケール。	<code>message</code> 要素が使用されている場合にのみ必要です
label（ <code>message</code> 要素の下にネストされています）	ユーザーメッセージ。	番号

データスペースの応答をロックする

```

<ns1:lockBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

    <status>String</status>

</ns1:lockBranch_Response>

```

と：

要素	説明
status	「00」は、操作が正常に実行されたことを示します。「94」は、データスペースが別のユーザーによってすでにロックされていることを示します。それ以外の場合は、SOAP例外がスローされます。

データスペースのロックを解除する

データスペースリクエストのロックを解除する

```
<m:unlockBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">

    <branch>String</branch>

</m:unlockBranch>
```

データスペース応答のロックを解除する

```
<ns1:unlockBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

    <status>String</status>

</ns1:unlockBranch_Response>
```

と：

要素	説明
status	「00」は、操作が正常に実行されたことを示します。それ以外の場合は、SOAP例外がスローされます。

データスペースのマージ

データスペースリクエストのマージ

```
<m:mergeBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">

    <branch>String</branch>

    <deleteDataOnMerge>boolean</deleteDataOnMerge>

    <deleteHistoryOnMerge>boolean</deleteHistoryOnMerge>

</m:mergeBranch>
```

と：

要素	説明	必須
deleteDataOnMerge	このパラメーターは、データスペースのマージ操作に使用できます。指定したデータスペースとそれに関連するスナップショットをマージ時に削除するかどうかを設定します。このパラメータがリクエストで指定されていない場合、デフォルト値は <code>false</code> になります。EBX® メイン構成ファイル でプロパティ <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> を指定することで、デフォルト値を再定義できます。詳細については、データと履歴の削除を参照してください。	番号
deleteHistoryOnMerge	このパラメーターは、データスペースのマージ操作に使用できます。指定したデータスペースに関連付けられている履歴をマージ時に削除するかどうかを設定します。デフォ	番号

要素	説明	必須
	ルト値は <code>false</code> です。このパラメータがリクエストで指定されていない場合、デフォルト値は <code>false</code> になります。EBX® メイン構成ファイルでプロパティ <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> を指定することで、デフォルト値を再定義できます。詳細については、データと履歴の削除を参照してください。	

注:

マージ決定ステップは、データサービスを介して実行されるマージ中にバイパスされます。このような場合、子データスペースのデータは親データスペースのデータを自動的に上書きします。

データスペースの応答をマージする

```
<ns1:mergeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

  <status>String</status>

</ns1:mergeBranch_Response>
```

と：

要素	説明
status	「00」は、操作が正常に実行されたことを示します。

データスペースまたはスナップショットを閉じる

データスペースまたはスナップショットリクエストを閉じる

データスペースリクエストを閉じる：

```
<m:closeBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">

  <branch>String</branch>

  <deleteDataOnClose>boolean</deleteDataOnClose>

  <deleteHistoryOnClose>boolean</deleteHistoryOnClose>

</m:closeBranch>
```

スナップショットリクエストを閉じる：

```
<m:closeVersion xmlns:m="urn:ebx-schemas:dataservices_1.0">

  <version>String</version>

  <deleteDataOnClose>boolean</deleteDataOnClose>

</m:closeVersion>
```

と：

要素	説明	必須
deleteDataOnClose	このパラメーターは、データスペースのクローズおよびスナップショットのクローズ操作に使用できます。指定されたスナップショット、またはデータスペースとそれに関連するスナップショットを閉じるときに削除するかどうかを設定します。このパラメーターがリクエストで指定されていない場合、デフォルト値は <code>false</code> です。EBX® メイン	番号

要素	説明	必須
	構成ファイル でプロパティ <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> を指定することで、このデフォルト値を再定義できます。詳細については、データと履歴の削除を参照してください。	
deleteHistoryOnClose	このパラメーターは、データスペースを閉じる操作に使用できます。指定したデータスペースに関連付けられている履歴を閉じるときに削除するかどうかを設定します。デフォルト値は <code>false</code> です。このパラメーターがリクエストで指定されていない場合、デフォルト値は <code>false</code> になります。EBX® メイン構成ファイル でプロパティ <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> を指定することで、デフォルト値を再定義できます。	番号

データスペースまたはスナップショット応答を閉じる

データスペースの応答を閉じる：

```
<ns1:closeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>String</status>
</ns1:closeBranch_Response>
```

スナップショット応答を閉じる：

```
<ns1:closeVersion_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>String</status>
```

```
</ns1:closeVersion_Response>
```

レプリケーションの更新

レプリケーション更新リクエスト

```
<m:replicationRefresh_${schema} xmlns:m="urn:ebx-
schemas:dataservices_1.0">

    <branch>String</branch>

    <instance>String</instance>

    <unitName>String</unitName>

</m:replicationRefresh_${schema}>
```

と：

要素	説明	必須
ブランチ	共通パラメータの説明を参照してください。	はい
実例	共通パラメータの説明を参照してください。	はい
ユニット名	レプリケーション ユニットの 名前。 も参照してください: • レプリケーション更新情報	はい

レプリケーション更新応答

```
<ns1:replicationRefresh_${schema}Response xmlns:ns1="urn:ebx-
schemas:dataservices_1.0">

    <status>String</status>

</ns1:replicationRefresh_${schema}Response>
```

と：

要素	説明
status	「00」は、操作が正常に実行されたことを示します。

データワークフローの操作

データワークフロー操作のパラメーターは、セッションのSOAPヘッダーから取得されます。

バージョン5.7.0以降は**非推奨**で、SOAPメッセージ本文にパラメータを定義します。

詳細については、[セッションパラメータ](#)を参照してください。

要素	説明	必須
パラメーター	バージョン5.7.0以降非推奨 下位互換性のために引き続き使用できますが、将来のメジャーバージョンで最終的に削除される予定です。 注: 少なくとも1つのセッションパラメータが定義されている場合、<code>parameters</code> 要素は無視されます。	番号
パラメータ（ <code>パラメータ</code> 要素の下にネストされています）	ワークフローの入力パラメーター。	番号

要素	説明	必須
す)。複数の パラメータ 要素を使用できます。		
名前（ パラメータ 要素の下にネストされています）	パラメータの名前	はい
値（ パラメータ 要素の下にネストされています）	パラメータの値。	番号

ワークフローを開始する

ワークフローランチャーからワークフローを開始します。ローカライズされたドキュメントと特定の入力パラメーター（名前とオプションの値）を使用してワークフローを開始することができます。

注:

ワークフロー作成者はセッションから初期化され、ワークフローの優先度は最後に公開されたバージョンから取得されます。

サンプルリクエスト：

```
<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>
```

と：

要素	説明	必須
公開されたプロセスキー	ワークフローランチャーの識別子。	はい
ドキュメンテーション	共通パラメータ の説明を参照してください。	番号
パラメーター	バージョン5.7.0以降非推奨 共通パラメータ の説明を参照してください。	番号

サンプルの回答：

```
<m:workflowProcessInstanceStart_Response xmlns:m="urn:ebx-
schemas:dataservices_1.0">

    <processInstanceKey>String</processInstanceKey>

</m:workflowProcessInstanceStart_Response>
```

と：

要素	説明	必須
processInstanceId	バージョン5.6.1以降非推奨このパラメーターは「processInstanceKey」パラメーターに置き換えられました。下位互換性のために引き続き使用できますが、将来のメジャーバージョンで最終的に削除される予定です。	番号
processInstanceKey	ワークフロー識別子。	番号

ワークフローを再開します

再開IDから待機ステップでワークフローを再開します。特定の入力パラメーター（名前とオプションの値）を定義することができます。

サンプルリクエスト：

```
<m:workflowProcessInstanceResume xmlns:m="urn:ebx-
schemas:dataservices_1.0">

    <resumeId>String</resumeId>

</m:workflowProcessInstanceResume>
```

と：

要素	説明	必須
resumeld	待機中のタスクのIDを再開します。	はい
パラメーター	バージョン5.7.0以降非推奨 共通パラメータの説明を参照してください。	番号

サンプルの回答：

```
<m:workflowProcessInstanceResume_Response xmlns:m="urn:ebx-
schemas:dataservices_1.0">

    <status>String</status>

    <processInstanceKey>String</processInstanceKey>

</m:workflowProcessInstanceResume_Response>
```

と：

要素	説明	必須
status	「00」は、操作が正常に実行されたことを示します。 「20」は、ワークフローが見つからなかったことを示します。「21」は、イベントがすでに受信されていることを示します。	はい
processInstanceKey	ワークフローの識別子。このパラメーターは、操作が正常に実行された場合に返されます。	番号

ワークフローを終了する

識別子からワークフローを終了します。

サンプルリクエスト：

```
<m:workflowProcessInstanceEnd xmlns:m="urn:ebx-schemas:dataservices_1.0">

  <processInstanceKey>String</processInstanceKey>

</m:workflowProcessInstanceEnd>
```

と：

要素	説明	必須
processInstanceKey	ワークフローの識別子。	このパラメーターまたは「publishedProcessKey」および「processInstanceId」パラメーターのいずれかを定義する必要があります。
公開されたプロセスキー	バージョン5.6.1以降非推奨制限により、このパラメーターは「processInstanceKey」パラメーターに置き換えられました。下位互換性のために引	番号

要素	説明	必須
	引き続き使用できますが、将来のメジャーバージョンで最終的に削除される予定です。	
processInstanceId	バージョン5.6.1以降非推奨制限により、このパラメーターは「processInstanceKey」パラメーターに置き換えられました。下位互換性のために引き続き使用できますが、将来のメジャーバージョンで最終的に削除される予定です。	番号

サンプルの回答：

```
<m:workflowProcessInstanceEnd_Response xmlns:m="urn:ebx-
schemas:dataservices_1.0">

    <status>String</status>

</m:workflowProcessInstanceEnd_Response>
```

と：

要素	説明	必須
status	「00」は、操作が正常に実行されたことを示します。	はい

行政サービス

ディレクトリサービス

ディレクトリのサービスは、デフォルト ディレクトリの「ユーザー」および「ロール」テーブルに対する操作を提供します。これらのサービスに関連する操作を実行するには、認証されたユーザーが組み込みロール「管理者」のメンバーである必要があります。

技術データスペースとデータセットは `ebx-directory` に設定する必要があります。すべての SOAP 操作構文の詳細については、[データ モデルから生成された操作](#) を参照してください。

ディレクトリにユーザーを作成する

この SOAP 挿入要求の例では、EBX® ディレクトリにユーザーを追加します。

```
<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>
```

挿入 SOAP 応答構文の詳細については、[挿入応答](#) を参照してください。

ユーザーインターフェース操作

詳細については、[アプリケーションのロック](#)を参照してください。

ユーザー インターフェースでの操作のパラメーターは次のとおりです。

要素	説明	必須
closedMessage	ユーザー インターフェースが閉じられてアクセスできないときにユーザーに表示されるメッセージ。	番号

ユーザーインターフェース要求を閉じる

閉じる操作により、このモードで受け入れられないすべてのユーザー セッションが削除されます。

```
<m:close xmlns:m="urn:ebx-schemas:dataservices_1.0">

  <closedMessage>Access is temporarily forbidden.</closedMessage>

</m:close>
```

ユーザーインターフェースの応答を閉じる

```
<ns1:close_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

  <status>String</status>

</ns1:close_Response>
```

オープンユーザーインターフェースリクエスト

```
<m:open xmlns:m="urn:ebx-schemas:dataservices_1.0"/>
```

オープンユーザーインターフェースの応答

```
<ns1:open_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

    <status>String</status>

</ns1:open_Response>
```

システム情報操作

この操作は、EBX® システム情報を返します。返される情報は、ログ ヘッダー `kernel.log` または UI タブ > に含まれる情報と同じです。応答には、EBX®の構成とステータスを表すいくつかのキー、ラベル、値が含まれます。この操作を実行するには、認証されたユーザーが組み込みロール「管理者」のメンバーである必要があります。

パラメーター

以下のパラメーターが適用されます。

パラメータ	説明	必須
<code>details</code>	応答メッセージに適用する必要がある属性を定義します。 属性 <code>locale</code> (デフォルト: EBX® デフォルト <code>locale</code>) は、システム項目メッセージが返される言語を定義します。	いいえ。ただし、指定する場合は、 <code>locale</code> 属性を指定する必要があります。

システム情報要求

この SOAP リクエストは、EBX® インスタンスのすべてのシステム情報を返し、"`en_US`" `ロケール` を使用してフォーマットします。

```
<m:systemInformation xmlns:m="urn:ebx-schemas:dataservices_1.0">

    <details locale="en_US" />

</m:systemInformation>
```

システム情報応答

```
<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 データサービス

- [はじめに](#)
- [ビルトイン RESTful サービス](#)
- [JSON 形式](#)
 - [はじめに](#)
 - [拡張](#)
 - [コンパクト形式](#)
 - [共通](#)
 - [その他](#)

はじめに

- 概要
- アクティベーションと構成
- インタラクション
 - ボディメディアタイプ
 - 追跡情報
 - セッションパラメーター
 - セッションチャネル
 - プロシージャのコンテキスト
 - 例外処理
- セキュリティ
 - 認証
 - グローバル許可
 - ルックアップメカニズム
- 監視
- SOAP と REST の比較
- 制限事項
 - 日付、時刻、日時の形式
 - JMS

概要

REST データサービスを使用すると、外部システムは、RESTful ビルトインサービスを使用して TIBCO EBX® リポジトリで管理されているデータと対話できます。

ビルトインサービスのリクエストと応答の構文については、[ビルトイン RESTful サービス](#)の章で説明しています。

ビルトイン REST データサービスにより、次のような操作を実行できます。

- レコードの選択、挿入、更新、削除、またはカウント
- 履歴レコードの選択またはカウント
- データセット値の選択、更新、またはカウント

- データスペースまたはスナップショット情報の選択または更新
- 子データスペースまたはスナップショットの選択
- データスペースの作成、マージ、またはクローズ
- スナップショットの作成または終了
- UI またはシステム情報へのアクセスを管理するための管理操作

注意:

SOAP と REST の比較を参照してください。

アクティベーションと構成

REST および SOAP データ サービスは、`ebx-dataservices` Web アプリケーションを他の EBX® モジュールとともに展開することでアクティブ化されます。詳細については、[Jakarta EE の展開](#) を参照してください。

リバースプロキシモードを使用するなど、特定の展開の場合、詳細については、[URL コンピューティング](#) を参照してください。

HTTP(S) プロトコルのみがサポートされています。

インタラクション

ボディメディアタイプ

REST ビルトインの入力メッセージと出力メッセージは、*排他的に* UTF-8 エンコードを使用した JSON 形式にする必要があります。ただし、入力メッセージを使用して HTML フォームデータを渡すこともできます。

詳細については、[Content-Type](#) を参照してください。

追跡情報

呼び出されるデータサービス操作によっては、セッション追跡情報を指定できる場合があります。

- RESTful 操作の例として、拡張 JSON 形式のリクエストには次のものが含まれます。


```

{

    "procedureContext":           // JSON Object (optional)

    {

        "trackingInformation": "String" // JSON String (optional)

    },

    ...

}

```

詳細については、Java API の `Session.getTrackingInfo` を参照してください。

セッションパラメーター

呼び出されるデータサービス操作に応じて、セッション入力パラメーターを指定できます。それらはリクエスト本文で定義されます。

入力パラメーターは、トリガー、アクセスルール、**カスタム** Web サービスなどのセッションオブジェクトを使用するカスタム Java コンポーネントで使用できます。これらは、データワークフロー操作でも使用できます。

- RESTful 操作の例として、拡張 JSON 形式のリクエストには次のものが含まれます。

```

{

    "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)

    },

    ...

]

},

...

}

```

詳細については、Java API の `Session.getInputParameterValue` を参照してください。

セッションチャネル

セッションチャネルを使用すると、REST ビルトインまたは REST Toolkit サービスを使用するときに、EBX® リポジトリから選択または変更できるものをフィルタリングできます。フィルターは、**デフォルトビュー**を指定することにより、データモデルを通じて可視性が定義されるテーブル、グループ、またはフィールドの構成に基づいています。

これは、クエリパラメーター **ebx-channel** を介して指定できます。使用可能な値は次のとおりです。

- `dataServices`
- `ui`

フィルターの動作は、次の組み合わせで記述されます。

データサービスチャネル

XML エLEMENT	値	スキーマノード タイプ	ビュー	動作
<code><hiddenInDataset</code>	<code>true</code>	フィールドノード	デフォルトの表 形式のビュー	コンテンツのため に非表示。並 べ替え不可

XML エlement	値	スキーマノード タイプ	ビュー	動作
CustomView (表形式または 階層形式)	メタ、コンテン ツ、フィル ター、並べ替え のために非表示			
デフォルトの フォームレコー ド	メタとコンテン ツのために非表 示			
デフォルトの フォームレコー ドフィールド	見つかりません			

ユーザーインターフェイスチャネル

XML エlement	値	スキーマノード タイプ	ビュー	動作
<hidden>	true	テーブルノード	ツリービュー	メタとコンテン ツのために非表 示
デフォルトの表 形式のビュー	見つかりません			
CustomView (表形式または 階層形式)	禁止			
デフォルトの フォームレコー ド	見つかりません			
デフォルトの フォームレコー ドフィールド				
カスタムフォー ムレコード				

XML エlement	値	スキーマノード タイプ	ビュー	動作
フィールドノード	デフォルトの表 形式のビュー	コンテンツのため に非表示。並 べ替え不可		
デフォルトの フォームレコード	コンテンツのため に非表示			
デフォルトの フォームレコード フィールド	見つかりません			
<hiddenInViews>	true	フィールドノード	CustomView (表形式または 階層形式)	メタ、コンテ ンツ、フィル ター、および並 べ替えで非表示
カスタムフォー ムレコード	メタとコンテ ンツのために非表 示			
<hiddenInSearch>	true	フィールドノード	デフォルトの表 形式のビュー	フィルタリング できません
CustomView (表形式または 階層形式)				
デフォルトの フォームレコード				
デフォルトの フォームレコード フィールド				
カスタムフォー ムレコード				

XML エlement	値	スキーマノード タイプ	ビュー	動作
textSearchOnly	デフォルトの表 形式のビュー	テキスト検索以 外はフィルタリ ングできません		
CustomView (表形式または 階層形式)				
デフォルトの フォームレコー ド				
デフォルトの フォームレコー ドフィールド				
カスタムフォー ムレコード				

注意:

上記のフィールドノードは、テーブルノードの下にのみ配置できます。

プロシージャのコンテキスト

呼び出されるデータサービス操作によっては、デフォルトのプロシージャコンテキスト構成を上書きすることができます。これらはリクエスト本文で定義され、ビルトインの操作内で適用されます。

プロシージャコンテキストは、カスタム REST Toolkit サービスに適用できます。

- RESTful 操作の例として、JSON リクエストの本文には次のものが含まれています。

```
{
  "procedureContext":          // JSON Object (optional)
  {
```

```

        "commitThreshold": Integer    // JSON Number (optional)

    },

    ...

}

```

詳細については、Java APIの `ProcedureContext.setCommitThreshold`、`SessionContext.getProcedureUtility`、および `ProcedureUtility.execute` を参照してください。

例外処理

エラーが発生すると、JSON 例外応答が呼び出し元に返されます。ほとんどの場合、例外は REST Toolkit のような JSON 形式を取ります。

以下も参照してください。:

- [REST Toolkit の例外処理](#)

```

{

    "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"
    }
]
}

```

複数のレコードの挿入または更新中にエラーが発生した場合、エラーレポートが返されます。

```

{
    "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)
        }
    ]
}

```

応答には、HTTP ステータスコードとエラーのテーブルが含まれています。各エラーの重大度は、可能な値 (`fatal` 、 `error` 、 `warning` 、 `info`) のいずれかを使用して文字で指定されます。

HTTP エラー `422` (処理不能エンティティ) は機能エラーに対応します。 `userCode` キーの下にユーザーコードが含まれ、JSON `String` 型です。

以下も参照してください。:

- HTTP コード
- `Severity.getLabel`

セキュリティ

認証

コンテンツへのアクセスには認証が必須です。いくつかの認証方法が利用可能で、以下で説明されています。説明は優先度順に並べられています (EBX® は最も優先度の高い認証方式を最初に適用します)。

- 「トークン認証スキーム」方式は、`RFC 2617` で説明されているように、HTTP ヘッダー `Authorization` に基づいています。

```
> Authorization: <tokenType> <accessToken>
```

この認証スキームの詳細については、[トークン認証操作](#)を参照してください。

以下も参照してください。:

- HTTP 認証ヘッダーポリシー

- 「基本認証スキーム」方式は、`RFC 2617 (基本認証スキーム)` で説明されているように、base64 エンコードの HTTP ヘッダー `Authorization` に基づいています。

ユーザーエージェントがユーザー ID「Alibaba」とパスワード「opensesame」を送信したい場合は、

次のヘッダーフィールドを使用します。

```
> Authorization: Basic QWxpYmFiYTpvvcGVuIHNlc2FtZQ==
```

注意:

`WWW-Authenticate` ヘッダーは、このメソッドで評価できます。

以下も参照してください。:

- HTTP 認証ヘッダーポリシー

- 「ベアラ認証スキーム」方式は、「トークン認証スキーム」に似ていますが、サードパーティプロバイダーから発行される可能性のあるベアラトークンで機能します。

```
> Authorization: bearer <accessToken>
```

この認証スキームの詳細については、[ベアラ認証構成](#)を参照してください。

- 「標準認証スキーム」は、HTTP `リクエスト` に基づいています。ユーザーとパスワードはリクエストパラメーターから抽出されます。リクエストパラメーターの詳細については、[パラメーターセクション](#)を参照してください。

この認証スキームの詳細については、`Directory.authenticateUserFromLoginPassword` を参照してください。

- 「RESTフォワード認証スキーム」は、現在の認証済みセッションを再利用する[ユーザーサービス](#)から REST サービスを呼び出す場合にのみ使用されます。

詳細については、[ユーザーサービスの実装](#)を参照してください。[REST データサービス](#)を呼び出します。

- 「特定の認証スキーム」は、HTTP リクエストに基づいています。たとえば、実装では、HTTP リクエストからパスワードダイジェストまたはチケットを抽出できます。詳細については、`Directory.authenticateUserFromHttpRequest` を参照してください。
- 「匿名認証スキーム」は、認証操作を処理する REST サービスにアクセスするためにのみ使用されます。認証情報の取得、パスワードの変更などは、ユーザーをまだ知ることができないことを意味します。

グローバル許可

グローバルアクセス権限は、REST ビルトインサービスと REST OpenAPI サービスに対して個別に定義できます。詳細については、[グローバル許可](#)を参照してください。

ルックアップメカニズム

EBX® はいくつかの認証方法を提供するため、条件に基づくルックアップメカニズムが設定され、特定のリクエストにどの方法を適用する必要があるかが分かります。メソッドの適用条件は、認証方式の優先度に従って評価されます。条件が満たされない場合、サーバーは次のメソッドを評価します。次の表に、サポートされている各プロトコルで使用可能な認証方法とその適用条件を示します。それらは、優先度の高いものから低いものの順に並べられます。

操作 / プロトコル	認証方法と適用条件
REST / HTTP	トークン • HTTP リクエストは、<code>Authorization</code> ヘッダーを保持する必要があります。 • <code>Authorization</code> ヘッダー値は <code>EBX</code> という単語で始まる必要があります。 • URL パラメーターに <code>login</code> は指定されていません。 基本 • HTTP リクエストは、<code>Authorization</code> ヘッダーを保持する必要があります。 • <code>Authorization</code> ヘッダー値は、<code>Basic</code> という単語で始まる必要があります。

操作 / プロトコル	認証方法と適用条件
	- URL パラメーターに <code>login</code> は指定されていません。 ベアートークン - HTTP リクエストは <code>Authorization</code> ヘッダーを保持する必要があります。 <code>Authorization</code> ヘッダー値は <code>Bearer</code> という単語で始まる必要があります。 - URL パラメーターに <code>login</code> は指定されていません。 標準 - HTTP リクエストは、<code>Authorization</code> ヘッダーを保持できません。 <code>login</code> と <code>password</code> は URL パラメーターで提供されます。 Rest フォワード - HTTP リクエストには、<code>Authorization</code> ヘッダーを含めることはできません。 - URL パラメーターに <code>login</code> は指定されていません。 特定 - HTTP リクエストを、以前の認証方法の条件を満たすものにすることはできません。 匿名 - 以前の認証方法はどれも適用できません。

操作 / プロトコル	認証方法と適用条件
	要求された REST サービスが認証操作を処理します。

同じリクエストに複数の認証方法が存在する場合、EBX® は HTTP コード **401 Unauthorized** を返します。

監視

データサービスイベントは、EBX® メイン構成ファイルで宣言されているログカテゴリ

`ebx.dataServices` を介して監視できます。たとえば、

`ebx.log4j.category.log.dataServices= INFO`、`ebxFile:dataservices` です。

以下も参照してください。:

- [EBX® ログの構成](#)
- [TIBCO EBX® メイン構成ファイル](#)

SOAP と REST の比較

操作	SOAP	REST
データ		
メタモデルを選択		X
レコードを選択またはカウント (フィルターおよび/またはパブリケーションの表示を使用)	X	X
関連値を持つレコードを選択 (第 1 レベル)	X	
ビジネスオブジェクトの選択		X

操作	SOAP	REST
レコードを更新または削除 (フィルターおよび/または ビューのパブリケーションを 使用)		X
可能な列挙値のセクター (フィルター付き)		X
作成または複製の準備		X
レコードの挿入、更新、また は削除	X	X
履歴レコードを選択またはカ ウント (フィルターおよび/ま たはパブリケーションの表示 を使用)		X
データセットからノード値を 選択	X	X
データセットからノード値を 更新		X
データスペースまたはスナッ プショット間のテーブルまた はデータセットの変更を取得	X	
レプリケーションユニットを 更新	X	
レコードの認証情報を取得	X	
サービス契約を生成	WSDL	OpenAPI
ビュー		
公開されたテーブルビューを 検索		X
データセット		

操作	SOAP	REST
データセット情報を選択		X
ルートまたは子データセットを選択		X
データスペース		
データスペースまたはスナップショット情報を選択		X
ルートまたは子のデータスペースを選択、またはスナップショットを選択		X
データスペースを作成、閉じる、マージ	X	X
スナップショットを作成、閉じる	X	X
データスペースまたはスナップショットを検証	X	
データセットの検証	X	
データスペースのロック、ロック解除	X	X
ワークフロー		
ワークフローの開始、再開、または終了	X	
管理		
デフォルトのディレクトリコンテンツ「ユーザー」、 「ロール」... テーブルの管理	X	X
ユーザーインターフェイスを開く、閉じる	X	X

操作	SOAP	REST
管理データセットを選択、挿入、更新、削除操作		X
システム情報を選択	X	X
ステージング API		X
ヘルスオペレーション		X
その他		
Java API からカスタムサービスを開発 (*)		X
カスタムサービス契約の生成 (*)		OpenAPI

(*) 詳細については、[REST Toolkit](#) を参照してください。

制限事項

日付、時刻、日時の形式

データサービスは、次の日付と時刻の形式のみをサポートします。

タイプ	形式	例
xs:date	yyyy-MM-dd	2007-12-31
xs:time	HH:mm:ss または HH:mm:ss.SSS	11:55:00
xs:dateTime	yyyy-MM-ddTHH:mm:ss または yyyy-MM-ddTHH:mm:ss.SSS	2007-12-31T11:55:00

JMS

- JMS プロトコルはサポートされていません。

組み込みRESTfulサービス

- はじめに
- リクエスト
 - HTTPメソッド
 - URL
 - ヘッダーフィールド
 - 共通パラメーター
 - メッセージボディ
- レスポンス
 - ヘッダーフィールド
 - HTTPコード
 - メッセージボディ
- 管理操作
 - ディレクトリ操作
 - ユーザーインターフェース操作
 - システム情報操作
- トークン認証操作
 - トークン作成操作
 - パスワード変更操作
 - トークン失効操作
- データ操作
 - 選択操作
 - 準備操作
 - 挿入操作
 - 更新操作
 - 削除操作
 - カウント操作
 - 楽観的ロック

- 継承
- テーブルビュー検索操作
- フォームデータ操作
 - フォーム挿入操作
 - フォーム更新操作
- データセット操作
 - データセットの選択
- データスペース操作
 - ベータ機能: データスペースまたはスナップショットの選択
 - ベータ機能: 子データスペースまたはスナップショットの作成
 - ベータ機能: データスペースのロック
 - ベータ機能: データスペースのロック解除
 - ベータ機能: データスペースのマージ
 - ベータ機能: データスペースまたはスナップショットのクローズ
- ヘルス操作
 - 概要
 - 開始操作
 - チェック操作
 - 停止操作
- OpenAPI操作
 - 概要
 - OpenAPIドキュメントの生成
- ステージング操作
 - ドメイン管理操作
 - ステージング要素ナビゲーション操作
 - アーカイブ操作
 - エクスポート操作

- インポート操作
- データモデル操作
- 制限事項
 - 一般的な制限事項
 - コンパクト形式の制限事項
 - ビジネスオブジェクトカテゴリの制限事項
 - 読み取り操作
 - 書き込み操作
 - ディレクトリ操作
 - OpenAPI操作

はじめに

使用されるアーキテクチャはROA（Resource-Oriented Architecture）と呼ばれ、SOA（Service-Oriented Architecture）の代替となり得ます。選択されたリソースは、リクエストの内容に応じて、サードパーティシステムによって読み取りおよび/または書き込み可能です。

組み込みRESTfulサービスのHATEOASアプローチは、直感的で分かりやすいナビゲーションも可能にし、データ詳細がリンクを通じて取得できることを意味します。

注記:

すべての操作はステートレスです。

リクエスト

この章では、HTTPメソッド、URL形式、ヘッダーフィールド、メッセージボディなど、準拠したRESTリクエストを構築するために使用する要素について説明します。

参照:

- インタラクション
- セキュリティ

HTTPメソッド

組み込みRESTfulサービスで考慮されるHTTPメソッドは次のとおりです。

- **GET** : URLで定義されたマスターデータを選択するために使用されます (URLのサイズ制限はアプリケーションサーバーとブラウザに依存し、2kB以下である必要があります)
- **POST** : テーブルにレコードを挿入するため、またはURLで定義されたマスターデータを選択するために使用されます (サイズ制限はアプリケーションサーバーによって2MB以上です。各パラメーターは1024文字を含む値に制限されます)
- **PUT** : URLで定義されたマスターデータを更新するために使用されます
- **DELETE** : URLで定義されたレコード、またはテーブルURLとメッセージボディ内のレコードキーによって定義された複数のレコードを削除するために使用されます

URL

REST URLには以下が含まれます。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/
{categoryVersion}/{specificPath}[:{extendedAction}]?{queryParameters}
```

ここで、

- **<ebx-dataservices>** は「ebx-dataservices.war」Webアプリケーションのパスに対応します。パスは、Webアプリケーション名に続く複数の、またはURIセグメントで構成されます
- **{category}** は**操作カテゴリ**に対応します
- **{categoryVersion}** はカテゴリバージョンに対応します。現在の値は **v1** です
- **{specificPath}** はカテゴリ内の特定のパスに対応します
- **{extendedAction}** は拡張アクション名に対応します (オプション)
- **{queryParameters}** はURLで渡される**共通**または専用の操作パラメーターに対応します

操作カテゴリ

操作を専門化します。 **{category}** のURLパスに追加され、次のいずれかの値を取ります。

admin

管理者専用の管理操作。

	管理操作の詳細については、こちらを参照してください。
auth	トークン認証メソッドを管理します。 トークン認証操作およびトークン認証スキームの詳細については、こちらを参照してください。
data または data-compact	データセットコンテンツをリスト表示します。データセットノード、テーブル、レコード、レコードフィールドに対する変更操作を含む、テーブル、レコード、またはフィールドレコードコンテンツをリクエストします。単純なユースケースでのインタラクションを容易にするために、コンパクト形式が利用可能です。 データスペースとスナップショットのライフサイクルを管理します。 データ操作、データスペース操作、およびコンパクト形式の制限事項の詳細については、こちらを参照してください。
data-bo または data-compact-bo	テーブル、レコード、またはフィールドレコードコンテンツをリクエストし、定義されたビジネスオブジェクトを介したナビゲーションを可能にします。単純なユースケースでのインタラクションを容易にするために、コンパクト形式が利用可能です。 ビジネスオブジェクトリレーションシップで定義された他のテーブルの下にある関連レコードとともにレコードを削除します。 データ操作の詳細については、こちらを参照してください。

form-data または form-data-compact	データセットノード、レコード、またはレコードフィールドを挿入または更新する前に、受信データを検証し、レポートを生成します。簡単なユースケースでのインタラクションを簡素化するために、コンパクト形式も利用可能です。 フォームデータ操作およびコンパクト形式の制限事項の詳細については、こちらを参照してください。
health	サーバーヘルス情報は、監視ツールまたはコンテナプローブの情報を提供します。 ヘルス操作の詳細については、こちらを参照してください。
history	履歴データセットのコンテンツをリスト表示します。履歴テーブル、レコードの履歴、または履歴レコードをリクエストします。 データ操作の詳細については、こちらを参照してください。 参照: • 履歴
api	選択されたリソースのOpenAPIドキュメントを生成します。 OpenAPI操作の詳細については、こちらを参照してください。
datamodel	データモデルアシスタントで定義されたデータモデルの公開を管理します。 データモデル操作の詳細については、こちらを参照してください。

参照:

- [データモデルの概要](#)
- [データモデルの公開](#)

ヘッダーフィールド

これらのヘッダーフィールド定義はTIBCO EBX®によって使用されます。

Accept

レスポンスで使用するコンテンツタイプを（優先順位順に）指定するために使用されます。最初にサポートされるものが選択され、レスポンスヘッダー `Content-Type` で指定されます。現在、唯一サポートされているのは `application/json` です。サポートされているものが何もない場合、結果はプロパティ

`ebx.dataservices.rest.request.checkAccept` に依存します。

- `true` の場合、HTTPエラーレスポンスがコード `406` とともに返されます
- `false` の場合、レスポンスはデフォルトのコンテンツタイプ、つまり `application/json` とともに返されます

参照:

- [データサービスの設定](#)

Accept-Language

レスポンスの優先ロケールを指定するために使用されます。サポートされているロケールはスキーマモデルで定義されています。

優先ロケールのいずれもサポートされていない場合、現在のモデルのデフォルトロケールが使用されます。

Authorization	サポートされている認証スキームには、「基本認証スキーム」と「トークン認証スキーム」が含まれます。別のスキームが使用された場合、リクエストは拒否されます。 参照: • 認証
Content-Type	リクエストボディのメディアタイプを指定するために使用されます。サポートされているタイプは <code>application/json</code> と <code>application/x-www-form-urlencoded</code> です。リクエスト値がサポートされていない場合、HTTPエラーメッセージがコード <code>415 (Unsupported media type)</code> とともに返されます。 参照: • データサービスの設定
X-Requested-With	存在し、認証失敗の場合、レスポンスに <code>WWW-Authenticate</code> ヘッダーが追加されるのを防ぎます。 参照: • レスポンスヘッダー <code>WWW-Authenticate</code>

HTTPヘッダーフィールド定義の詳細については、[RFC2616](#)を参照してください。

共通パラメーター

これらのオプションパラメーターは、すべてのデータサービス操作で利用可能です。

パラメーター	説明
<code>disableRedirectionToLastBroadcast</code>	このパラメーターは、D3アーキテクチャでのみ影響があります。 <code>true</code> の場合、D3プライマリノード上の配信データスペースへのアクセスは、最後のブロードキャストスナップショットにリダイレクトされません。それ以外の場合、そのようなデータスペースへのアクセスは常に最後のブロードキャストスナップショットにリダイレクトされます。 指定されたデータスペースがD3プライマリノード上の配信データスペースでない場合、このパラメーターは無視されます。 <code>Boolean</code> 型の値。このパラメーターが存在しない場合、構成プロパティ <code>ebx.dataservices.disableRedirectionToLastBroadcast.default</code> が設定されていない限り、デフォルトは <code>false</code>（D3マスターへのリダイレクトが有効）です。 参照: プライマリノード
<code>ebx-indent</code> <code>indent</code> (6.0.0以降非推奨)	レスポンスをインデントするかどうかを指定します。これにより、人間にとって読みやすくなります。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。
<code>ebx-channel</code>	セッションチャンネルを指定します。 <code>String</code> 型、可能な値は次のとおりです。

パラメーター	説明
	<code>dataServices</code> <code>ui</code> デフォルト値は <code>dataServices</code> です。 参照: <code>Constants.Data.PARAM_EBX_CHANNEL</code>

メッセージボディ

JSON形式のリクエストデータが含まれています。拡張JSONリクエストボディおよびコンパクトJSONリクエストボディを参照してください。

注記:

リクエストは、`POST` または `PUT` HTTPメソッドを使用する場合にのみメッセージボディを定義できます。

レスポンス

この章では、組み込みRESTfulサービスによって返されるレスポンスについて説明します。

- 標準エラー処理（HTTPコードが `300` 以上の場合）の詳細については、[例外処理](#)を参照してください

ヘッダーフィールド

これらのヘッダーフィールド定義はEBX®によって使用されます。

<code>Content-Language</code>	レスポンスでラベルと説明に使用されるロケールを示します。
<code>Content-Type</code>	レスポンスボディのコンテンツタイプを示します。

Location	新しいレコードが正常に挿入された場合、このフィールドによってそのレコードのクエリURLが返されます。
WWW-Authenticate	このヘッダーフィールドは、認証が401 (<i>Unauthorized</i>) HTTPコードで失敗した場合にHTTPレスポンスに追加されます。その値は、リクエストURIに適用可能な少なくとも1つの認証メソッドを含むリストで構成されます。これは、次の条件が満たされている場合にのみ存在します。 「基本認証スキーム」メソッドが有効であること、および - X-Requested-With HTTPヘッダーが存在しないこと クライアントが認証メソッドを解釈できる場合、適切な資格情報を提供してリクエストを再送信することが可能です。 管理プロパティ <code>ebx.dataservices.rest.auth.tryBasicAuthentication</code> は <code>true</code> に設定する必要があります。 参照: - リクエストヘッダー X-Requested-With - 認証

HTTPコード

HTTPコード	説明
200 (OK)	リクエストは正常に処理されました。

HTTPコード	説明
201 (Created)	新しいレコードが作成されました。この場合、ヘッダーフィールド Location にはそのリソースURLが返されます。
204 (No content)	リクエストは正常に処理されましたが、レスポンスボディは返されません。
206 (Partial Content)	リクエストは正常に処理されましたが、一貫性のある部分的なレスポンスボディが返されます。
400 (Bad request)	リクエストURLまたはボディの形式が正しくないか、無効なコンテンツが含まれています。
401 (Unauthorized)	認証に失敗しました。
403 (Forbidden)	認証されたユーザーに対して、指定されたリソースの読み取りまたは変更の権限が拒否されました。 このエラーは、ユーザーが次のいずれかの場合にも返されます。 - リクエストメッセージボディで言及されているフィールドを変更する権限がない - RESTコネクターにアクセスする権限がない 詳細については、グローバル権限を参照してください。
404 (Not found)	URLで指定されたリソースが見つかりません。
406 (Not acceptable)	リクエストの Accept パラメーターで定義されたコンテンツタイプはサポートされて

HTTPコード	説明
	いません。このエラーは、EBX®プロパティ <code>ebx.rest.request.checkAccept</code> が <code>true</code> に設定されている場合にのみ返されます。
409 (Conflict)	同時変更が発生しました。 参照: 楽観的ロック
415 (Unsupported media type)	リクエストコンテンツはサポートされていません。リクエストヘッダー値 <code>Content-Type</code> は操作でサポートされていません。
422 (Unprocessable entity)	新しいリソースのコンテンツは、セマンティックな理由により受け入れられません。
500 (Internal error)	アプリケーションによって予期しないエラーがスローされました。エラーの詳細は通常、EBX®ログで確認できます。

メッセージボディ

レスポンスボディのコンテンツ形式は、HTTPコードの値によって異なります。

- 200 以上 300 未満のHTTPコード: コンテンツ形式は関連するリクエストに依存します（拡張JSONおよびコンパクトJSONの例）。

コード 204 *(No content)*は例外です。

- 300 以上のHTTPコード: コンテンツはエラーを記述します。形式の詳細については、JSONを参照してください。

管理操作

管理操作は以下に関連します。

- 管理カテゴリ

- `data` カテゴリを通じてアクセス可能な管理データスペース

注記:

管理カテゴリと管理データスペースは管理者のみが使用できます。

ディレクトリ操作

EBX®のデフォルトディレクトリ構成は、組み込みRESTfulサービスで管理可能です。ユーザーとロールのテーブル、メーリングリスト、その他のオブジェクトが対象となります。詳細については、[ユーザーとロールのディレクトリ](#)を参照してください。

注記:

ディレクトリのテーブルにはトリガーが存在し、データの一貫性を保証します。

URL形式は次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/Bebx-directory/  
ebx-directory
```

ディレクトリ構成操作

EBX®のデフォルトディレクトリ構成は、データセットノードと同様に管理可能です。`data` カテゴリ操作を通じてアクセスおよび変更できます。メタモデルがリクエストされると、各フィールドは自己記述されます。

詳細については、[選択](#)および[更新](#)操作を参照してください。

メーリングリスト操作

EBX®ディレクトリには、全員用と管理者用の2つのデフォルトメーリングリストを設定できます。これらのリストは、`data` カテゴリ操作を通じてデータセットノードと同様に処理できます。

詳細については、[選択](#)および[更新](#)操作を参照してください。

ディレクトリユーザー操作

ユーザーは、`data` カテゴリのレコードと同様に、その操作を使用して操作できます。セキュリティ上の理由から、管理者は自分自身を削除できません。ユーザーの敬称は、「salutations」テーブルで利用可能なものの中から選択する必要があります。

詳細については、[選択](#)、[更新](#)、[挿入](#)、および[削除](#)操作を参照してください。

ディレクトリロール操作

ロールは `data` カテゴリのレコードであり、その操作で管理できます。EBX®ロールは、「usersRoles」関連テーブルを通じてユーザーに割り当てられます。「usersRoles」は、ユーザーインターフェースを通じてディレクトリが管理されるときに自動的に入力されます。ただし、データサービスを通じてはそうではなく、ロールの割り当てには手動操作が必要です。ロールの包含は、「rolesInclusions」関連テーブルで指定されます。「usersRoles」テーブルと同様に、ロールの包含の管理には手動操作が必要です。メタモデルがリクエストされると、各テーブルは自己記述されます。

詳細については、[選択](#)、[更新](#)、[挿入](#)、および[削除](#)操作を参照してください。

ユーザーインターフェース操作

EBX®ユーザーインターフェースは、メンテナンスの必要に応じてユーザーに対して開閉できます。処理される情報は、UIタブ「管理」>「ユーザーインターフェース構成」>「詳細パースペクティブ」>「グラフィカルインターフェース構成」>「アプリケーションロック」に含まれるものと同様です。

URL形式は次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/Bebx-manager/ebx-manager/domain/toolStatus
```

参照:

- [アプリケーションロック](#)

ユーザーインターフェースの状態の取得

ユーザーインターフェースの状態と利用不可メッセージは、データセットノードと同様にアクセス可能です。

詳細については、[選択操作](#)および[拡張JSON](#)または[コンパクトJSON](#)の例を参照してください。

ユーザーインターフェースの開閉

ユーザーインターフェースの状態と利用不可メッセージは、更新操作を使用してデータセットノードと同様に変更できます。ユーザーインターフェースを開くには `toolStatus` の内容を `true` に設定し、閉じるには `false` に設定します。

詳細については、[更新操作](#)および[拡張JSON](#)または[コンパクトJSON](#)の例を参照してください。

システム情報操作

この操作は、EBX®サーバーのシステム情報を返します。これは `GET` および `POST` HTTPメソッドで受け入れられます。警告: リクエストボディが無視されるため、`POST` HTTPメソッドでは更新はできません。返される情報は、ログヘッダー `kernel.log` またはUIタブ「管理」>「システム情報」に含まれる情報と同じです。レスポンスには、EBX®の構成と状態を表すいくつかのキー、ラベル、値が含まれます。レスポンスの表現モードは、フラットまたは階層的である場合があります。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/admin/v1/  
systemInformation
```

参照:

- [TIBCO EBX®メイン構成ファイル](#)
- [リポジトリ管理](#)

パラメーター

以下のパラメーターが適用されます。

パラメーター	説明
<code>systemInformationMode</code>	返されるモードを指定します。 <code>flat</code> : 次の情報グループ (<code>bootInfoEBX</code> 、 <code>repositoryInfo</code> 、 <code>bootInfoVM</code>) の下のフラットな表現 <code>hierarchical</code> : 階層表現 <code>String</code> 型、デフォルト値は <code>flat</code> です。

HTTPコード

HTTPコード	説明
<code>200</code> (OK)	システム情報が正常に返されました。
<code>400</code> (Bad request)	リクエストが正しくありません。次のいずれかのエラーが含まれています。 - HTTPメソッドが <code>GET</code> でも <code>POST</code> でもない - HTTPパラメータ <code>systemInformationMode</code> が正しくない - 操作がサポートされていない - リクエストパスが無効である
<code>403</code> (Forbidden)	ユーザーは管理者ではありません。

レスポンスボディ

HTTPコードが `200` (OK) の場合にのみ返されます。コンテンツ構造は、指定されたパラメータ `systemInformationMode` またはそのデフォルト値によって異なります。

フラット表現のJSON例を参照してください。

階層表現のJSON例を参照してください。

トークン認証操作

これらの操作により、認証トークンを作成または取り消すことができます。認証トークンにはタイムアウト期間があります。この期間内にEBX®サーバーへのアクセスに使用されなかった場合、トークンは自動的に取り消されます。このタイムアウト期間は、EBX®サーバーへのアクセスごとに更新されます。

注記:

トークンのタイムアウトは、管理プロパティ `ebx.dataservices.rest.auth.token.timeout`（デフォルト値は30分）を介して変更できます。

注記:

トークンは、メインディレクトリで定義されたユーザーに対してのみ生成できます。ディレクトリの詳細については、[ユーザーとロールのディレクトリ](#)を参照してください。

トークン作成操作

この操作では、ユーザーの資格情報と、オプションでセッションパラメータを含むリクエストで `POST` HTTPメソッドを使用する必要があります。

URL形式は次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/token:create
```

メッセージボディ

HTTPリクエストでメッセージボディを定義する必要があります。これは、次のいずれかのデータセットを必ず含みます。

- `login` と `password` の値。両方のJSON属性は必須であり、`String` 型です。

詳細については、`Directory.authenticateUserFromLoginPassword` を参照してください。

- `specific` JSON属性が `true` に設定されていること。このフラグがアクティブ化されると、HTTPリクエスト全体に対してユーザー認証を実行できます。警告: JSONリクエストのボ

ディに `login` および `password` 属性が定義されていても、`specific` を `true` に設定すると、特定のユーザー認証が行われます。

詳細については、`Directory.authenticateUserFromHttpRequest` を参照してください。

トークン作成リクエストのJSON例を参照してください。

HTTPコード

HTTPコード	説明
<code>200</code> (OK)	トークンが正常に作成されました。
<code>400</code> (Bad request)	次のいずれかの理由によるものです。 • 構文が正しくない • HTTPメソッドが <code>POST</code> ではない • 操作がサポートされていない
<code>401</code> (Unauthorized)	次のいずれかの理由によるものです。 • <code>login</code> または <code>password</code> が正しくない • 他のメソッドの認証データが定義されている
<code>422</code> (Unprocessable entity)	次のいずれかの理由によるものです。 • <code>PasswordMustChange</code> : パスワードを変更する必要があります（デフォルトディレクトリでのみ利用可能）。パスワード変更操作を参照してください。 • <code>RestrictedAccess</code> : メンテナンスまたはその他のアクションのためにユーザーアクセスが閉じられています（管理者専用）。

レスポンスボディ

HTTPコードが `200 (OK)` の場合、ボディにはトークン値とそのタイプが含まれます。

トークン作成レスポンスのJSON例を参照してください。

トークンは、後でHTTPヘッダー `Authorization` を適切に設定することで、ユーザーを認証するために使用できます。

関連項目:

- 「トークン認証スキーム」メソッド

パスワード変更操作

この操作は、既存のユーザーアカウントのパスワードを変更します。認証されたコンテキストで使用できます。 `login` パラメータが存在する場合、現在のセッションと照合され、存在しない場合はセッションから取得されます。また、非認証コンテキストでも使用できます。たとえば、[トークン作成操作](#)がHTTPコード `422` *(処理できないエンティティ)*で理由 `PasswordMustChange` により中止された場合などです。

次の使用が必要です。

- EBX®デフォルトディレクトリ
- `POST` HTTPメソッド
- 以下に指定された構造を含むメッセージボディ

URL形式は次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/  
user:changePassword
```

メッセージボディ

リクエストでメッセージボディを定義する必要があります。これは、 `password` と `passwordNew` を必ず含み、 `login` はオプションです（すべて `String` 型）。

パスワード変更およびトークン作成リクエストのJSON例を参照してください。

HTTPコード

HTTPコード	説明
204 (No content)	パスワードが変更されました。
400 (Bad request)	次のいずれかの理由によるものです。 • EBX®デフォルトディレクトリが必要である • 構文が正しくない • HTTPメソッドが POST ではない • 提供された login とユーザーのセッションの login が一致しない • 操作がサポートされていない
401 (Unauthorized)	次の理由によるものです。 • login または password が正しくない
422 (Unprocessable entity)	次のいずれかの理由によるものです。 • PasswordChangeAbort : passwordNew が空である • PasswordChangeAbort : passwordNew が password と同じである • PasswordChangeAbort : password が正しくない • RestrictedAccess : メンテナンスまたはその他のアクションのためにユーザーアクセスが閉じられています（管理者専用）

レスポンスボディ

HTTPコード `204 (No content)` が返された場合、パスワードが変更されました。

トークン取り消し操作

この操作では、`POST` HTTPメソッドを使用する必要があります。メッセージボディは不要です。

URL形式は次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/token:revoke
```

ヘッダーフィールド

`Authorization`

このフィールドは必須であり、`tokenType` フィールドと `accessToken` フィールドには「トークン作成」操作から返された値が必要です。

> Authorization: <tokenType> <accessToken>

HTTPコード

HTTPコード	説明
<code>204 (No content)</code>	トークンが正常に取り消されました。
<code>400 (Bad request)</code>	次のいずれかの理由によるものです。 • 設定がアクティブ化されていない • 構文が正しくない • HTTPメソッドが <code>POST</code> ではない • 操作がサポートされていない
<code>401 (Unauthorized)</code>	認証に失敗しました。

データ操作

`data` カテゴリ操作は、データセット、データセットフィールド、テーブル、レコード、またはレコードフィールドに関するものです。

`data-compact` カテゴリ操作は、データセットフィールド、テーブル、レコード、またはレコードフィールドに関するものです。

`data-bo` カテゴリ操作は、[ビジネスオブジェクト](#)をナビゲートする機能を備えた `data` カテゴリ操作に関するものです。

`data-compact-bo` カテゴリ操作は、[ビジネスオブジェクト](#)をナビゲートする機能を備えた `data-compact` カテゴリ操作に関するものです。

`history` カテゴリ操作は、データセット、テーブル、レコード、またはレコードフィールドからの履歴コンテンツに関するものです。

`form-data` カテゴリ操作は、コンテンツの作成または更新時に制約が有効である必要がある場合のデータセットフィールド、レコード、またはレコードフィールドに関するものです。

`form-data-compact` カテゴリ操作は、コンテンツの作成または更新時に制約が有効である必要がある場合のデータセットフィールド、レコード、またはレコードフィールドに関するものです。

詳細については、[フォームデータ操作](#)を参照してください。

選択操作

選択操作は階層コンテンツを返します。この操作では、次のいずれかのメソッドを使用できます。

- `GET` HTTPメソッド
- メッセージボディなしの `POST` HTTPメソッド
- メッセージボディ、`:select` URL拡張アクション、およびオプションで[セッションパラメータ](#)を含む `POST` HTTPメソッド

URL形式は次のとおりです。

- [データセットツリー](#)（[操作カテゴリ](#)による）

`data` または `data-compact` カテゴリは、グループノードとテーブルノードを含む、選択されたデータセットの階層を返します。

`history` カテゴリは、履歴テーブルノードのみの剪定されたグループを含む、選択された履歴データセットの階層を返します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}
```

注記:

ターミナルノードとサブノードは含まれません。

- **データセットノード:** `data` または `data-compact` カテゴリは、選択されたノードに含まれるターミナルノードを返します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}[:select]
```

注記:

`history` カテゴリには適用されません。

- **テーブル (操作カテゴリによる)**

`data` または `data-compact` カテゴリは、テーブルコンテンツおよび/またはメタモデル、現在のページレコード、およびページネーション用のURLを返します。

`history` カテゴリは、履歴テーブルコンテンツおよび/またはメタモデル、現在のページレコード、およびページネーション用のURLを返します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}[:select]
```

関連項目:

- [カウント操作](#)
- [テーブルビュー検索操作](#)

- レコード（操作カテゴリによる）

`data` または `data-compact` カテゴリは、レコードコンテンツおよび/またはメタモデルを返します。

`history` カテゴリは、履歴レコードコンテンツおよび/またはメタモデルを返します。

```
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}
```

注記:

プライマリキー（`primaryKey` パラメータ）によるレコードアクセスは、そのルートノードに限定されます。この制限を上書きするには、`details` フィールドで利用可能なエンコードされたプライマリキーを使用することをお勧めします。同様に、履歴レコードの場合、`historyDetails` フィールドで利用可能なエンコードされたプライマリキーを使用してください。

- フィールド（操作カテゴリによる）

`data` または `data-compact` カテゴリは、そのタイプに応じて構造が異なるフィールドレコードコンテンツを返します。

`history` カテゴリは、そのタイプに応じて構造が異なるフィールド履歴レコードコンテンツを返します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
[:select]
```


注記:

フィールドは、関連ノード、選択ノード、ターミナルノード、またはそれ以上である必要があります。

ここで、

- `{category}` は操作カテゴリに対応します（可能な値は `data`、`data-compact`、`data-bo`、`data-bo-compact`、または `history` です）。
- `{dataspace}` は、データスペース識別子に続く `B`、またはスナップショット識別子に続く `V` に対応します。
- `{dataset}` はデータセット識別子に対応します。
- `{pathInDataset}` は、グループノードまたはテーブルノードであるデータセットノードのパスに対応します。
- `{encodedPrimaryKey}` は、プライマリキーのパーセントエンコード表現に対応します（RFC-3986 Uniform Resource Identifierを参照）。
- `{xpathExpression}` は、XPath式を使用したレコードプライマリキーに対応します。
- `{pathInRecord}` は、テーブルノードから始まるパスに対応します。
- `:select` 拡張アクションは、ボディメッセージとともに `POST` HTTPメソッドが使用される場合に必要です。

ビジネスオブジェクトカテゴリ（`data-bo` および `data-compact-bo`）は、他のテーブルからのデータを追加することで、`Table`、`Record`、および `Field` リクエストのレスポンスを強化します。

パラメータ

次のパラメータは選択操作に適用されます。

パラメータ	説明
<code>includeContent</code>	選択に対応するコンテンツを含む <code>content</code> フィールドを含めます。 <code>Boolean</code> 型、デフォルト値は <code>true</code> です。

パラメータ	説明
<code>includeDetails</code>	メタモデルおよびコンテンツに <code>details</code> フィールドを含め、間接的に到達可能な各リソースについて含めます。返される値は、そのURLリソースに対応します。 <code>Boolean</code> 型、デフォルト値は <code>true</code> です。 関連項目: • includeMetamodel
<code>includeHistory</code>	履歴コンテンツの次のフィールドを含めます。 • メタモデルの <code>history</code> プロパティ。返される値はブール値に対応します。 • コンテンツおよび間接的に到達可能な各リソースの <code>historyDetails</code> プロパティ。この点は <code>includeDetails</code> パラメータと関連しています。返される値は、そのURLリソースに対応します。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。 注記: <code>includeHistory</code> パラメータは <code>history</code> カテゴリでは無視され、デフォルト値は <code>true</code> です。 関連項目: • includeMetamodel • includeDetails

パラメータ	説明
<code>includeLabel</code>	各単純型 <code>content</code> に関連付けられた <code>label</code> フィールドを含めます。 可能な値は次のとおりです。 • <code>yes</code> : 外部キー、列挙、レコード、およびセクタ値に <code>label</code> が含まれます。 • <code>all</code> : <code>yes</code> 値の場合と同様に、単純型のコンテンツにも <code>label</code> フィールドが含まれます。 関連項目: ◦ <code>SchemaNode.displayOccurrence</code> • <code>no</code> : <code>label</code> フィールドは含まれません (統合ユースケース)。 <code>String</code> 型、デフォルト値は <code>yes</code> です。 注記: <code>label</code> フィールドが <code>content</code> フィールドと同じ場合、<code>label</code> フィールドは含まれません。
<code>includeMeta</code>	6.1.0以降非推奨、includeMetamodelに置き換えられました。
<code>includeMetadata</code>	レスポンスには指定されたメタデータが含まれます。返される各レコードには、<code>ebx-metadata</code> ルート要素の下にステップと呼ばれる追加要素が含まれます。 たとえば、「system」ステップ値には技術データが含まれます。詳細については、楽観的ロックを参照してください。

パラメータ	説明
	文字列値（デフォルト値は空）のステップはコンマで区切る必要があります: <code>system, teamUp</code>。すべてのステップは <code>ebx-all</code> 値を使用して返すことができます。 関連項目: • 技術データ
<code>includeMetamodel</code>	<code>content</code> フィールドで返される構造の説明に対応する <code>meta</code> フィールドを含めます。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。 関連項目: • メタモデル • includeHistory • includeDetails
<code>includeMergeInfo</code>	履歴トランザクションの技術データのフィールドに対応し、アクセスコストが高い可能性がある <code>merge_info</code> を含めます。 <code>Boolean</code> 型、デフォルト値は <code>true</code> です。 注記: このパラメータは <code>data</code> カテゴリでは無視されます。 関連項目: • 履歴テーブルへのRESTアクセス

パラメータ	説明
<code>includeOpenApiDetails</code>	各記述可能ノードのOpenAPI仕様URLを含めます。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。 注記: このクエリパラメータは、<code>includeDetails</code>が <code>false</code> に設定されている場合、無視されます。 関連項目: • OpenAPI操作 • includeDetails
<code>includeSelector</code>	レスポンスに <code>selector</code> フィールドを含め、間接的に到達可能な各リソースについて含めます。返される値は、そのURLリソースに対応します。 <code>Boolean</code> 型、デフォルト値は <code>true</code> です。 関連項目: • セクタ
<code>includeSortCriteria</code>	適用されたソート条件のリストに対応する <code>sortCriteria</code> フィールドを含めます。 ソート条件パラメータは、次を使用して追加されます。 • ソート • ラベルによるソート • ビュー公開

パラメータ	説明
	<code>Boolean</code> 型、デフォルト値は <code>false</code> です。 例: JSON
<code>includeTechnicals</code>	6.1.0以降非推奨、<code>includeMetadata</code> に置き換えられました。 注記: このパラメータは <code>history</code> カテゴリでは無視されます。
<code>includeValidation</code>	選択に対応する検証レポートを含めます。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。 注記: このパラメータは <code>history</code> カテゴリでは無視されます。 関連項目: • includeDetails

テーブルパラメータ

次のパラメータは、テーブル、関連、および選択ノードに適用されます。

パラメータ	説明
<code>filter</code>	クイック検索述語または完全なXPath述語式は、リクエストが適用されるフィールド値を定義します。空の場合、すべてのレコードが取得されます。 <code>String</code> 型の値。

パラメータ	説明
	注記: 履歴コード操作値は、このフィールドに関連付けられた <code>meta</code> セクションの <code>ebx-operationCode</code> パスフィールドで使用できます。 Seealso: <pre><title> TIBCO EBX® ドキュメンテーション - 組み込みRESTfulサービス</title> <meta content="text/html; charset=utf-8" http- equiv="Content-Type"/><meta content="Copyright 2001-2026. Cloud Software Group, Inc. All rights reserved." name="copyright"/><meta content="dataservices_rest_v1" name="doc_id"/><link href=" ../resources/ stylesheets/ebx_common.css" rel="stylesheet" type="text/css"/><link href=" ../resources/ stylesheets/ebx_docPage.css" rel="stylesheet" type="text/css"/><link href=" ../resources/ syntaxHighlighter/styles/shCoreEclipse.css" rel="stylesheet" type="text/css"/><link href=" ../ resources/icons/brand.ico" rel="icon" type="image/x-icon"/></pre> • 関連性によるソート • <code>includeMetadata</code> パラメーター 参照: • 検索述語
<code>historyMode</code>	テーブルに適用されるフィルターコンテキストを指定します。 文字列型、指定可能な値は次のとおりです。 • <code>CurrentDataSpaceOnly</code> : 現在のデータスペースの履歴

パラメータ	説明
	<code>CurrentDataSpaceAndAncestors</code> : 現在のデータスペースと祖先の履歴 <code>CurrentDataSpaceAndMergedChildren</code> : 現在のデータスペースとマージされた子孫の履歴 <code>AllDataSpaces</code> : すべてのデータスペースの履歴 デフォルト値は <code>CurrentDataSpaceOnly</code> です。 参照: <code>history</code> 注記: このパラメータは <code>data</code> カテゴリでは無視されます。
<code>includeOcculting</code>	オカルティングモードのレコードを含めます。 ブール値型、デフォルト値は <code>false</code> です。 参照: 継承
<code>primaryKey</code>	XPath式を使用して、主キーでレコードを検索します。XPath述語式には、主キーのフィールドのみがすべて含まれている必要があります。フィールドは演算子 <code>and</code> で区切られます。フィールドは、その単純型に応じて次のいずれかの可能性で表されます。

パラメータ	説明
	• <code>date</code>、<code>time</code>、または <code>dateTime</code> 型の場合: <code>date-equal(path, value)</code> を使用します。 • その他の型の場合: <code>path</code>、<code>=</code> 演算子、および <code>value</code> を指定します。 複合主キーの例: <code>./pk1i=1 and date-equal(./pk2d, '2015-11-13')</code> 応答には対応するレコードのみが含まれ、それ以外の場合はエラーが返されます。結果として、他のテーブルパラメーター（フィルター、ビュー公開、ソートなど）は無視されます。 文字列型の値。
<code>pageFirstRecordFilter</code>	5.9.0以降非推奨、 <code>pageRecordFilter</code> に置き換えられました。
<code>pageRecordFilter</code>	レコードを指す、ページネーション用のサーバーサイドで構築されたフィルターを指定します。このフィルターは <code>pageAction</code> 値に強くリンクされており、クライアント側で変更してはなりません。フィルターは、ページネーションコンテキストを特定するために使用されるレコードのXPath述語式の形式を取ります。 文字列型。 参照: • ページネーション

パラメータ	説明
<code>pageAction</code>	<code>pageRecordFilter</code> によって保持される識別子から実行するページネーションアクションを指定します。 文字列型、デフォルト値は <code>first</code> です。指定可能な値は次のとおりです。 • <code>first</code> • <code>previous</code> • <code>next</code> • <code>last</code> 参照: • <code>RequestPagination</code>
<code>pageSize</code>	1ページあたりの最大レコード数を指定します。 整数型、デフォルト値はユーザー設定に基づきます。指定された値が <code>0</code> の場合、推奨される最大ページサイズ（デフォルトは10000）で選択されます。 参照: • ebx.dataservices.pagination.maxSize.default 注記: 指定されたページサイズ値が最大ページサイズを超過する場合、最大ページサイズが選択されます。
<code>sort</code>	操作結果が指定された基準に従ってソートされることを指定します。基準は1つ以上の条件で構成され、結果は左から優先順位に従ってソートされます。条件はフィールドパ

パラメータ	説明
	スト、オプションでソート順序（昇順または降順、値またはラベルによる）で構成されます。このパラメータは以下と組み合わせることができます。 1. <code>sort</code> の後に新しい基準として追加される <code>sortOnLabel</code> パラメータ。 2. <code>sort</code> の後に新しい基準として追加される <code>viewPublication</code> パラメータ。 値の構造は次のとおりです。 <pre><path1>:<order>;...;<pathN>:<order></pre> ここで、 • <code><path1></code> は優先度 <code>1</code> のフィールドパスに対応します。 • <code><order></code> はソート順序に対応し、次のいずれかの値を取ります。 ◦ <code>asc</code> : 値による昇順（デフォルト） ◦ <code>desc</code> : 値による降順 ◦ <code>lasc</code> : ラベルによる昇順 ◦ <code>ldesc</code> : ラベルによる降順 文字列型、デフォルト値は主キーフィールドに従ってソートされます（値による昇順）。 注記: 履歴コード操作値は、このフィールドに関連付けられた <code>meta</code> セクションの <code>ebx-operationCode</code> パスフィールドで使用できます。 注記:

パラメータ	説明
	このパラメーターは、関連性によるソートが有効になっている場合は無視されます。 注記: <code>includeMetadata</code> パラメーター 参照: <code>Request.setSortCriteria</code>
<code>sortByRelevancy</code>	操作結果が関連性によってソートされることを指定します。ただし、以下の条件が満たされている場合に限りです。 クイック検索が、<code>osd:search</code> 述語を フィルターパラメーターに直接指定することで有効になっていること。 - リクエストが履歴テーブルまたはマッピングされたテーブルに適用されていないこと。 文字列型。指定可能な値は次のとおりです。 <code>lasc</code> : ラベルによる昇順 <code>ldesc</code> : ラベルによる降順 関連性によるソートが有効になっている場合、次のパラメーターは無視されます: ソート、<code>sortOnLabel</code>、<code>sortPriority</code>、および <code>viewPublication</code> を通じて定義されたソート基準。 参照: 検索述語

パラメータ	説明
<code>sortOnLabel</code>	操作結果がレコードラベルに従ってソートされることを指定します。このパラメーターは以下と組み合わせることができます。 1. <code>sortOnLabel</code> の前に新しい基準として追加される ソート パラメーター。 2. <code>sortOnLabel</code> の後に新しい基準として追加される <code>viewPublication</code> パラメーター。 値の構造は次のとおりです。 <code><order></code> ここで、 • <code><order></code> はソート順序に対応し、次のいずれかの値を取ります。 ◦ <code>lasc</code> : ラベルによる昇順 ◦ <code>ldesc</code> : ラベルによる降順 このパラメーターの動作は、<code>defaultLabel</code> セクションで説明されています。 文字列型の値。 参照: • 制限 注記: このパラメーターは、関連性によるソート が有効になっている場合は無視されます。
<code>sortPriority</code>	ソートグループのデフォルトの優先度を上書きします。

パラメータ	説明
	• <code>sort</code> • <code>sortOnLabel</code> • <code>sortFromView</code> カンマ区切りの文字列型、デフォルト値は <code>sort,sortOnLabel,sortFromView</code> です。 注記: このパラメーターは、関連性によるソートが有効になっている場合は無視されます。
<code>viewPublication</code>	公開されたビューの名前を指定します。このパラメーターは以下と組み合わせることができます。 1. 論理 <code>and</code> 操作としてのフィルターパラメーター。 2. <code>viewPublication</code> の前に新しい基準として追加されるソートパラメーター。 3. <code>viewPublication</code> の前に新しい基準として追加される <code>sortOnLabel</code> パラメーター。 このパラメーターの動作は、WebコンポーネントとしてのEBX®セクションで説明されています。 文字列型の値。 注記: このパラメーターを通じて定義されたソート基準は、関連性によるソートが有効になっている場合は無視されます。 参照:

パラメータ	説明
	テーブルビュー操作のルックアップ

セレクトパラメーター

以下のパラメーターは、列挙、外部キー、または `osd:resource` を返すフィールドにのみ適用されます（例: `JSON`）。デフォルトでは、ページネーションメカニズムは常に有効です。一部のセレクトの選択操作には入力値が必要です。したがって、`POST` HTTPメソッドのメッセージボディを使用すると、レコードコンテンツを提供できます。

パラメーター	説明
<code>selector</code>	以下を指定します。 <code>true</code> : すべての可能な値（そのラベルを含む）を返します。 <code>false</code> : 現在のフィールドの現在の値を返します。 ブール値型、デフォルト値は <code>false</code> です。 注記: このパラメーターは <code>history</code> カテゴリでは無視されます。
<code>firstElementIndex</code>	返される最初の要素のインデックスを指定します。 <code>0</code> 以上の整数である必要があります。 整数型、デフォルト値は <code>0</code> です。
<code>pageSize</code>	1ページあたりの最大要素数を指定します。 整数型、デフォルト値はユーザー設定に基づきます。指定された値が <code>0</code> の場合、推奨される最大ページサイズ（デフォルトは 10000）で選択されます。

パラメーター	説明
	参照: <code>ebx.dataservices.pagination.maxSize.default</code> 注記: 指定されたページサイズ値が最大ページサイズを超過する場合、最大ページサイズが選択されます。
<code>selectorFilter</code>	セレクターのフィルターを指定します。 文字列型の値、構文はクイック検索に準拠します。

HTTPコード

HTTPコード	説明
<code>200</code> (OK)	選択されたリソースが正常に取得されました。
<code>206</code> (Partial Content)	部分的なコンテンツが返されます。これは、ビジネスオブジェクトカテゴリーで応答のサイズが定義された制限を超過した場合に発生します。 詳細については、<code>ebx.dataservices.rest.bo.maxResponseSizeInKB</code>とパフォーマンスを参照してください。 注記: 部分的なコンテンツ内のシリアル化されたレコードは切り捨てられません。
<code>400</code> (Bad request)	リクエストが不正です。これは次の場合に発生します。

HTTPコード	説明
	- レコードまたはデータセット内の選択されたフィールドがサブターミナルである。 <code>filter</code> パラメーターのXPath述語が不正な形式であるか、フィルターできないノードが含まれている。 <code>primaryKey</code> パラメーターのXPath述語が不正な形式であるか、レコードの主キーではない。 <code>sort</code> パラメーターのソート基準の構文が無効であるか、ソートできないノードが含まれている。 <code>pageAction</code> パラメーター値が許可された値に含まれていない、または次または前のページを選択する際に <code>pageRecordFilter</code> が不正な形式であるか存在しない。 <code>pageSize</code> パラメーター値が2未満である。 <code>viewPublication</code> パラメーターのテーブルビューが階層型である、存在しない、または公開されていない。 <code>selector</code> パラメーターが非列挙ノードに使用されている、または <code>firstElementIndex</code> が負の値であるか、値の数以上である。
<code>403</code> (<i>Forbidden</i>)	選択されたリソースは認証されたユーザーに対して非表示です。
<code>404</code> (<i>Not found</i>)	選択されたリソースが見つかりません。

応答ボディ

データセット、テーブル、レコード、またはフィールドの選択が成功した後、結果は応答ボディで返されます。コンテンツは、提供されたパラメーターと選択されたデータによって異なります。

例: [拡張JSON](#)、[コンパクトJSON](#)。

準備操作

作成または複製操作の準備は、初期コンテンツを持つ新しい一時レコード、または複製するレコードのコンテンツを持つ新しい一時レコードを作成するために使用されます。一時レコードは、まだ永続化されていないコンテンツに対応します。テーブルトリガーによって初期化されたデフォルト値とフィールドが考慮されます。読み取り専用アクセス権限を持つフィールド値のみが返されます。これらの操作は、クライアント側でのデータキャプチャを改善および支援するために、オプションでメタモデルを返すことができます。自動インクリメントフィールドはメタモデルでのみ返されます。セクターパラメーターを有効にすることで、一時レコードのフィールドを照会して、列挙フィールド、外部キーなどの可能な値を取得できます。さらに、セクターの選択操作により、カスタムプログラム列挙のメタモデルの取得などのユースケースを管理するために、[入力レコードデータ](#)を提供できます。

参照:

- `TableTrigger.handleNewContext`

参照:

- [デフォルト値](#)

クライアント側で変更された後、レコードは組み込みの[フォーム挿入操作](#)または[挿入操作](#)を使用して永続化のために送信できます。

これらの準備操作では、次のいずれかのメソッドを使用できます。

- `GET` HTTPメソッド
- メッセージボディなしの `POST` HTTPメソッド
- メッセージボディとオプションの[セッションパラメーター](#)を持つ `POST` HTTPメソッド。

利用可能なURL形式は次のとおりです。

- 作成の準備

- レコード:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/
{dataspace}/{dataset}/{tablePath}:prepareForCreation
```

- レコードフィールド:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/
{dataspace}/{dataset}/{tablePath}:prepareForCreation/{pathInRecord}
```

- 複製の準備

- レコード:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/
{dataspace}/{dataset}/{tablePath}/
{encodedPrimaryKey}:prepareForDuplication
```

- レコードフィールド:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/
{dataspace}/{dataset}/{tablePath}/
{encodedPrimaryKey}:prepareForDuplication/{pathInRecord}
```

ここで、

- `{category}` は操作カテゴリに対応します（指定可能な値は `data` または `data-compact` です）。
- `{dataspace}` は `B` にデータスペース識別子が続くものに対応します。
- `{dataset}` はデータセット識別子に対応します。
- `{tablePath}` はテーブルパスノードに対応します。

- `{encodedPrimaryKey}` は主キーのパーセントエンコードされた表現に対応します (RFC-3986 Uniform Resource Identifierを参照)。
- `{pathInRecord}` はテーブルノードから始まるパスに対応します。

パラメーター

以下のパラメーターは準備操作に適用されます。

パラメーター	説明
<code>includeDetails</code>	メタモデルとコンテンツに <code>details</code> フィールドを含めます。これは、間接的に到達可能な各リソースに対して行われます。返される値は、そのURLリソースに対応します。 ブール値型、デフォルト値は <code>true</code> です。 参照: • <code>includeMetamodel</code>
<code>includeMeta</code>	6.1.0以降非推奨、<code>includeMetamodel</code> に置き換えられました。
<code>includeMetamodel</code>	<code>content</code> フィールドで返される構造の説明を保持する <code>meta</code> フィールドを含めます。 ブール値型、デフォルト値は <code>false</code> です。 参照: • メタモデル
<code>includeSelector</code>	コンテンツに <code>selector</code> フィールドを含めます。これは、間接的に到達可能な各リソースに対して行われます。返される値は、そのURLリソースに対応します。 ブール値型、デフォルト値は <code>true</code> です。

パラメーター	説明
	参照: セレクトター

セレクトターパラメーター

利用可能なパラメーターは、セレクトター操作のパラメーターと同様です。

参照:

- セレクトターパラメーター

HTTPコード

HTTPコード	説明
200 (OK)	選択されたリソースが正常に取得されました。
400 (Bad request)	リクエストが不正です。これは次の場合に発生します。 - レコード内の選択されたフィールドがサブターミナルである。 <code>pageSize</code> パラメーター値が2未満であるか、<code>unbounded</code> とは異なる文字列である。 <code>selector</code> パラメーターが非列挙ノードに使用されている、または <code>firstElementIndex</code> が負の値であるか、値の数以上である。
401 (Unauthorized)	認証に失敗しました。

HTTPコード	説明
403 (Forbidden)	選択されたリソースは認証されたユーザーから非表示です。
404 (Not found)	選択されたリソースが見つかりませんでした。

レスポンスボディ

作成または複製リクエストの準備が成功した後、一時レコードがレスポンスボディで返されます。コンテンツは、提供されたパラメーターと選択されたデータによって異なります。ただし、選択操作レコードと同様の形式を取ります。

例: [拡張JSON](#)、[コンパクトJSON](#)。

挿入操作

挿入操作では `POST` HTTPメソッドを使用します。データを指定するにはボディメッセージが必要です。この操作は、単一のトランザクションで1つ以上のレコードの挿入をサポートします。さらに、パラメーター化によってレコードを更新することも可能です。

- **レコード:** 選択したテーブルに新しいレコードを挿入するか、既存のレコードを変更します
- **レコードテーブル:** 選択したテーブルに1つ以上のレコードを挿入または変更し、一貫性のある応答を確保します。操作はクライアント側で定義された順序で順次実行されます。テーブル操作中にエラーが発生した場合、すべての更新はキャンセルされ、クライアントは詳細情報を含むエラーメッセージを受け取ります

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}[:mass]
```

説明:

- `{category}` は**操作カテゴリ**に対応します (指定可能な値: `data` または `data-compact`)
- `{dataspace}` はデータスペース識別子に続く `B`、またはスナップショット識別子に続く `V` に対応します
- `{dataset}` はデータセット識別子に対応します
- `{pathInDataset}` はテーブルノードのパスに対応します

- `:mass` 拡張アクションは、フィルタリングされたレコードセットを更新する場合に必要です (フィルターパラメーターを参照)

パラメーター

以下のパラメーターは挿入操作に適用可能です。

パラメーター	説明
<code>includeDetails</code>	データ詳細にアクセスするために、コンテンツに <code>details</code> フィールドを含めます。返される値は、そのURLリソースに対応します。 型は <code>Boolean</code> で、デフォルト値は <code>false</code> です。 注記: レコードテーブルにのみ適用されます 参照: • 挿入操作レポート
<code>includeForeignKey</code>	各レコードの応答に <code>foreignKey</code> フィールドを含めます。返される値は、このレコードを参照していた外部キーフィールドの値に対応します。 型は <code>Boolean</code> で、デフォルト値は <code>false</code> です。 注記: レコードテーブルにのみ適用されます 参照: • 挿入操作レポート
<code>includeLabel</code>	各レコードの応答に <code>label</code> フィールドを含めます。

パラメーター	説明
	指定可能な値: • <code>yes</code>: <code>label</code> フィールドが含まれます • <code>no</code>: <code>label</code> フィールドは含まれません (ユースケース: 統合) 型は <code>String</code> で、デフォルト値は <code>no</code> です。 注記: レコードテーブルにのみ適用されます 参照: • 挿入操作レポート
<code>updateOrInsert</code>	挿入するレコードがすでに存在する場合の動作を指定します: • <code>true</code> の場合: 既存のレコードは新しいデータで更新されます。 レコードテーブルに対するリクエストの場合、これが挿入 <code>201</code> か更新 <code>204</code> を指定するために、レポートに <code>code</code> フィールドが追加されます。 • <code>false</code> (デフォルト値) の場合: クライアントエラーが返され、操作は中止されます <code>Boolean</code> 型の値。
<code>byDelta</code>	リクエストボディで定義されていないノードの値を設定する動作を指定します。これは更新モードセクションで説明されています。 型は <code>Boolean</code> で、デフォルト値は <code>true</code> です。

パラメーター	説明
	注記: 更新モードのレコード、および <code>updateOrInsert</code> パラメーターが <code>true</code> の場合に適用されます
<code>filter</code>	クイック検索述語、または多数のレコードが更新される場合の完全なXPath述語式を使用して、更新されるレコードを決定します。リクエストボディは、更新するフィールドを持つレコード構造である必要があります。このパラメーターは、:mass 拡張アクションを使用する必要があります。そうしないと、エラーが返されます。 すべてのレコードを選択するには、ebx-all 値を使用します。 <code>viewPublication</code> クエリパラメーターを使用する場合、スコープはフィルタリングされたビューに限定されます。 <code>String</code> 型の値で、空であってはなりません。空の場合、エラーが返されます。 注記: <code>includeDetails</code>、<code>includeLabel</code>、または <code>includeForeignKey</code> クエリパラメーターを使用する場合、レポートには最初の1000件のレポートエントリのみが返されます
<code>viewPublication</code>	大量更新中に考慮される公開ビューの名前を指定します。 このパラメーターの動作は、WebコンポーネントとしてのEBX® セクションで説明されています。

パラメーター	説明
	<code>String</code> 型の値。
<code>blockingConstraintsDisabled</code>	ブロッキング制約を無視するかどうかを指定します。無視する場合、作成された検証エラーに関係なく操作はコミットされます。そうでない場合、操作は中止されます。 型は <code>Boolean</code> で、デフォルト値は <code>false</code> です。 詳細については、ブロッキング制約と非ブロッキング制約を参照してください。

メッセージボディ

リクエストはメッセージボディを定義する必要があります。形式は挿入されるオブジェクトのタイプによって異なります:

- **レコード:** レコードのヘッダーがないことを除いて、レコードの選択操作と同様です (例: [拡張JSON](#)、[コンパクトJSON](#))
- **レコードテーブル:** ページネーション情報がないことを除いて、テーブルの選択操作と同様です (例: [拡張JSON](#)、[コンパクトJSON](#))

参照:

- [継承](#)

HTTPコード

HTTPコード	説明
<code>200</code> (OK)	リクエストがレコードテーブルに関連する場合。

HTTPコード	説明
	挿入リクエストは正常に適用され、オプションのレポートがレスポンスボディで返されます。
201 (Created)	リクエストがレコードに関連する場合。 新しいレコードが作成され、この場合、ヘッダーフィールド Location がそのリソースURLとともに返されます。
204 (No content)	リクエストがレコードに関連する場合。 updateOrInsert が true の場合にのみ利用可能で、既存のレコードが正常に更新され、この場合、ヘッダーフィールド Location がそのリソースURLとともに返されます。
400 (Bad request)	リクエストが不正です。これは、メッセージボディの構造が メッセージボディ で言及されている内容に準拠していない場合に発生します。
403 (Forbidden)	認証されたユーザーはレコードを作成する権限がないか、リクエストボディに読み取り専用フィールドが含まれています。
404 (Not found)	選択されたリソースが見つかりません。
409 (Conflict)	同時変更。 updateOrInsert が true の場合にのみ利用可能で、 楽観的ロック がアクティブ化されており、その間にコンテンツが変更された場合、更新前に再ロードする必要があります。
422 (Unprocessable entity)	リクエストを処理できません。これは次の場合に発生します:

HTTPコード	説明
	• ブロッキング検証エラーが発生した場合 (<code>blockingConstraintsDisabled</code> が <code>false</code> の場合にのみ利用可能) • 同じ主キーを持つレコードがすでに存在するため、レコードを挿入できない場合 (<code>updateOrInsert</code> が <code>false</code> の場合にのみ利用可能) • 主キーの定義が存在しないか不完全であるため、レコードを挿入できない場合 • 主キーの値を変更できないため、レコードを更新できない場合

レスポンスボディ

レスポンスボディの形式は、挿入されたオブジェクトのタイプによって異なります:

- **レコード:** 操作が正常に実行された場合、空です。ヘッダーフィールド `Location` がそのURLリソースとともに返されます。
- **レコードテーブル:** (オプション) 挿入操作レポートに対応する要素のテーブルが含まれます (例: [JSON](#))。このレポートは、以下のオプションの少なくとも1つが設定されている場合、レスポンスボディに自動的に含まれます:
 - `includeForeignKey`
 - `includeLabel`
 - `includeDetails`

参照:

- [継承](#)

更新操作

この操作により、単一のデータセットまたはレコードの変更が可能です。PUT HTTPメソッドを使用する必要があります。利用可能なURL形式は次のとおりです:

- データセットノード: 選択したノードに含まれる末端ノードの値を変更します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}
```

- レコード: 選択したレコードのコンテンツを変更します。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}
```

注記:

POST HTTPメソッドでも利用可能です。この場合、URLはテーブルを指し、パラメーター `updateOrInsert` は `true` に設定する必要があります

注記:

複数のレコードのコンテンツを変更するには、`updateOrInsert=true` および `byDelta=true` パラメーターを指定して挿入操作を使用します

- フィールド: 選択したレコードの単一フィールドを更新します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
```

注記:

フィールドは末端ノードまたはそれ以上である必要があります

説明:

- `{category}` は操作カテゴリに対応します (指定可能な値: `data` または `data-compact`)
- `{dataspace}` はデータスペース識別子に続く `B`、またはスナップショット識別子に続く `V` に対応します
- `{dataset}` はデータセット識別子に対応します
- `{pathInDataset}` はデータセットノードのパスに対応します:
 - データセットノード操作の場合、これはテーブルノードを除く任意の末端ノードまたはそれ以上である必要があります
 - レコードおよびフィールド操作の場合、これはテーブルノードに対応します
- `{encodedPrimaryKey}` は主キーのパーセントエンコード表現に対応します ([RFC-3986 Uniform Resource Identifier](#)を参照)
- `{pathInRecord}` はテーブルノードから始まるパスに対応します

パラメーター

更新操作に適用可能なパラメーターは次のとおりです。

パラメーター	説明
<code>blockingConstraintsDisabled</code>	ブロッキング制約を無視するかどうかを指定します。無視する場合、作成された検証エラーに関係なく操作はコミットされます。そうでない場合、操作は中止されます。型は <code>Boolean</code> で、デフォルト値は <code>false</code> です。詳細については、 ブロッキング制約と非ブロッキング制約 を参照してください。
<code>byDelta</code>	リクエストボディで定義されていないノードの値を設定する動作を指定します。これは 更新モード セクションで説明されています。型は <code>Boolean</code> で、デフォルト値は <code>true</code> です。
<code>checkNotChangedSinceLastUpdateTime</code>	レコードが最後に読み取られてから変更されていないことを確認するために使用される、日時形式のタイムスタンプです。 楽観的ロック セクションも参照してください。 <code>DateTime</code> 型の値。

メッセージボディ

リクエストはメッセージボディを定義する必要があります。

構造は以下と同じです:

- データセットノード (サンプル [拡張JSON](#)、[コンパクトJSON](#))
- レコード (サンプル [拡張JSON](#)、[コンパクトJSON](#))
- レコードフィールド (サンプル [拡張JSON](#)、[コンパクトJSON](#))

更新されたスコープに応じて、`content` エントリのみを保持します。

参照:

- [継承](#)

HTTPコード

HTTPコード	説明
<code>204</code> (<i>No content</i>)	レコード、フィールド、またはデータセットノードが正常に更新されました。
<code>400</code> (<i>Bad request</i>)	リクエストが不正です。これは、ボディリクエストの構造が準拠していない場合に発生します。
<code>403</code> (<i>Forbidden</i>)	認証されたユーザーは指定されたリソースを更新する権限がないか、リクエストボディに読み取り専用フィールドが含まれています。
<code>404</code> (<i>Not found</i>)	選択されたリソースが見つかりません。
<code>409</code> (<i>Conflict</i>)	同時変更。 楽観的ロック がアクティブ化されており、その間にコンテンツが変更された場合、更新前に再ロードする必要があります。

HTTPコード	説明
422 (<i>Unprocessable entity</i>)	リクエストを処理できません。これは次の場合に発生します: • ブロッキング検証エラーが発生した場合 (<code>blockingConstraintsDisabled</code> が <code>false</code> の場合にのみ利用可能) • 主キーの値を変更できないため、レコードを更新できない場合

削除操作

この操作では `DELETE` HTTPメソッドを使用します。

2つのURL形式が利用可能です:

- **レコード:** URLで指定されたレコードを削除します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/
{dataset}/{pathInDataset}/{encodedPrimaryKey}
```

- **レコードテーブル:** 指定されたテーブル内の複数のレコードを削除し、一貫性のある応答を提供します。このモードでは、レコードテーブルを含むボディメッセージが必要です。削除は、テーブルで定義された順序に従って順次実行されます。テーブル操作中にエラーが発生した場合、すべての削除はキャンセルされ、詳細情報を含むエラーメッセージが表示されます。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/
{dataset}/{pathInDataset}[:mass]
```

説明:

- `{dataspace}` はデータスペース識別子に続く `B`、またはスナップショット識別子に続く `v` に対応します
- `{dataset}` はデータセット識別子に対応します
- `{pathInDataset}` はテーブルノードのパスに対応します

- `{encodedPrimaryKey}` は主キーのパーセントエンコード表現に対応します (RFC-3986 Uniform Resource Identifierを参照)
- `:mass` 拡張アクションは、フィルタリングされたレコードセットを削除する場合に必要です (フィルターパラメーターを参照)

子データセットのコンテキストでは、この操作はレコードの `inheritanceMode` プロパティ値を次のように変更します:

- 継承モードが `inherit` または `overwrite` に設定されているレコードは `occult` になります
- 継承モードが `occult` に設定されているレコードは、`inheritIfInOccultingMode` 操作パラメーターが `true` に設定されているか未定義の場合、`inherit` になります。そうでない場合、変更はありません
- 継承モードが `root` に設定されているレコードは単純に削除されます

参照:

- 継承

ビジネスオブジェクトカテゴリ (`data-bo` および `data-compact-bo`) は、ビジネスオブジェクトリレーションを介して、削除対象レコードに関連するすべてのレコードをカスケード削除します。

パラメーター

削除操作に適用可能なパラメーターは次のとおりです。

パラメーター	説明
<code>includeOcculting</code>	オカルト化されたレコードを含めます。型は <code>Boolean</code> で、デフォルト値は <code>false</code> です。
<code>inheritIfInOccultingMode</code>	バージョン5.8.1以降非推奨。後方互換性のために引き続き利用可能ですが、将来のバージョンで最終的に削除されます。レコードがオカルトモードの場合に継承します。型は <code>Boolean</code> で、デフォルト値は <code>true</code> です。

パラメーター	説明
<code>checkNotChangedSinceLastUpdateTime</code>	レコードが最後に読み取られてから変更されていないことを確認するために使用される、日時形式のタイムスタンプです。 楽観的ロック セクションも参照してください。 <code>DateTime</code> 型の値。
<code>blockingConstraintsDisabled</code>	ブロッキング制約を無視するかどうかを指定します。無視する場合、作成された検証エラーに関係なく操作はコミットされます。そうでない場合、操作は中止されます。型は <code>Boolean</code> で、デフォルト値は <code>false</code> です。詳細については、 ブロッキング制約と非ブロッキング制約 を参照してください。
<code>filter</code>	クイック検索述語、または大量削除が適用されるレコードを定義する完全なXPath述語式。このパラメーターは、 <code>:mass</code> 拡張アクションとともに使用する必要があります。そうしないと、エラーが返されます。すべてのレコードを選択するには、 <code>ebx-all</code> 値を使用します。 viewPublication クエリパラメーターを使用する場合、スコープはフィルタリングされたビューに限定されます。 <code>String</code> 型の値で、空であってはなりません。空の場合、エラーが返されます。
<code>viewPublication</code>	大量削除中に考慮される公開ビューの名前を指定します。このパラメーターの動作は、 WebコンポーネントとしてのEBX®セクション で説明されています。 <code>String</code> 型の値。

メッセージボディ

リクエストは、`filter` または `deleteAll` クエリパラメーターを使用せずに複数のレコードを削除する場合にのみ、メッセージボディを定義する必要があります:

- **レコードテーブル:** メッセージには、レコードに関連する要素のテーブルが含まれ、各要素には次のいずれかのプロパティがあります:
 - `details`: レコードURLに対応し、選択操作によって返されます

- `primaryKey` : XPath式を使用して、レコードの主キーに対応します
- `foreignKey` : 外部キーがレコードを参照する場合に持つ値に対応します

参照:

- `PrimaryKey`

例: 拡張JSON、コンパクトJSON。

HTTPコード

HTTPコード	説明
<code>200</code> (OK)	操作は正常に実行されました。レポートがレスポンスボディで返されます。
<code>400</code> (Bad request)	リクエストが不正です。これは次の場合に発生します: • メッセージボディの構造がメッセージボディに準拠していない場合 • URLがレコードを指定しているにもかかわらず、メッセージボディにレコードテーブルが含まれている場合
<code>403</code> (Forbidden)	認証されたユーザーは指定されたレコードを削除またはオカルト化する権限がありません。
<code>404</code> (Not found)	選択されたレコードが見つかりません。子データセットのコンテキストでは、 <code>includeOcculting</code> パラメーターを使用する必要がある場合があります。
<code>409</code> (Conflict)	同時変更。楽観的ロックがアクティブ化されており、その間にコンテンツが変更された場

HTTPコード	説明
	合、レコードを削除する前に再ロードする必要があります。 パラメーター値 <code>checkNotChangedSinceLastUpdateTime</code> は存在しますが、レコードの実際の最終更新日と一致しません。
<code>422</code> (<i>Unprocessable entity</i>)	<code>blockingConstraintsDisabled</code> が <code>false</code> の場合にのみ利用可能で、ブロッキング検証エラーのため操作が失敗します。

レスポンスボディ

レコードの削除またはオカルト化が成功した後、レポートがレスポンスボディで返されます。これには、削除、オカルト化、および継承されたレコードの数が含まれます。

例: [JSON](#)。

カウント操作

カウント操作では、次のいずれかのメソッドを使用できます:

- `GET` HTTPメソッド
- メッセージボディなしの `POST` HTTPメソッド
- メッセージボディありの `POST` HTTPメソッド (ただしルートに `content` フィールドなし)

URL形式は次のとおりです:

- **データセットノード:** `data` カテゴリは、選択したノードに含まれる末端ノードの数を返します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}:count
```

注記:

`history` カテゴリには適用されません

- テーブル (操作カテゴリによる):

`data` カテゴリはテーブルレコードの数を返します。

`history` カテゴリはテーブル履歴レコードの数を返します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}:count
```

- フィールド (操作カテゴリによる):

`data` カテゴリはレコードフィールドをカウントします。

`history` カテゴリは履歴レコードフィールドをカウントします。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}/
{pathInRecord}:count
```

注記:

フィールドは、関連ノード、選択ノード、末端ノード、またはそれ以上である必要があります

説明:

- `{category}` は操作カテゴリに対応します (指定可能な値: `data` または `data-compact`)
- `{dataspace}` はデータスペース識別子に続く `B`、またはスナップショット識別子に続く `V` に対応します
- `{dataset}` はデータセット識別子に対応します
- `{pathInDataset}` はデータセットノードのパスに対応し、グループノードまたはテーブルノードのいずれかです
- `{encodedPrimaryKey}` は主キーのパーセントエンコード表現に対応します ([RFC-3986 Uniform Resource Identifier](#)を参照)

- `{pathInRecord}` はテーブルノードから始まるパスに対応します

パラメーター

以下のパラメーターはカウント操作に適用可能です。

パラメーター	説明
<code>count</code>	バージョン6.0.0以降非推奨。URLからの拡張アクションに置き換えられました。これがカウント操作か選択操作かを指定するために使用されます。型は <code>Boolean</code> で、デフォルト値は <code>false</code> です。

TIBCO EBX® Documentation - Built-in RESTful services

テーブルパラメーター

以下のパラメーターは、テーブル、関連付け、および選択ノードに適用されます。

パラメーター	説明
<code>filter</code>	クイック検索述語または完全なXPath述語式は、リクエストが適用されるフィールド値を定義します。空の場合、すべてのレコードが考慮されます。 <code>String</code> 型値。 注記: 履歴コード操作値は、このフィールドに関連付けられた <code>meta</code> セクションの <code>ebx-operationCode</code> パスフィールドで使用できます。
<code>historyMode</code>	テーブルに適用されるフィルターコンテキストを指定します。 <code>String</code> 型、可能な値は次のとおりです。

パラメーター	説明
	• <code>CurrentDataSpaceOnly</code> : 現在のデータスペースの履歴 • <code>CurrentDataSpaceAndAncestors</code> : 現在のデータスペースと祖先の履歴 • <code>CurrentDataSpaceAndMergedChildren</code> : 現在のデータスペースとマージされた子孫の履歴 • <code>AllDataSpaces</code> : すべてのデータスペースの履歴 デフォルト値は <code>CurrentDataSpaceOnly</code> です。 参照: • <code>history</code> 注記: このパラメーターは <code>data</code> カテゴリでは無視されます。
<code>includeOcculting</code>	オカルトモードのレコードを含めます。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。
<code>viewPublication</code>	カウント実行中に考慮される公開ビューの名前を指定します。このパラメーターは以下と組み合わせることができます。 • 論理 <code>and</code> 操作としてのフィルターパラメーター このパラメーターの動作は、「EBX®をWebコンポーネントとして使用」セクションで説明されています。

パラメーター	説明
	<code>String</code> 型値。

セレクトーパラメーター

以下のパラメーターは、列挙、外部キー、または `osd:resource` を返すフィールドにのみ適用されます。

パラメーター	説明
<code>selector</code>	以下を指定します。 • <code>true</code> : 可能なすべての値の数を返します • <code>false</code> : 現在のフィールドの可能な値の数を返します <code>Boolean</code> 型、デフォルト値は <code>false</code> です。 注記: このパラメーターは <code>history</code> カテゴリでは無視されます。
<code>selectorFilter</code>	セレクトーのフィルターを指定します。 <code>String</code> 型値、構文はクイック検索に準拠します。

HTTPコード

HTTPコード	説明
<code>200</code> (OK)	選択されたリソースが正常にカウントされました。

HTTPコード	説明
400 (<i>Bad request</i>)	リクエストが不正です。これは次の場合に発生します。 - レコードまたはデータセットで選択されたフィールドがサブターミナルである場合 - 選択されたデータセットフィールドがデータセットツリーである場合 <code>filter</code> パラメーターのクイック検索述語または完全なXPath述語が不正な形式であるか、フィルターできないノードが含まれている場合 <code>viewPublication</code> パラメーターのテーブルビューが階層的であるか、存在しないか、または公開されていない場合 <code>selector</code> パラメーターが非列挙ノードに使用されているか、<code>firstElementIndex</code> が負の値であるか、値の数以上である場合
403 (<i>Forbidden</i>)	認証されたユーザーに対して選択されたリソースが非表示になっています。
404 (<i>Not found</i>)	選択されたリソースが見つかりません。

楽観的ロック

以前に読み取られたが、その間に変更された可能性のあるレコードに対する更新または削除操作を防ぐために、楽観的ロックメカニズムが提供されています。

楽観的ロックを有効にするには、選択リクエストで `system` 値を含む `includeMetadata` パラメーターを設定する必要があります。

詳細については、「[技術データ](#)」を参照してください。

`update_time` プロパティ値は、後続のリクエストに含める必要があります。指定された時刻以降にレコードが変更されている場合、更新または削除操作はキャンセルされます。

- **レコード:** 選択されたレコードの全体または部分的なコンテンツを更新します。

変更されたレコードの更新を防ぐために、`update_time` 値をリクエストボディに追加する必要があります。

レコードのJSON例を参照してください。

- **フィールド:** 選択されたレコードの単一フィールドを更新します。

変更されたレコードの更新を防ぐために、`update_time` 値は

`checkNotChangedSinceLastUpdateTime` パラメーターによってリクエストURLで宣言する必要があります。

`update_time` プロパティ値は、変更されたレコードの削除を防ぐために、リクエストURLの `checkNotChangedSinceLastUpdateTime` パラメーターでも使用できます。

注記:

`checkNotChangedSinceLastUpdateTime` パラメーターは複数回使用できますが、同じレコードに対してのみです。これは、リクエストURLが複数のレコードを返す場合、リクエストが失敗することを意味します。

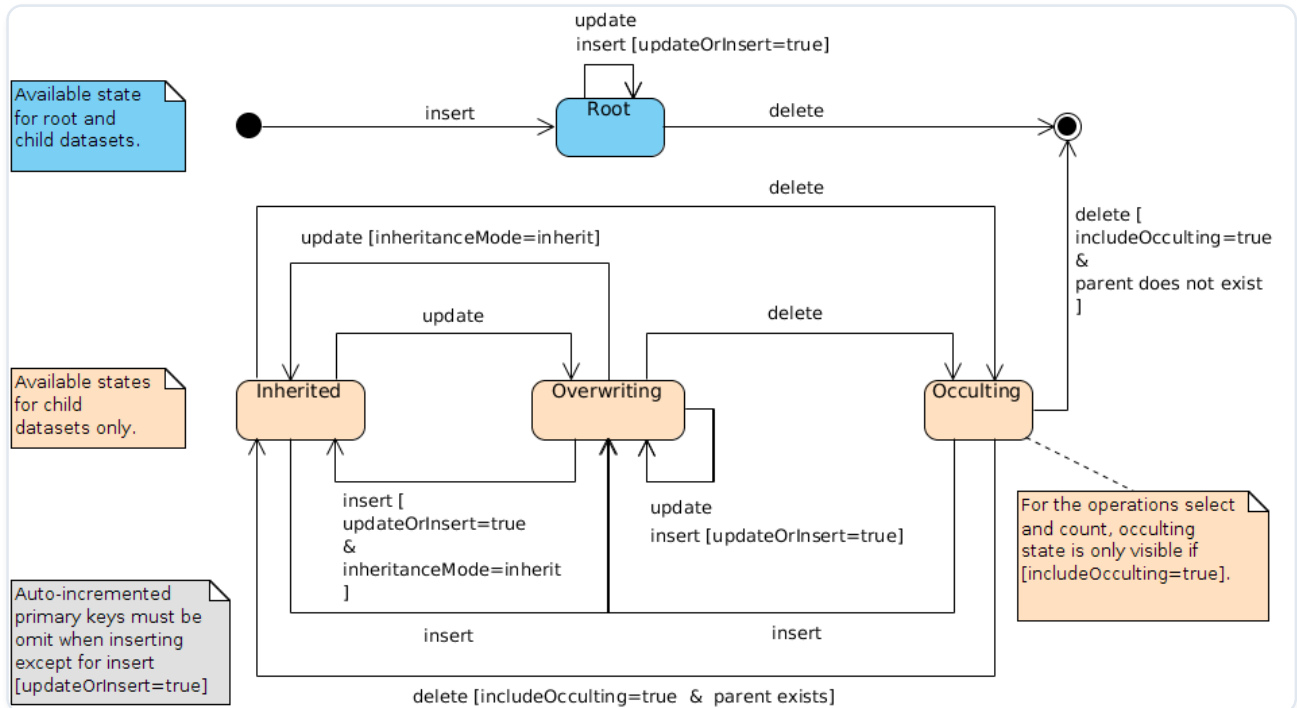
継承

EBX®の継承機能は、特定のプロパティと自動動作を使用して、組み込みのRESTfulサービスによってサポートされています。ほとんどの場合、継承状態はレコードとフィールドの定義またはコンテンツに従ってサーバーによって自動的に計算されます。レコードまたはフィールドを変更するすべてのアクションは、これらの状態に間接的な影響を与える可能性があります。継承ライフサイクルを完全に処理するために、特定の条件下で状態の直接変更が許可されます。禁止または矛盾する明示的な変更試行は無視されます。

参照:

- [継承と値の解決](#)

組み込みRESTfulサービスにおけるレコード継承ライフサイクル



継承プロパティ

以下の表は、EBX®の継承機能に関連するプロパティについて説明しています。

プロパティ	場所	説明
<code>inheritance</code>	レコードまたはテーブルのメタモデル	テーブルに対してデータセット継承がアクティブ化されているかどうかを指定します。この値はデータモデルから計算され、組み込みのRESTfulサービスを通じて変更することはできません。 参照: メタモデルの <code>inheritance</code> プロパティ

プロパティ	場所	説明
<code>inheritedField</code>	フィールドのメタモデル	フィールドの値のソースを指定します。ソースデータはデータモデルから直接取得され、組み込みのRESTfulサービスを通じて変更することはできません。 参照: メタモデルの <code>inheritedField</code> プロパティ
<code>inheritanceMode</code>	子データセットのレコード	レコードの継承状態を指定します。レコードの継承を <code>overwrite</code> から <code>inherit</code> に設定するには、その <code>inheritanceMode</code> 値をリクエストで明示的に指定する必要があります。この特定のケースでは、<code>content</code> プロパティが存在しても無視されます。 <code>occult</code> および <code>root</code> の明示的な値は常に無視されます。<code>content</code> プロパティのない <code>overwrite</code> の明示的な値は無視されます。 注記: 継承されたレコードのフィールドは必然的に <code>inherit</code> です。 注記:

プロパティ	場所	説明
		子データセットのルートレコードは常に <code>root</code> になります。 可能な値は、<code>root</code>、<code>inherit</code>、<code>overwrite</code>、<code>occult</code> です。詳細については、「レコード検索メカニズム」を参照してください。
	<code>overwrite</code> レコードのフィールド	フィールドの継承状態を指定します。フィールドの継承を <code>inherit</code> に設定するには、その <code>inheritanceMode</code> 値をリクエストで明示的に指定する必要があります。この場合、<code>content</code> プロパティは無視されます。<code>content</code> プロパティのない <code>overwrite</code> の明示的な値は無視されます。 注記: フィールドレベルの <code>inheritanceMode</code> は、<code>root</code>、<code>inherit</code>、<code>occult</code> レコード、主キー、および読み取り専用フィールドでは使用できません。 注記: <code>inheritedFieldMode</code> と <code>inheritanceMode</code> プロパティは、同じフィールドに

プロパティ	場所	説明
		両方設定することはできません。 可能な値は、<code>inherit</code>、<code>overwrite</code> です。詳細については、「継承と値の解決」を参照してください。
<code>inheritedFieldMode</code>	継承されたフィールド	継承されたフィールドの継承状態を指定します。 フィールドの継承を <code>inherit</code> に設定するには、その <code>inheritedFieldMode</code> 値をリクエストで明示的に指定する必要があります。この場合、<code>content</code> プロパティは無視されます。 <code>content</code> プロパティのない <code>overwrite</code> の明示的な値は無視されます。 注記: <code>inheritedFieldMode</code> は読み取り専用フィールドには表示されません。 注記: <code>inheritedFieldMode</code> と <code>inheritanceMode</code> プロパティは、同じフィールドに両方設定することはできません。 注記:

プロパティ	場所	説明
		<code>inheritedFieldMode</code> は <code>inheritanceMode</code> プロパティよりも優先されます。可能な値は、 <code>inherit</code> 、 <code>overwrite</code> です。詳細については、「 値検索メカニズム 」を参照してください。

テーブルビュー検索操作

「公開ビューの検索」操作では、以下のいずれかのメソッドを使用できます。

- `GET` HTTPメソッド、または
- メッセージボディなしの `POST` HTTPメソッド

URL形式は次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/
{dataset}/{tablePath: [&#94;:]*}:publishedViews
```

ここで、

- `{dataspace}` は、データスペース識別子に続く `B`、またはイメージ識別子に続く `V` に対応します
- `{dataset}` は、データセット識別子に対応します
- `{tablePath: [^:]*}` は、テーブルパスに対応します

パラメーター

この操作に固有のパラメーターはありません。

HTTPコード

HTTPコード	説明
<code>200</code> (OK)	情報が正常に返されました。

HTTPコード	説明
400 (<i>Bad request</i>)	不正なリクエスト、エラーが含まれています。
401 (<i>Unauthorized</i>)	認証に失敗しました。
403 (<i>Forbidden</i>)	認証されたユーザーに対して、指定されたリソースの読み取り権限が拒否されました。
404 (<i>Not found</i>)	選択されたリソースが見つかりません。

レスポンスボディ

承認されたビュー情報（アクセスURIを含む）のコレクションが含まれています。

例: [JSON](#)

フォームデータ操作

フォームデータカテゴリの操作は、コンテンツの作成または変更時にユーザーフォーム送信の制約に準拠する必要があるデータセットフィールド、レコード、またはレコードフィールドに関するものです。これらはユーザーフォーム管理コンテキストで使用することを目的としています。

操作のリクエストボディは、受信データ検証機能と結果レポートがレスポンスに追加されているという点で、`data` カテゴリの同等の操作と非常によく似ています。データ検証は非アクティブ化できません。`data` カテゴリ操作のパラメーター `blockingConstraintsDisabled` は適用されません。検証フェーズは、`blocksCommit` レベルが `onInsertUpdateOrDelete` または `onUserSubmit-checkModifiedValues` に設定された制約が少なくとも1つ違反された場合に失敗します。

詳細については、「[ブロッキング制約と非ブロッキング制約](#)」を参照してください。

フォーム挿入操作

フォーム挿入は `POST` HTTPメソッドを使用し、データを保持するメッセージボディを必要とします。この操作は、単一のトランザクションで1つまたは複数のレコードをサポートします。さらに、レコードを更新することも可能です。

- **レコード:** 選択されたテーブルに新しいレコードを検証して挿入するか、既存のレコードを変更します。
- **レコードテーブル:** 選択されたテーブルで1つまたは複数のレコードを検証して挿入または変更し、一貫性のあるレスポンスを確保します。操作はクライアント側で定義された順序で順次実

行されます。テーブル操作中にエラーが発生した場合、すべての更新はキャンセルされ、クライアントは詳細情報を含むエラーメッセージを受け取ります。レコードの最大数は100に制限されています。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/
{dataset}/{pathInDataset}[:mass]
```

ここで、

- `{category}` は操作カテゴリに対応します（可能な値は `form-data` または `form-data-compact` です）
- `{dataspace}` は、データスペース識別子に続く `B`、またはスナップショット識別子に続く `V` に対応します
- `{dataset}` は、データセット識別子に対応します
- `{pathInDataset}` は、テーブルノードのパスに対応します
- `:mass` 拡張アクションは、フィルターされたレコードセットを更新する場合に必要です（フィルターパラメーターを参照）

パラメーター

以下のパラメーターは、この挿入操作に適用されます。

パラメーター	説明
<code>includeDetails</code>	データ詳細にアクセスするために、回答に <code>details</code> フィールドを含めます。返される値は、そのURLリソースに対応します。 型は <code>Boolean</code>、デフォルト値は <code>false</code> です。 注記: 複数のレコード挿入にのみ適用されます。
<code>includeForeignKey</code>	各レコードの回答に <code>foreignKey</code> フィールドを含めます。返される値は、このレコード

パラメーター	説明
	を参照していた外部キーフィールドの値に対応します。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。 注記: 複数のレコード挿入にのみ適用されます。
<code>includeLabel</code>	各レコードの回答に <code>label</code> フィールドを含めます。 可能な値は次のとおりです。 • <code>yes</code> : <code>label</code> フィールドが含まれます • <code>no</code> : <code>label</code> フィールドは含まれません (ユースケース: 統合) <code>String</code> 型、デフォルト値は <code>no</code> です。 注記: 複数のレコード挿入にのみ適用されます。
<code>updateOrInsert</code>	挿入するレコードがすでに存在する場合の動作を指定します。 • <code>true</code> の場合: 既存のレコードは新しいデータで更新されます • <code>false</code> の場合: クライアントエラーが返され、操作は中止されます <code>Boolean</code> 型、デフォルト値は <code>false</code> です。
<code>filter</code>	多数のレコードが更新される場合に、クイック検索述語または完全なXPath述語式を使用して更新されるレコードを決定します。リクエストボディは、更新するフィールドを

パラメーター	説明
	持つレコード構造である必要があります。このパラメーターは:mass拡張アクションを使用する必要があります。そうでない場合、エラーが返されます。 すべてのレコードを選択するには、ebx-all 値を使用します。 viewPublication クエリパラメーターが使用されている場合、スコープはフィルターされたビューに限定されます。 空であってはならない String 型値。空の場合、エラーが返されます。 注記: includeDetails、includeLabel、または includeForeignKey クエリパラメーターを使用する場合、レポートには最初の1000件のレポートエントリのみが返されます。
viewPublication	大量更新中に使用される公開ビューの名前を指定します。 このパラメーターの動作は、「EBX®をWebコンポーネントとして使用」セクションで説明されています。 String 型値。

メッセージボディ

リクエストはメッセージボディを定義する必要があります。形式は、**data** カテゴリの挿入操作のメッセージボディに似ています。

HTTPコード

HTTPコード	説明
200 (OK)	- リクエストボディに複数のレコードが含まれている場合: 挿入/更新リクエストは正常に適用され、レポートがレスポンスボディで返されます。 - リクエストボディに1つのレコードのみが含まれ、<code>updateOrInsert</code> が <code>true</code> の場合: 更新リクエストは正常に適用され、レポートがレスポンスボディで返されます。
201 (Created)	リクエストボディに既存でない1つのレコードのみが含まれている場合: 新しいレコードが作成され、ヘッダーフィールド <code>Location</code> がそのリソースURLとともに返されます。さらに、レポートがレスポンスボディで返されます。
400 (Bad request)	リクエストが不正です。これは、ボディメッセージ構造が次の場合に発生します。 - メッセージボディで言及された内容に準拠していない場合 - メッセージボディで言及された内容に準拠しているが、レコードの挿入/更新数が100を超える場合
403 (Forbidden)	認証されたユーザーはレコードを作成することを許可されていないか、リクエストボディに読み取り専用フィールドが含まれています。
404 (Not found)	選択されたリソースが見つかりません。

HTTPコード	説明
409 (Conflict)	同時変更。 <code>updateOrInsert</code> が <code>true</code> の場合にのみ利用可能で、 楽観的ロック がアクティブ化されており、その間にコンテンツが変更された場合、更新前に再ロードする必要があります。
422 (Unprocessable entity)	リクエストを処理できません。これは次の場合に発生します。 • ブロッキング制約に違反した場合。この場合、適切なレポートがレスポンスボディで返されます。 • 同じ主キーを持つ別のレコードがすでに存在するため、レコードを挿入できない場合（ <code>updateOrInsert</code> が <code>false</code> の場合にのみ利用可能） • 主キーの定義が存在しないか不完全であるため、レコードを挿入できない場合 • 主キーの値を変更できないため、レコードを更新できない場合

レスポンスボディ

レスポンスボディには常に検証レポートが含まれます。ただし、失敗した場合は、レスポンスボディはJSONの `Exception handling` レスポンスに対応します。

- **レコード:** ヘッダーフィールド `Location` がそのURLリソースとともに返されます。検証レポートはレスポンスボディに含まれます。
- **レコードテーブル:** (オプション) 各要素の検証レポートのリストが含まれ、複数の検証レポートに対応します。

参照:

- フォームデータカテゴリ

フォーム更新操作

`data` または `data-compact` カテゴリの更新操作と同様に、単一のデータセットまたはレコードの変更を許可し、`PUT` HTTPメソッドを使用する必要があります。利用可能なURL形式は次のとおりです。

- **データセットノード:** 選択されたノードに含まれるターミナルノードの値を検証および変更します。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}
```

- **レコード:** 選択されたレコードのコンテンツを検証および変更します。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}
```

注記:

`POST` HTTPメソッドでも利用可能です。この場合、URLはテーブルを指し、パラメーター `updateOrInsert` は `true` に設定する必要があります。

- **フィールド:** 選択されたレコードの単一フィールドを検証および更新します。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
```

注記:

フィールドはターミナルノードであるか、それ以上である必要があります。

ここで、

- `{category}` は操作カテゴリに対応します（可能な値は `form-data` または `form-data-compact` です）
- `{dataspace}` は、データスペース識別子に続く `B`、またはスナップショット識別子に続く `V` に対応します
- `{dataset}` は、データセット識別子に対応します
- `{pathInDataset}` は、データセットノードのパスに対応します。
 - データセットノード操作の場合、これはテーブルノードを除く任意のターミナルノードまたはそれ以上である必要があります
 - レコードおよびフィールド操作の場合、これはテーブルノードに対応します
- `{encodedPrimaryKey}` は、主キーのパーセントエンコードされた表現に対応します（RFC-3986 Uniform Resource Identifierを参照）
- `{pathInRecord}` は、テーブルノードから始まるパスに対応します

パラメーター

更新操作に適用されるパラメーターは次のとおりです。

パラメーター	説明

TIBCO EBX® ドキュメント - 組み込みRESTfulサービス | `byDelta` | リクエストボディで定義されていないノードの値を設定する動作を指定します。これは、「更新モード」セクションで説明されています。 `Boolean` 型で、デフォルト値は `true` です。 ||

`checkNotChangedSinceLastUpdateTime` | レコードが最終読み取り以降に変更されていないことを確認するために使用される日時形式のタイムスタンプです。また、「楽観的ロック」セクションも参照してください。 `DateTime` 型の値です。 |

メッセージボディ

リクエストはメッセージボディを定義する必要があります。形式は `data` カテゴリの更新操作の「メッセージボディ」と同じです。

HTTPコード

HTTPコード	説明
200 (Ok)	更新リクエストは正常に適用され、レポートがレスポンスボディで返されます。
400 (Bad request)	リクエストが不正です。これは、ボディリクエストの構造が準拠していない場合に発生します。
403 (Forbidden)	認証済みユーザーは指定されたリソースを更新する権限がないか、リクエストボディに読み取り専用フィールドが含まれています。
404 (Not found)	選択されたリソースが見つかりません。
409 (Conflict)	同時変更が発生しました。 楽観的ロック がアクティブ化されており、その間にコンテンツが変更されたため、更新前に再読み込みする必要があります。
422 (Unprocessable entity)	リクエストを処理できません。これは次の場合に発生します。 • ブロッキング制約に違反しました。この場合、適切なレポートがレスポンスボディで返されます。 • 主キーの値を変更できないため、レコードを更新できません。

レスポンスボディ

レスポンスボディは、「**フォーム挿入操作**」と同じ形式と動作を持ちます。

データセット操作

これらの操作は、データセットに対する選択を実行します。

参照:

- データセット

データセットの選択

選択操作では、`GET` または `POST` メソッドを使用できます。

注記:

`POST` が使用される場合、メッセージボディは常に見捨てられます。

URL形式は次のとおりです。

- **ルート:** 指定されたデータスペースのルートデータセットを返します。

ページネーションメカニズムは常に有効です。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/  
{dataspace}
```

- **子:** 指定されたデータセットの子データセットを返します。

ページネーションメカニズムは常に有効です。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/  
{dataspace}/{dataset}:children
```

- **情報:** データセット情報を返します。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/  
{dataspace}/{dataset}:information
```

ここで、

- `{category}` は「**操作カテゴリ**」に対応します（指定可能な値は `data` または `data-compact` です）。
- `{dataspace}` は、データスペースの識別子に続く `B`、またはスナップショットの識別子に続く `V` に対応します。
- `{dataset}` はデータセット識別子に対応します。

パラメーター

以下のクエリパラメーターは、ルートおよび子操作に適用されます。

パラメーター	説明
<code>includeLabel</code>	詳細については、「 includeLabel 」を参照してください。
<code>includeDetails</code>	詳細については、「 includeDetails 」を参照してください。
<code>includeOpenApiDetails</code>	詳細については、「 includeOpenApiDetails 」を参照してください。
<code>includeValidation</code>	詳細については、「 includeValidation 」を参照してください。
<code>includeHistory</code>	詳細については、「 includeHistory 」を参照してください。
<code>firstElementIndex</code>	詳細については、「 firstElementIndex 」を参照してください。
<code>pageSize</code>	詳細については、「 pageSize 」を参照してください。

以下のクエリパラメーターは、**情報**操作に適用されます。

パラメーター	説明
<code>includeLabel</code>	詳細については、「 includeLabel 」を参照してください。

HTTPコード

HTTPコード	説明
200 (OK)	リクエストは正常に処理されました。
400 (Bad request)	リクエストが不正です。これは次の場合に発生します。 • <code>firstElementIndex</code> パラメーターの値が不正な形式であるか、範囲外です。 • <code>pageSize</code> パラメーターの値が不正な形式であるか、範囲外です。
401 (Unauthorized)	認証に失敗しました。
403 (Forbidden)	選択されたリソースは認証済みユーザーから非表示になっています。
404 (Not found)	選択されたリソースが見つかりませんでした。

レスポンスボディ

選択が成功すると、結果がレスポンスボディで返されます。コンテンツは、指定されたパラメーターと選択されたデータによって異なります。

形式は選択されたオブジェクトタイプにリンクされています。

- ルートおよび子の場合、「[拡張JSON](#)」および「[コンパクトJSON](#)」の例を参照してください。
- 情報の場合、「[拡張JSON](#)」および「[コンパクトJSON](#)」の例を参照してください。

データスペース操作

これらの操作は、データスペースに対する選択またはライフサイクル管理を実行します。

参照:

- データスペース

ベータ機能: データスペースまたはスナップショットの選択

選択操作では、`GET` または `POST` メソッドのいずれかを使用できます。ページネーションメカニズムは常に有効です。

URL形式は次のとおりです。

- ルート: ルートデータスペースを返します

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/
```

- 子: 指定されたデータスペースの子データスペースを返します

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/  
{dataspace}:children
```

- スナップショット: 指定されたデータスペースのスナップショットを返します

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/  
{dataspace}:snapshots
```

- 情報: データスペースまたはスナップショットの情報を返します

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/  
{dataspace}:information
```

ここで、

- `{dataspace}` は、データスペースの識別子に続く `B`、またはスナップショットの識別子に続く `V` に対応します。

パラメーター

以下のクエリパラメーターは、ルート、子、およびスナップショット操作に適用されます。

パラメーター	説明
<code>includeClosed</code>	クローズされたデータスペースを選択に含めます。 <code>Boolean</code> 型で、デフォルト値は <code>false</code> です。
<code>includeAdministration</code>	管理データスペースを選択に含めます。 <code>Boolean</code> 型で、デフォルト値は <code>false</code> です。
<code>pageRecordFilter</code>	ページネーションのために、レコードを指すサーバー側で構築されたフィルターを指定します。このフィルターは <code>pageAction</code> 値に強くリンクされており、クライアント側で変更すべきではありません。フィルターは、ページネーションコンテキストを特定するために使用されるレコードのXPath述語式の形式を取ります。 <code>String</code> 型です。 参照: ページネーション
<code>pageAction</code>	<code>pageRecordFilter</code> によって保持される識別子から実行するページネーションアクションを指定します。 <code>String</code> 型で、デフォルト値は <code>first</code> です。指定可能な値は次のとおりです。 <code>first</code> <code>previous</code> <code>next</code>

パラメーター	説明
	• <code>last</code> 参照: • <code>RequestPagination</code>
<code>pageSize</code>	1ページあたりの最大レコード数を指定します。 <code>Integer</code> 型で、デフォルト値はユーザー設定に基づきます。値は <code>2</code> から <code>100</code> の間である必要があります。

HTTPコード

HTTPコード	説明
<code>200</code> (OK)	リクエストは正常に処理されました。
<code>400</code> (Bad request)	リクエストが不正です。これは次の場合に発生します。 • <code>pageAction</code> パラメーターの値が許可された値に含まれていません。 • 次または前のページを選択する際に、<code>pageRecordFilter</code> が不正な形式であるか、存在しません。 • <code>pageSize</code> パラメーターの値が範囲外です。
<code>401</code> (Unauthorized)	認証に失敗しました。
<code>403</code> (Forbidden)	選択されたリソースは認証済みユーザーから非表示になっています。

HTTPコード	説明
404 (Not found)	選択されたリソースが見つかりませんでした。

レスポンスボディ

選択が成功すると、結果がレスポンスボディで返されます。コンテンツは、指定されたパラメーターと選択されたデータによって異なります。

形式は選択されたオブジェクトタイプにリンクされています。

- ルート、子、またはスナップショットの場合、「JSON」の例を参照してください。
- 情報の場合、「JSON」の例を参照してください。

ベータ機能: 子データスペースまたはスナップショットの作成

指定されたデータスペースまたはスナップショットを作成します。この操作は、ボディリクエスト（特定のクエリパラメーターなし）とともに `POST` メソッドを使用します。

参照:

- `Repository.createHome`

URL形式は次のとおりです。

- データスペース:

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:createDataspace
```

- スナップショット:

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:createSnapshot
```

ここで、

- `{dataspace}` は、データスペース識別子に続く `B` に対応します。

リクエストボディ

ボディは、作成するデータスペースまたはスナップショットの機能を指定します。

参照:

- `HomeCreationSpec`

「JSON」の例を参照してください。

HTTPコード

HTTPコード	説明
<code>201</code> (<i>Created</i>)	新しいデータスペースまたはスナップショットが作成されました。この場合、ヘッダーフィールド <code>Location</code> がそのリソースURLとともに返されます。
<code>400</code> (<i>Bad request</i>)	リクエストボディが整形形式でないか、無効なコンテンツが含まれています。
<code>401</code> (<i>Unauthorized</i>)	認証に失敗しました。
<code>403</code> (<i>Forbidden</i>)	認証済みユーザーに対して、指定されたリソースを作成する権限が拒否されました。
<code>404</code> (<i>Not found</i>)	リクエストボディで指定されたリソースが見つかりません。
<code>500</code> (<i>Internal error</i>)	アプリケーションによって予期せぬエラーがスローされました。エラーの詳細は通常、EBX® ログで見つけることができます。

ベータ機能: データスペースのロック

指定されたデータスペースをロックします。データスペースがすでにロックされている場合:

- 同じユーザーによってロックされている場合、ロックは保持されます。
- 別のユーザーによってロックされており、待機時間中にロックが取得された場合、ロックは取得されます。

- それ以外の場合、ロックは拒否されます。

この操作は `POST` メソッドを使用し、`Content-Type` ヘッダーを次のように設定して消費します。

- `application/x-www-form-urlencoded` : ボディにHTTPパラメーターを含めるか、
- `application/json` : URLにHTTPパラメーターを含め、JSONボディに「セッションパラメーター」を含めます。

成功した場合、レスポンスボディは返されません。

参照:

- データスペースのロックまたはロック解除の権限
- `LockSpec.lock`

URL形式は次のとおりです。

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:lock
```

ここで、

- `{dataspace}` は、データスペース識別子に続く `B` に対応します。

パラメーター

以下のクエリパラメーターがこの操作に適用されます。

パラメーター	説明
<code>durationToWaitForLock</code>	データスペースが別のユーザーによってロックされている場合、ロックを取得するために待機する最大期間を指定します。期間は秒単位で指定されます。値が <code>0</code> に設定されている場合、ロック試行は直ちに実行されます。いくつかの理由により、待機期間は <code>60</code> 秒を超えることはできません。そうでない場合、値は

パラメーター	説明
	最大値で上書きされます。 <code>Integer</code> 型で、デフォルト値は <code>0</code> です。

HTTPコード

HTTPコード	説明
<code>204</code> (No content)	リクエストは正常に処理されましたが、レスポンスボディは返されません。
<code>400</code> (Bad request)	リクエストURLまたはボディが整形形式でないか、無効なコンテンツが含まれています。
<code>401</code> (Unauthorized)	認証に失敗しました。
<code>403</code> (Forbidden)	認証済みユーザーに対して、指定されたリソースをロックする権限が拒否されました。
<code>404</code> (Not found)	URLで指定されたリソースが見つかりません。
<code>409</code> (Conflict)	リソースはすでに別のユーザーによってロックされており、 <code>durationToWaitForLock</code> パラメーターの値の後にロックが拒否されます。
<code>500</code> (Internal error)	アプリケーションによって予期せぬエラーがスローされました。エラーの詳細は通常、EBX® ログで見つけることができます。

ベータ機能: データスペースのロック解除

指定されたデータスペースのロックを解除します。データスペースが次の場合:

- 同じユーザーによってロックされている場合、ロックは解除されます。
- 別のユーザーによってロックされており、現在のユーザーが管理者であり、`forceByAdministrator` クエリパラメーターが `true` に設定されている場合、ロックは解除されます。
- ロックされていない場合、ロックステータスは変更されません。
- それ以外の場合、ロック解除は拒否されます。

この操作は `POST` メソッドを使用し、`Content-Type` ヘッダーを次のように設定して消費します。

- `application/x-www-form-urlencoded` : ボディにHTTPパラメーターを含めるか、
- `application/json` : URLにHTTPパラメーターを含め、JSONボディに「セッションパラメーター」を含めます。

成功した場合、レスポンスボディは返されません。

参照:

- データスペースのロックまたはロック解除の権限
- `LockSpec.unlock`

URL形式は次のとおりです。

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:unlock
```

ここで、

- `{dataspace}` は、データスペース識別子に続く `B` に対応します。

パラメーター

以下のクエリパラメーターがこの操作に適用されます。

パラメーター	説明

TIBCO EBX® ドキュメンテーション - 組み込み RESTful サービス | `forceByAdministrator` | データスペースが他のユーザーによってロックされている場合、管理者はロック解除を強制できません。 `Boolean` 型、デフォルト値は `false` です。 |

HTTP コード

HTTP コード	説明
204 (コンテンツなし)	リクエストは正常に処理されましたが、レスポンスボディは返されません。
400 (不正なリクエスト)	リクエスト URL またはボディの形式が正しくないか、無効なコンテンツが含まれています。
401 (認証なし)	認証に失敗しました。
403 (禁止)	認証されたユーザーに対して、指定されたリソースのロック解除が拒否されました。
404 (見つかりません)	URL で指定されたリソースが見つかりません。
409 (競合)	リソースがすでに他のユーザーによってロックされているか、現在のユーザーが管理者であるにもかかわらず <code>forceByAdministrator</code> パラメーターが <code>false</code> のため、ロック解除が拒否されました。
500 (内部エラー)	アプリケーションによって予期しないエラーがスローされました。エラーの詳細は通常、EBX® ログで確認できます。

ベータ機能: データスペースのマージ

指定されたデータスペースをその親にマージします。マージ後に履歴および/またはデータを削除することが可能です。

この操作は `POST` メソッドを使用し、`Content-Type` ヘッダーを以下に設定して消費します。

- `application/x-www-form-urlencoded` : HTTP パラメーターをボディに含めるか、
- `application/json` : HTTP パラメーターを URL に含め、**セッションパラメーター**を JSON ボディに含めます。

成功した場合、レスポンスボディは返されません。

注記:

データサービスを介して実行されるマージの場合、マージ決定ステップはバイパスされます。このような場合、子データスペースのデータは自動的に親のデータを上書きします。

参照:

- `ProcedureContext.doMergeToParent`
- データと履歴の削除

URL 形式は次のとおりです。

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:merge
```

ここで、

- `{dataspace}` は `B` の後にデータスペース識別子が続きます。

パラメーター

以下のクエリパラメーターがこの操作に適用されます。

パラメーター	説明
<code>deleteHistoryOnMerge</code>	マージ時に指定されたデータスペースに関連付けられた履歴を削除するかどうかを設定します。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。このパラメーターがリクエストで指定されていない場合、デフォルト値は <code>false</code> です。デフォルト値は、 EBX® メイン設定ファイル でプロパティ <code>ebx.dataservices.historyDeletionOnCloseOrMerge.def</code> を指定することで再定義できます。
<code>deleteDataOnMerge</code>	マージ時に指定されたデータスペースとその関連スナップショットを削除するかどうかを設定します。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。このパラメーターがリクエストで指定されていない場合、デフォルト値は

パラメーター	説明
	<code>false</code> です。デフォルト値は、EBX® メイン設定ファイルでプロパティ <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code>を指定することで再定義できます。

HTTP コード

HTTP コード	説明
<code>204</code> (コンテンツなし)	リクエストは正常に処理されましたが、レスポンスボディは返されません。
<code>400</code> (不正なリクエスト)	リクエスト URL に無効なコンテンツが含まれています。
<code>401</code> (認証なし)	認証に失敗しました。
<code>403</code> (禁止)	認証されたユーザーに対して、指定されたりソースのマージが拒否されました。
<code>404</code> (見つかりません)	URL で指定されたりソースが見つかりません。
<code>500</code> (内部エラー)	アプリケーションによって予期しないエラーがスローされました。エラーの詳細は通常、EBX® ログで確認できます。

ベータ機能: データスペースまたはスナップショットのクローズ

指定されたデータスペースまたはスナップショットをクローズします。クローズ後に履歴および/またはデータを削除することが可能です。

この操作は `POST` メソッドを使用し、`Content-Type` ヘッダーを以下に設定して消費します。

- `application/x-www-form-urlencoded` : HTTP パラメーターをボディに含めるか、
- `application/json` : HTTP パラメーターを URL に含め、[セッションパラメーター](#)を JSON ボディに含めます。

成功した場合、レスポンスボディは返されません。

参照:

- `Repository.closeHome`
- データと履歴の削除

URL 形式は次のとおりです。

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:close
```

ここで、

- `{dataspace}` は `B` の後にデータスペース識別子が続くか、`V` の後にスナップショット識別子が続きます。

パラメーター

以下のクエリパラメーターがこの操作に適用されます。

パラメーター	説明
<code>deleteHistoryOnClose</code>	クローズ時に指定されたデータスペースに関連付けられた履歴を削除するかどうかを設定します。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。このパラメーターがリクエストで指定されていない場合、デフォルト値は <code>false</code> です。デフォルト値は、 EBX® メイン設定ファイル でプロパティ <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> を指定することで再定義できます。
<code>deleteDataOnClose</code>	クローズ時に指定されたデータスペースとその関連スナップショットを削除するかどうかを設定します。 <code>Boolean</code> 型、デフォルト値は <code>false</code> です。このパラメーターがリクエストで指定されていない場合、デフォルト値は <code>false</code> です。デフォルト値は、 EBX® メイン設定ファイル でプロパティ

パラメーター	説明
	<code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> を指定することで再定義できます。

HTTP コード

HTTP コード	説明
204 (コンテンツなし)	リクエストは正常に処理されましたが、レスポンスボディは返されません。
400 (不正なリクエスト)	リクエスト URL またはボディの形式が正しくないか、無効なコンテンツが含まれています。
401 (認証なし)	認証に失敗しました。
403 (禁止)	認証されたユーザーに対して、指定されたリソースのクローズが拒否されました。
404 (見つかりません)	URL で指定されたリソースが見つかりません。
422 (処理できないエンティティ)	ブロッキング制約に違反しました。
500 (内部エラー)	アプリケーションによって予期しないエラーがスローされました。エラーの詳細は通常、EBX® ログで確認できます。

ヘルス操作

概要

ヘルス操作は EBX® サーバーの監視に役立ちます。

開始済み操作

EBX® が開始され初期化されている場合はステータスコード 200 (OK) を返し、それ以外の場合は 503 (サービス利用不可) を返します。

URL 形式は次のとおりです。


```
http[s]://<host>[:<port>]/.../health/v1/started
```

HTTP コード

HTTP コード	説明
200 (OK)	リクエストは正常に処理され、レスポンスボディが返されます。
503 (サービス利用不可)	EBX® はまだ開始および初期化されていません。診断に役立つメッセージが返されます。

レスポンスボディ

JSON の例を参照してください。

チェック操作

EBX® が開始されチェックされている場合はステータスコード 200 (OK) を返し、それ以外の場合は 503 (サービス利用不可) を返します。

URL 形式は次のとおりです。

```
http[s]://<host>[:<port>]/.../health/v1/check
```

HTTP コード

HTTP コード	説明
200 (OK)	リクエストは正常に処理され、レスポンスボディが返されます。
503 (サービス利用不可)	EBX® はまだ開始および初期化されていません。診断に役立つメッセージが返されます。

レスポンスボディ

JSON の例を参照してください。

停止済み操作

EBX® リポジトリがシャットダウンされている場合はステータスコード 200 (OK) を返し、それ以外の場合は 503 (サービス利用不可) を返します。

URL 形式は次のとおりです。

```
http[s]://<host>[:<port>]/.../health/v1/stopped
```

HTTP コード

HTTP コード	説明
200 (OK)	リクエストは正常に処理され、レスポンスボディが返されます。
503 (サービス利用不可)	EBX® はまだ開始および初期化されていません。診断に役立つメッセージが返されます。

レスポンスボディ

JSON の例を参照してください。

OpenAPI 操作

概要

api カテゴリの操作は、OpenAPI 仕様 3.0.X に準拠して JSON ドキュメントを生成します。これらのドキュメントは、利用可能な REST 組み込みリソースとそれらに関連する操作を構造化および記述することで、開発と利用を容易にします。

OpenAPI ドキュメントの HATEOAS リンクは、[includeOpenApiDetails](#) クエリパラメーターを使用する際に、選択データ操作を通じて利用できます。

REST OpenAPI サービスの権限は、[グローバル権限](#)でグローバルに定義されます。

特定の言語でクライアントを生成するために、適切な型マッピングを行うことが不可欠です。

[OpenAPI 形式](#)は、入力の検証や、選択したプログラミング言語での特定の値へのマッピングに使用できます。[単純型のコンテンツ](#)の詳細については、こちらを参照してください。

注記:

特定の形式をサポートしないツールは、`type` のみにフォールバックする場合があります。

OpenAPI ドキュメントの生成

この操作は、`GET` または `POST` HTTP メソッドを使用して、データセット、テーブル、またはスキーマノードの OpenAPI ドキュメントを生成します。

URL 形式は次のとおりです。

- データセット:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/{category}/v1/  
{dataspace}/{dataset}
```

- テーブルとスキーマノード:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/{category}/v1/  
{dataspace}/{dataset}/{pathInDataset}
```

ここで、

- `{category}` は、操作カテゴリのうち、データ、データコンパクト、またはフォームデータのいずれかに対応します。
- `{dataspace}` は `B` の後にデータスペース識別子が続くか、`V` の後にスナップショット識別子が続きます。
- `{dataset}` はデータセット識別子に対応します。
- `{pathInDataset}` は次のパスに対応します。
 - テーブルノード
 - データセットの終端ノードまたはそれ以上

注記:

生成されるドキュメントはユーザー権限に依存しません。すべてのフィールドが表示されます。

HTTP コード

HTTP コード	説明
200 (OK)	リクエストは正常に処理されました。
401 (認証なし)	認証に失敗しました。
403 (禁止)	指定されたリソースの読み取りが拒否されました。
404 (見つかりません)	URL で指定されたリソースが見つかりません。

ステージング操作

ステージング REST API は、ソースサーバーとターゲットサーバーとの対話用に設計されています。これにより、ユーザーはステージングアーカイブのエクスポートとインポートを使用してドメインを管理できます。

サービスの完全な説明は、以下のリンクから OpenAPI 経由で利用できます。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/staging/v1
```

開発実行モードでは、Swagger UI が以下のリンクで利用できます。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/staging/v1/ui
```

参照:

- ビジュアルドキュメンテーション

ドメイン管理操作

ドメインサービスにより、ユーザーはドメインを管理できます。提供される機能は次のとおりです。

- 利用可能なドメインのリストを選択
- 一意の名前でドメインを選択
- 新しいドメインを作成

- ドメインのコンポーネントを選択
- ドメインにコンポーネントを追加
- ドメインからコンポーネントを削除 (コンポーネントは削除されます)

ステージング要素ナビゲーション操作

要素サービスは、インスタンスのすべての要素を階層的に表示するように設計されています。主に、レベル処理を伴う階層ビューをサポートする UI インターフェース向けに設計されています。構造化されたボディの代わりにクエリパラメーターを使用する目的は、ナビゲーションを容易にするためにリンクを使用することです。提供される機能は次のとおりです。

- 要素のコンテンツを展開
- 選択した要素の下にある特定の種類の要素のコンテンツを展開

アーカイブ操作

アーカイブサービスは、ステージングアーカイブを実行中のインスタンスとの間で単一のリクエストでインポート/エクスポートするために設計されています。また、非同期インポートのためにアーカイブをアップロードするためにも使用されます。提供される機能は次のとおりです。

- ドメインのアーカイブをエクスポート
- アーカイブを実行中のインスタンスにインポート
- 非同期インポートのためにアーカイブをアップロード

cURL を介してステージングアーカイブをエクスポートする最も簡単な方法

```
curl --request GET 'http[s]://<host>[:<port>]/ebx-dataservices/rest/staging/v1/archives/<domain>?login=<login>&password=<password>'
```

cURL を介してステージングアーカイブをインポートする最も簡単な方法

```
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>'
```

エクスポート操作

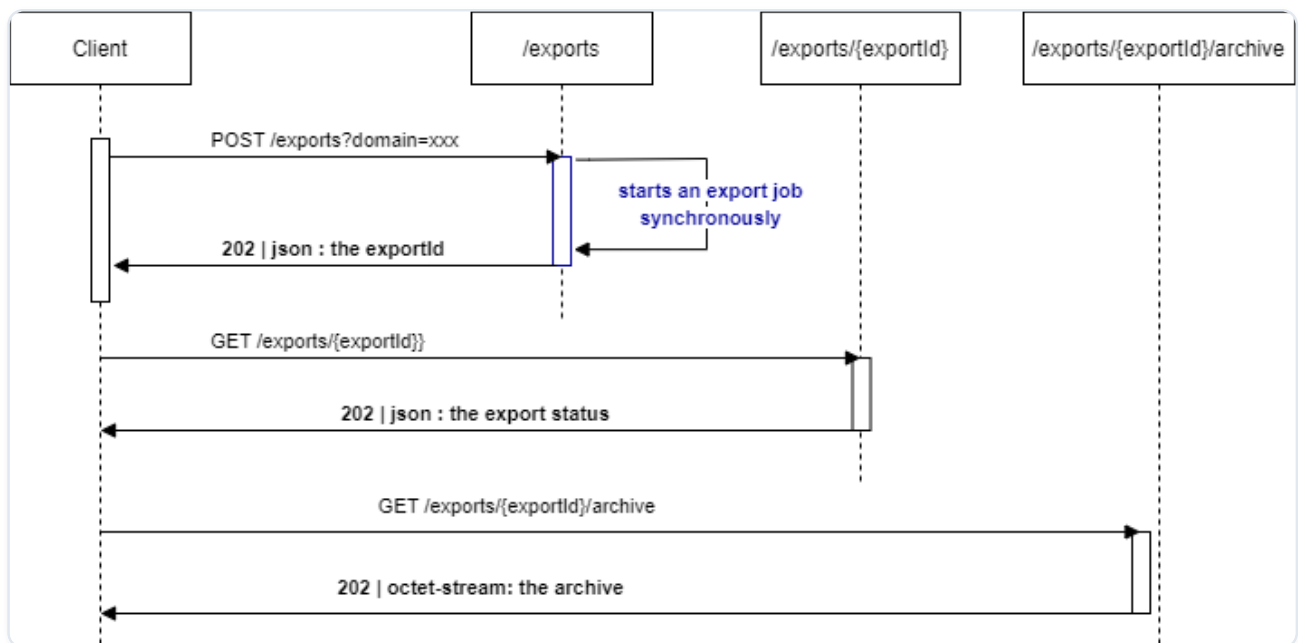
エクスポートサービスは、高度なエクスポート機能処理するように設計されています。アーカイブはエクスポートされ、所定の期間サーバーに保存されます。ユーザーはアーカイブに関する情報を取得でき、有効期限が切れるまでダウンロードできます。提供される機能は次のとおりです。

- エクスポートを開始
- 開始されたエクスポートのステータスを取得
- エクスポートされたアーカイブをダウンロード

このエンドポイントは、同期モードと非同期モードの両方で動作できます。

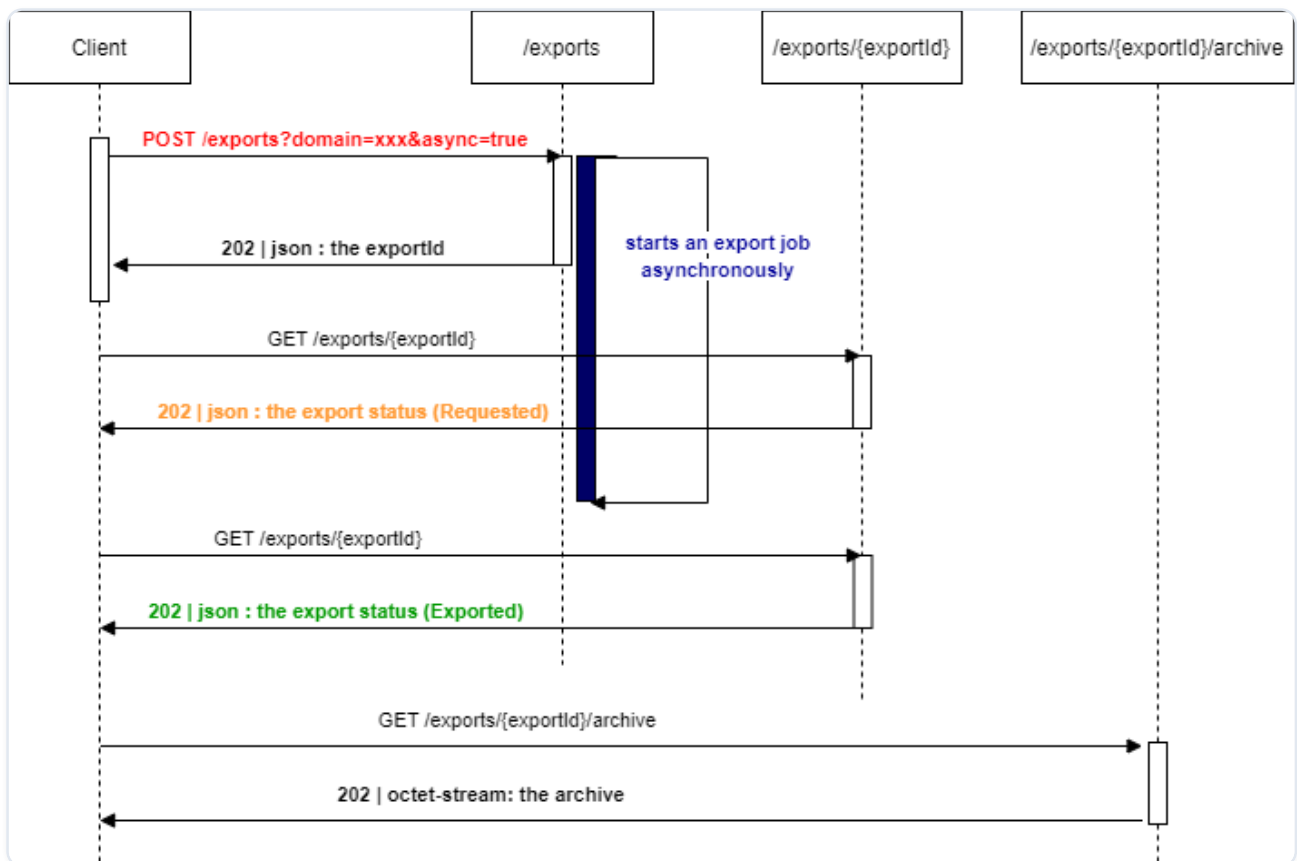
同期モード

アーカイブ全体をサーバーにエクスポートした後に応答するブロッキングモードです。アーカイブはすぐにダウンロード可能になります。



非同期モード

エクスポート ID で応答し、アーカイブをエクスポートするタスクを起動する非ブロッキングモードです。クライアントは、エクスポートのステータスを取得するためにステータスエンドポイントをポーリングする必要があります。ステータスが「Exported」になると、アーカイブはダウンロード可能になります。



整合性のため、ステージングの非同期ジョブは、60 分の作業時間内に完了しない場合、中止されます。

インポート操作

インポートサービスは、高度なインポート機能処理するように設計されています。アーカイブはアップロードされ、所定の期間サーバーに保存されます。ユーザーは、非同期でインポートを開始したり、進行中または完了したインポートに関する情報を取得したりできます。

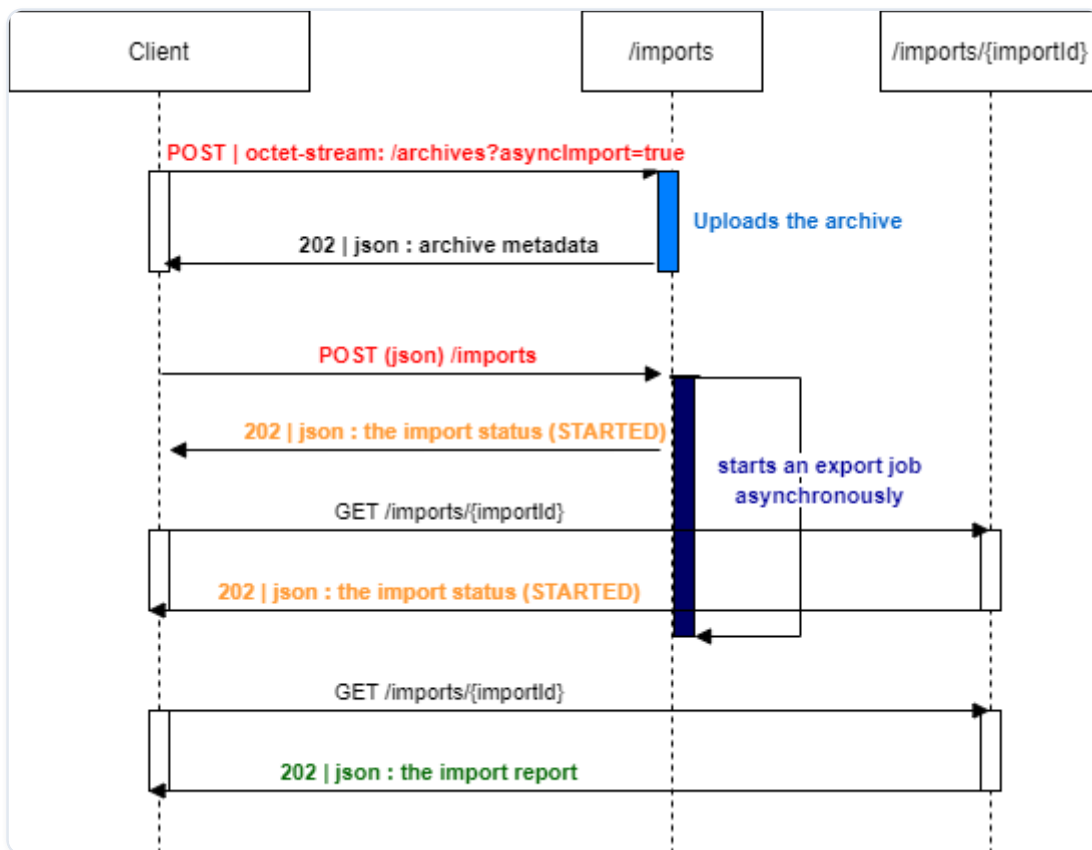
アーカイブは、オプション `asyncImport=true` を指定してアーカイブサービスを使用してアップロードされます。

提供される機能は次のとおりです。

- すでにアップロードされたアーカイブを使用してインポートを開始
- 開始されたインポートのステータス/レポートを取得

アーカイブ名は、異なるオプションで同じアーカイブを再インポートするために、以前のインポートから取得できます。これは有効期限が切れるまで利用可能です。

このエンドポイントは非同期モードでのみ動作します。



データモデル操作

REST API を使用すると、データモデルアシスタント (DMA) で定義された既存のデータモデルを公開できます。利用可能な機能の詳細については、[DMA データモデルの公開](#)を参照してください。

制限事項

一般的な制限事項

- リクエスト URL `{pathInDataset}` または `{pathInRecord}` のインデックスはサポートされていません。
- サブターミナルであるノードに適用されるデータセットノードおよびフィールド操作はサポートされていません。

ターミナルノードの詳細については、[アクセスプロパティ](#)を参照してください。

コンパクト形式の制限事項

- [継承](#)、[追跡情報](#)、[セッションパラメーター](#)、[プロシージャコンテキスト](#)、および[選択ノード](#)としてのフィールドは、[コンパクト形式](#)では処理されません。[拡張形式](#)を使用してください。
- [履歴カテゴリ](#) は、[コンパクト形式](#)ではサポートされていません。

ビジネスオブジェクトカテゴリの制限事項

- ビジネスオブジェクトカテゴリでは、ソート、フィルター、および **履歴** 選択機能は、ビジネスオブジェクトで定義されたリレーションシップの下フィールドには適用できません。

読み取り操作

- **selector** 内では、ページネーションコンテキストは **nextPage** プロパティに限定されます。
- **sortByRelevancy** パラメーターがアクティブ化されている場合、以下のパラメーターは無視されます: **ソート**、**sortOnLabel**、**sortPriority**、および **viewPublication** を通じて定義されたソート条件。
- **viewPublication** パラメーター内では、階層ビューとタイルビューはサポートされていません。
- **sortOnLabel** パラメーターはプログラムによるラベルを無視します。
- システム情報レスポンスのプロパティは、階層表現の REST URL を介して参照できません。

詳細については、[システム情報操作](#)を参照してください。

- 作成準備操作はデータセットノードをサポートしていません。
- **データスペース選択操作**は、ラベルによるソートを行いません。
- **スナップショット選択操作**は、初期スナップショットを含めることができません。

書き込み操作

- 関連付けフィールドは更新できないため、関連付けられたレコードのリストを直接変更することはできません。
- ユーザーインターフェースのコントロールポリシー **onUserSubmit-checkModifiedValues** はサポートされていません。検証エラーを取得するには、**includeValidation** パラメーターを含めてリソースに対する選択操作を呼び出します。

詳細については、[ブロッキング制約と非ブロッキング制約](#)を参照してください。

ディレクトリ操作

- ユーザーのパスワードの変更またはリセットはサポートされていません。

OpenAPI 操作

- ドキュメント生成はユーザーインターフェースを通じて利用できません。

- ドキュメント生成 REST サービスは YAML 形式をサポートしていません。
- データ操作とフォームデータ操作のみをサポートします。ただし、以下は除きます。
 - データセットツリー、データセットノード、およびフィールド操作の選択
 - 単一のリクエストでの複数のレコードの挿入または削除
 - リクエストボディでクエリパラメーターを送信するための HTTP ヘッダー `content-type: x-www-form-urlencoded` の使用
- 「基本認証スキーム」のみが記述されたメソッドです。

TIBCO EBX® ドキュメンテーション - 組み込み RESTful サービス

JSON 形式

- はじめに
- 拡張
- コンパクト形式
- 共通
- その他

はじめに

概要

JSON (JavaScript Object Notation) は、TIBCO EBX® の **RESTful 操作** で使用されるデータ交換形式です。

この形式は軽量で自己記述的であり、UI を設計したり、EBX® を企業の情報システムに統合したりするために使用できます。

次の JSON 形式を使用できます。

- **拡張形式**を使用すると、データコンテキストは網羅的になり、メタモデルデータ、ラベル、検証レポート、継承データ、URL (OpenAPI、詳細、セレクター、履歴詳細など)、履歴の詳細データを取得する機能が含められます (関連付けフィールドと選択ノードを除く)。ただし、これらのフィールドには、デフォルトで含められる詳細と呼ばれる URL リンクを介して応答からアクセスできます
- **コンパクト形式**を使用すると、データコンテキストは、列挙されたフィールドのリンクを除き、メタモデルデータのないマスターデータに限定されます

以下も参照してください。:

- データサービスでのフィールドの非表示

取得されるデータの量は、構成可能なページ付けメカニズムによって制限されます。

URL リンクからは、次の項目に到達できます。

- `details` プロパティを介したテーブル、レコード、データセットの非ターミナルノード、外部キー、リソースフィールド。詳細については、[includeDetails](#) を参照してください
- `selector` パラメーターのアクティベーションによる、外部キーまたは列挙型の可能な値。詳細については、[includeSelector](#) を参照してください

以下も参照してください。:

- アクティベーションと構成

注意:

JSON データは常に UTF-8 文字セットでエンコードされます。

拡張形式

- はじめに
- グローバル構造
 - JSON リクエスト本文
 - JSON 応答本文
- メタモデル
 - テーブルの構造
 - フィールドの構造
 - 外部キーフィールドの構造

- 関連付けフィールドの構造
- リンクされたフィールドの構造
- 列挙依存関係の構造
- 並べ替え基準
- 検証レポート
- 制約
 - 制約タイプ
- コンテンツ
 - ノード値
 - 単純型のコンテンツ
 - グループとリストのコンテンツ
 - セレクター

はじめに

JSON 拡張形式は、マスターデータ、技術情報、およびメタデータを取得するために使用されます。これは、検証、並べ替えなどのいくつかの機能を含めることができるように拡張された方法で設計されています。拡張形式をアクティブにするには、`data` や `form-data` などのサフィックスのない REST カテゴリを URL で使用する必要があります。

グローバル構造

JSON リクエスト本文

リクエストの本文は JSON `オブジェクト` で表され、その内容は操作とカテゴリによって異なります。

データカテゴリ

リクエストの本文には、マスターデータ値を保持する少なくとも `content` プロパティが含まれています。

- データセットノード

指定したノードの下のターミナルノードのターゲット値を指定します。このリクエストは、データセットノードの Update 操作に使用されます。

```
{
  "content": {
    "nodeName1": {
      "content": true
    },
    "nodeName2": {
      "content": 2
    },
    "nodeName3": {
      "content": "Hello"
    }
  }
}
```

以下も参照してください。:

- [Update 操作](#)

• レコード

各フィールドの値を設定して、ターゲットレコードの内容を指定します。欠落しているフィールドの場合、動作はリクエストパラメーター `byDelta` によって異なります。この構造体は、ローカル列挙依存関係フィールド値が宣言されている場合に、テーブルレコードの挿入、レコードの更新、またはセクターの Select 操作に使用されます。

以下も参照してください。:

- 。 継承

`lastUpdateDate` など、一部の技術データを `content` プロパティの横に追加できます。

以下も参照してください。:

- 。 楽観ロック

```
{
  ...
  "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"

            }

        ]

    }

}

}

```

以下も参照してください。:

- [Inset 操作](#)
- [Update 操作](#)

• レコードフィールド

各フィールドの値を設定して、レコードターミナルノードの下のフィールドのターゲット値を指定します。欠落しているフィールドの場合、動作はリクエストパラメーター `byDelta` によって異なります。この構造は、テーブルレコードの更新にのみ使用されます。

以下も参照してください。:

- [継承](#)


```
{
  "content": [
    {
      "content": "CM"
    },
    {
      "content": "PI"
    }
  ]
}
```

以下も参照してください。:

- [Update 操作](#)

• レコードテーブル

各フィールドの値を示すことにより、1 つ以上のレコードの内容を定義します。欠落しているフィールドの場合、動作はリクエストの `byDelta` パラメーターによって異なります。この構造は、テーブルのレコードを挿入または更新するときに使用されます。

```
{
  "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"
        },
        ...
    }
}
]
}

```

以下も参照してください。:

- Inset 操作
- Update 操作

• 削除するレコードテーブル

1 つ以上のレコードを定義します。この構造は、同じテーブルから複数のレコードを削除するときに使用されます。

```

{
    "rows": [
        {
            "details": "http://.../root/table/1"

```

```

    },

    {
        "details": "http://.../root/table/2"
    },

    {
        "primaryKey": "./oid=3"
    },

    {
        "foreignKey": "4"
    },

    ...

]
}

```

以下も参照してください。:

- [Delete 操作](#)

• フィールド

ターゲットフィールドの内容を指定します。このリクエストは、フィールドの更新に使用されます。

リクエストは、`content` エントリのみを保持することにより、`node value` で定義されたものと同じ構造になります。追加のエントリは単に無視されます。

以下も参照してください。:

- Update 操作

- ユーザーインターフェイスを開くまたは閉じる

ユーザーインターフェイスが開いているか閉じているか、および使用不可メッセージを指定します。

```
{
  "content": {
    "toolStatus": {
      "content": true    // or false
    },
    "toolStatusCloseMessage": {
      "content": "Access is temporarily forbidden for maintenance."
    }
  }
}
```

以下も参照してください。:

- ユーザーインターフェイスの操作

リクエストで言及できるのは書き込み可能なフィールドのみです。これには、次の場合は含まれません。

- 関連付けノード
- 選択ノード

- 値関数
- セッターを持たない JavaBean フィールド
- 認証されたユーザーのノードに対する書き込み不可権限

JSON 応答本文

応答の本文は JSON `オブジェクト` で表され、その内容は操作とカテゴリによって異なります。

データカテゴリ

Select 操作には、2 つの異なる部分が含まれます。

`meta` という名前の最初のものには、応答の完全な構造が含まれています。

2 番目の `content`、`rows`、`pagination`... などの再グループ化には、リクエストに対応する値が含まれています。

• データセットツリー

`テーブル` ノードと非ターミナル `グループ` ノードの階層が含まれます。

```
{  
  
  "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"
}
}
}
}
}
}
}
}

```

`meta` および `validation` プロパティはオプションです。

以下も参照してください。:

- [メタモデル](#)
- [検証レポート](#)
- [Select 操作](#)

• データセットノード

指定されたノードの下のターミナルノードのリストが含まれます。

```
{  
  
  "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."

            }

        ]

    }

}

}

```

}

以下も参照してください。:

- [Select 操作](#)

• データセットリスト

指定されたデータスペースまたはデータセットそれぞれのルートデータセットまたは子データセットのリストを含む JSON `オブジェクト`。

すべてのデータセット JSON `オブジェクト` には次のプロパティがあります。

JSON プロパティ	JSON 形式	説明	必須
<code>key</code>	<code>String</code>	データセットのフォーマット設定されたキーを指定します。 形式: <code>[a-zA-Z_]</code> <code>[-a-zA-Z0-9_.*]</code>	はい
<code>label</code>	<code>String</code>	現在のロケールのドキュメントラベル。 これは、値がキーと異なる場合にのみ存在します。	いいえ
<code>isSelectAllowed</code>	<code>Boolean</code>	ユーザーの権限に従ってデータセットを選択できるかどうかを指定します。	はい
<code>details</code>	<code>String</code>	データセットの REST リソース	いいえ

JSON プロパティ	JSON 形式	説明	必須
		URL を指定します。	
<code>information</code>	<code>String</code>	データセットの情報 REST リソース URL を指定します。	いいえ
<code>hasChildren</code>	<code>Boolean</code>	ユーザーの権限に従って、データセットに子があるかどうかを指定します。	はい
<code>children</code>	<code>String</code>	データセットの子の REST リソース URL を指定します。これは、 <code>hasChildren</code> プロパティキーの値が <code>true</code> で、詳細リンクが要求された場合にのみ存在します。	いいえ
<code>childrenHistory</code>	<code>String</code>	履歴データセットの子の REST リソース URL を指定します。これは、 <code>hasChildren</code> プロパティキーの値が <code>true</code> で、詳細リンクが要求された場合にのみ存在します。	いいえ

JSON プロパティ	JSON 形式	説明	必須
		以下も参照してください。: <code>includeHistory</code>	

```
{
  "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"

  }

}

```

以下も参照してください。:

- データセットの選択

• データセット情報

次のプロパティを含む JSON オブジェクト

JSON プロパティ	JSON 形式	説明	必須
content	Object	データセット情報に対応します。	はい
key	String	データセットのフォーマット設定されたキー値を指定します。形式: [a-zA-Z_][-a-zA-Z0-9_]*、最大 64 文字に制限されます。	はい
documentation	JSON オブジェクトの Array	ロケールごとに JSON オブジェクトを使用してローカライズされたドキュメントに対応します。	いいえ

JSON プロパティ	JSON 形式	説明	必須
<code>locale</code> (<code>documentation</code> プロパティの下にネスト)	<code>String</code>	ドキュメントのロケール。	はい
<code>label</code> (<code>documentation</code> プロパティの下にネスト)	<code>String</code>	ドキュメントラベル。	いいえ
<code>description</code> (<code>documentation</code> プロパティの下にネスト)	<code>String</code>	ドキュメントの説明。	いいえ
<code>dataModel</code>	JSON オブジェクト	データセットスキーマの場所を指定します。モジュールスキーマにパッケージ化されたものとパブリケーションのみがサポートされます。	はい
<code>module</code> (<code>dataModel</code> プロパティの下にネスト)	<code>String</code>	データセットスキーマが配置されているモジュール名を指定します。	いいえ
<code>pathInModule</code> (<code>dataModel</code> プロパティの下にネスト)	<code>String</code>	データセットスキーマが配置されているモジュール内のパスを指定します。	いいえ
<code>publication</code> (<code>dataModel</code> プロパティの下にネスト)	<code>String</code>	データセットスキーマのパブリケーション名を指定します。	いいえ
<code>dataModel</code> (<code>dataModel</code> プロ	<code>String</code>	データセットスキーマパブリケーション	いいえ

JSON プロパティ	JSON 形式	説明	必須
パティの下にネスト)		のデータモデル名を指定します。	
<code>versionName</code> (<code>dataModel</code> プロパティの下にネスト)	<code>String</code>	データセットスキーマパブリケーションのバージョン名を指定します。	いいえ
<code>parent</code>	<code>String</code>	親データセットのフォーマット設定されたキー値を指定します。親がない場合は、 <code>null</code> 値が保持されます。	はい
<code>activated</code>	<code>Boolean</code>	データセットがアクティブ化されているかどうかを指定します。	はい
<code>owner</code>	<code>String</code>	所有者プロファイルを指定します。	はい
<code>creator</code>	<code>String</code>	作成者プロファイルを指定します。	はい
<code>creation_time</code>	<code>Date</code>	作成時刻を指定します。	はい
<code>updater</code>	<code>String</code>	アップデータープロファイルを指定します。	はい
<code>update_time</code>	<code>Date</code>	更新時刻を指定します。	はい

```
{
  "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": "Beverlyone",

"creator": "Uadmin",

"creation_time": "2023-10-10T15:49:03.023",

"updater": "Uadmin",

```

```

      "update_time": "2023-10-10T15:49:03.005"

    }

  }

```

以下も参照してください。:

- データセットの選択

• テーブル

次のプロパティを含む JSON オブジェクト

- (オプション) テーブルメタモデル
- (オプション) 適用される並べ替え基準
- (オプション) テーブル検証レポート
- 選択したレコードの JSON 配列 に対応する `rows` プロパティ。各レコードは、JSON オブジェクト で表されます。レコードが選択されていない場合、JSON 配列 は空です。
- (オプション) `pagination` データを含む `pagination` プロパティ。

```

{

  "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"

}

}

```

以下も参照してください。:

- Select 操作

• レコード

以下を含む JSON オブジェクト

- ラベル
- (オプション) レコード URL
- (オプション) 技術データ
- (オプション) テーブルメタモデル
- (オプション) レコード検証レポート
- (オプション) レコードの継承モード。 `root`、`inherit`、`overwrite`、または `occult` のいずれかになります。

以下も参照してください。:

- レコードルックアップメカニズム
- 継承

- レコードの内容

```
{  
  
  "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": {  
  
  ...  
  
}
```

```
}
```

以下も参照してください。:

- [Select 操作](#)
- [準備操作](#)

• フィールド

関連付けノードまたは選択ノードの場合、`includeDetails` パラメーターが `true` に設定されている場合に限り、関連付けられたレコードを含むターゲットテーブルが含まれます。

他の種類のノードの場合、現在のノード値が含まれます。

以下も参照してください。:

- [Select 操作](#)

• ユーザーインターフェイスの状態の取得

ユーザーインターフェイスのステータスと使用不可メッセージが含まれます。

```
{
  "content": {
    "toolStatus": {
      "content": true,
      "label": "Open",
      "selector": "http://.../domain/toolStatus/toolStatus?
selector=true"
    },
```

```

    "toolStatusCloseMessage": {

        "content": "Access is temporarily forbidden for maintenance."

    }

}

}

```

以下も参照してください。:

- 。ユーザーインターフェイスの操作

注意:

ノード、レコードとフィールド、プロパティと値は、解決された権限 ([権限](#) を参照) に応じて非表示になる場合があります。

メタモデル

このセクションは、`includeMetamodel` パラメーターを使用してオンデマンドでアクティブ化できます。`content` セクションの構造と JSON タイプについて説明します。

このセクションは、Select 操作ではデフォルトで無効になっています。

以下も参照してください。:

- [Select 操作](#)
- [準備操作](#)

テーブルの構造

テーブルメタモデルは、次のプロパティを持つ JSON `オブジェクト` で表されます。

JSON プロパティ	JSON 形式	説明	必須
<code>name</code>	<code>String</code>	モデルで定義されている許可されたテーブルの名前を指定します。	はい
<code>label</code>	<code>String</code>	テーブルのラベルを指定します。未定義の場合、スキーマノードの名前が返されます。	はい
<code>説明</code>	<code>String</code>	テーブルの説明を指定します。	いいえ
<code>type</code>	<code>String</code>	ノードのタイプを指定します。値は常に <code>table</code> と等しくなります。	はい
<code>minOccurs</code>	<code>Number</code>	許可された最小レコード数を指定します。	はい
<code>maxOccurs</code>	<code>Number</code> または <code>String</code>	許可された最大レコード数または <code>unbounded</code> を指定します。	はい
<code>history</code>	<code>Boolean</code>	テーブルの内容が履歴化されるかどうかを指定します。履歴がアクティブ化されている場合、その値は <code>true</code> であり、	いいえ

JSON プロパティ	JSON 形式	説明	必須
		そうでない場合は <code>false</code> です。 以下も参照してください。: • History	
<code>primaryKeyFields</code>	<code>Array</code>	パスの配列として表される主キーの構成を指定します。	はい
<code>inheritance</code>	<code>Boolean</code>	テーブルのデータセット継承をアクティブにするかどうかを指定します。その値は、継承がアクティブ化されている場合は <code>true</code> であり、それ以外の場合は <code>false</code> です。 以下も参照してください。: • 継承と値の解決	いいえ
<code>fields</code>	<code>Array</code>	フィールドの配列として表されるレコードノードの直接の子を指定します。各フィールドには、サブフィールドが再帰的に含まれる場合があります。	はい

フィールドの構造

許可された各フィールドは、次のプロパティを持つ JSON オブジェクト で表されます。

JSON プロパティ	JSON 形式	説明	必須
<code>name</code>	<code>String</code>	モデルで定義されている許可フィールドの名前を指定します。	はい
<code>label</code>	<code>String</code>	ノードのラベルを指定します。未定義の場合、スキーマノードの名前が返されます。	はい
<code>説明</code>	<code>String</code>	フィールドの説明を指定します。	いいえ
<code>type</code>	<code>String</code>	フィールドのタイプを指定します。これには、 <code>単純型</code> 、 <code>group</code> 、 <code>table</code> 、 <code>foreignKey</code> 、 <code>association</code> などがあります。	はい
<code>minOccurs</code>	<code>Number</code>	許可された最小オカレンスの数を指定します。	はい
<code>maxOccurs</code>	<code>Number</code> または <code>String</code>	許可された最大オカレンスまたは <code>unbounded</code> の数を指定します。	はい
<code>readOnly</code>	<code>Boolean</code>	フィールドアクセス許可がデータモデルアクセスプロパティ	はい

JSON プロパティ	JSON 形式	説明	必須
		から読み取り専用として定義されているかどうかを指定します。つまり、そのデータの読み取りまたは書き込みが可能かどうかです。 フィールドが読み取り専用の場合、その値は <code>true</code> であり、それ以外の場合は <code>false</code> です。 以下も参照してください。: • アクセスプロパティ • <code>AccessPermission</code>	
<code>autoIncrement</code>	<code>Boolean</code>	フィールドを自動インクリメントするかどうかを指定します。このプロパティは、整数フィールドタイプでのみ使用できます。フィールドが自動インクリメントされる場合、その値は <code>true</code> であり、それ以外の場合は <code>false</code> です。 以下も参照してください。:	いいえ

JSON プロパティ	JSON 形式	説明	必須
		自動インクリメントされた値	
<code>checkNullInput</code>	<code>Boolean</code>	ユーザー入力時に <code>null</code> フィールドの制約のチェックをアクティブにするかどうかを指定します。その値は、アクティブ化されている場合は <code>true</code>、それ以外の場合は <code>false</code> です。 制約チェックは、フィールドのプロパティ <code>osd:checkNullInput</code> が <code>true</code> に設定されている場合にアクティブになります。	はい
<code>inheritedField</code>	<code>Object</code>	継承されたフィールドの値ソースに関連する情報を保持します。 <pre>{ "inheritedField": { "sourceRecord": "/path/to/record", // (optional) "sourceNode": "./path/to/Node" } }</pre>	いいえ

JSON プロパティ	JSON 形式	説明	必須
		} 以下も参照してください。: 継承と値の解決	
foreignKey	Object	ターゲットテーブルに関連する情報を保持します。 以下も参照してください。: 外部キーフィールドの構造	いいえ (*)
association	Object	関連付けテーブルに関連する情報を保持します。 以下も参照してください。: 関連付けフィールドの構造	いいえ (*)
enumeration	String	フィールドが列挙値であるかどうかを指定します。可能な値は次のとおりです。 - dynamic - foreignKey - nomenclature	いいえ

JSON プロパティ	JSON 形式	説明	必須
		• <code>programmatic</code> • <code>resource</code> • <code>static</code> 以下も参照してください。: • <code>SchemeFacetEnumeration</code> その値に従って、<code>selector</code> リクエストパラメーターを使用してフィールドを使用して可能な値を取得できるかどうかを示します。	
<code>enumerationDependencies</code>	JSON オブジェクトの <code>Array</code>	メタデータ列挙の依存関係リストを指定します。 <code>externalKey</code>、<code>dynamic</code>、または <code>programmatic</code> の列挙型でのみ使用できます。 以下も参照してください。: • 列挙依存関係の構造	いいえ
<code>valueFunction</code>	<code>Boolean</code>	フィールドが計算値であるかどうかを指定します。	いいえ

JSON プロパティ	JSON 形式	説明	必須
		以下も参照してください。: 計算値	
<code>linkedField</code>	<code>Object</code>	リンクされたフィールドに関連する情報を保持します。 以下も参照してください。: <code>SchemaNode.getSchemaLinkedField()</code> - リンクされたフィールドの構造	いいえ
<code>pathInDataset</code>	<code>String</code>	スキーマノードから始まる相対フィールドのパスを指定します。	いいえ (**)
<code>pathInRecord</code>	<code>String</code>	テーブルノードから始まる相対フィールドのパスを指定します。	いいえ (*)
<code>filterable</code>	<code>Boolean</code>	<code>filter</code> リクエストパラメーターを使用してレコードをフィルタリングするためにフィールドを使用できるかどうかを指定します。	いいえ (*)

JSON プロパティ	JSON 形式	説明	必須
<code>hiddenFilterPolicy</code>	<code>String</code>	非表示のフィルターポリシーを指定します。 可能な値は、<code>textSearchOnly</code>、<code>true</code>、または <code>false</code> です。 以下も参照してください。: - 構造化検索ツールでのフィールドの非表示 <code>SchemaNodeDefaultView.getHiddenInSearch</code>	いいえ (*)
<code>sortable</code>	<code>Boolean</code>	<code>sort</code> リクエストパラメーターを使用して、フィールドを並べ替え基準で指定します。	いいえ (*)
<code>constraints</code>	JSON オブジェクトの <code>Array</code>	フィールドの非プログラム制約を指定します。 以下も参照してください。: 制約	いいえ
<code>fields</code>	JSON オブジェクトの <code>Array</code>	各フィールドグループの構造と入力を保持します。	いいえ

JSON プロパティ	JSON 形式	説明	必須
isBusinessObjectRelat	Boolean	フィールド構造が現在のテーブル内にあるのか、ビジネスオブジェクトを通じて与えられるのかを指定します。	いいえ

(*) テーブル、レコード、およびレコードフィールド操作でのみ使用できます。

(**) データセットツリー操作でのみ使用できます。

外部キーフィールドの構造

外部キーフィールドのメタモデルは、JSON **オブジェクト** で表されます。

```
{
  "dataspace": "BAuthors",
  "dataset": "Authors",
  "tablePath": "/root/Authors",
  "details": "http://.../BAuthors/Authors/root/Authors"
}
```

次のプロパティを保持します。

JSON プロパティ	JSON 形式	説明	必須
dataspace	String	ターゲットデータスペースまたはスナップショット識別子を指定します。指定しない場合は、テーブルのデータスペース	いいえ

JSON プロパティ	JSON 形式	説明	必須
		と同じデータスペースに対応します。	
<code>dataset</code>	<code>String</code>	ターゲットデータセットの識別子を指定します。指定しない場合は、テーブルのデータセットと同じデータセットに対応します。	いいえ
<code>tablePath</code>	<code>String</code>	ターゲットテーブルのパスを指定します。	はい
<code>details</code>	<code>String</code>	ターゲットテーブルの REST リソース URL を指定します。 以下も参照してください。: <code>includeDetails</code> パラメーター	いいえ
<code>historyDetails</code>	<code>String</code>	履歴ターゲットテーブルの REST リソース URL を指定します。 以下も参照してください。: <code>includeDetails</code> パラメーター	いいえ

関連付けフィールドの構造

関連フィールドのメタモデルは、JSON オブジェクト で表されます。このオブジェクトは、関連付けタイプに依存するプロパティを保持します。

- `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"
}
```

次のプロパティを保持します。

JSON プロパティ	JSON 形式	説明	必須
<code>type</code>	<code>String</code>	関連付けタイプを指定します。可能な値は、<code>tableRefInverse</code>、<code>linkTable</code>、<code>xpathLink</code> です。 以下も参照してください。: • 関連付け	はい
<code>dataspace</code>	<code>String</code>	ターゲットデータスペースまたはスナップショット識別子を指定します。指定しない場合は、テーブル	いいえ

JSON プロパティ	JSON 形式	説明	必須
		ルのデータスペース と同じデータスペース に対応します。	
<code>dataset</code>	<code>String</code>	ターゲットデータ セットの識別子を指 定します。指定しな い場合は、テーブル のデータセットと同 じデータセットに対 応します。	いいえ
<code>tablePath</code>	<code>String</code>	ターゲットテーブル のパスを指定しま す。	はい
<code>details</code>	<code>String</code>	ターゲットテーブル の REST リソース URL を指定しま す。 以下も参照してくだ さい。: <code>includeDetails</code> パラメーター	いいえ
<code>fieldToSource</code>	<code>String</code>	関連付けのソース テーブルを参照する フィールドを指定し ます。 関連付けタイプが <code>tableRefInverse</code> または <code>linkTable</code> のど	いいえ

JSON プロパティ	JSON 形式	説明	必須
		ちらであるかを定義します。	
<code>fieldToTarget</code>	<code>String</code>	ターゲットテーブルへのフィールドのパスを指定します。 関連付けタイプが <code>linkTable</code> の場合に定義されます。	いいえ
<code>linkTablePath</code>	<code>String</code>	リンクテーブルのパスを指定します。 関連付けタイプが <code>linkTable</code> の場合に定義されます。	いいえ
<code>predicate</code>	<code>String</code>	現在のノードを基準にして、関連付けの基準を指定します。 関連付けタイプが <code>xpathLink</code> の場合に定義されます。	いいえ
<code>filter</code>	<code>String</code>	関連するオブジェクトをフィルタリングするための XPath 述語式を指定します。	いいえ

リンクされたフィールドの構造

リンクされたフィールドのメタモデルは、JSON `オブジェクト` で表されます。

```
{
  "relationshipField": "/au_id",
  "tablePath": "/root/Authors",
  "linkedFieldPath": "/birth_date",
  "details": "http://.../BAuthors/Authors/root/Authors"
}
```

次のプロパティを保持します。

JSON プロパティ	JSON 形式	説明	必須
<code>relationshipField</code>	<code>String</code>	このリンクされたフィールドによって使用される関係を保持するフィールドのパスを指定します。 以下も参照してください。: <code>SchemaLinkedField.getRelationshipNode</code>	はい
<code>tablePath</code>	<code>String</code>	このリンクされたフィールドによって参照されるテーブルのパスを指定します。	はい
<code>linkedFieldPath</code>	<code>String</code>	このリンクされたフィールドによって参照されるパスを指定します。	はい

JSON プロパティ	JSON 形式	説明	必須
		以下も参照してください。: <code>SchemaLinkedField.getLinkedField</code>	
<code>details</code>	<code>URI</code>	ターゲットテーブルの REST リソース URL を指定します。 以下も参照してください。: <code>includeDetails</code> パラメーター	いいえ

以下も参照してください。:

- リンクされたフィールド
- `SchemaLinkedField`

列挙依存関係の構造

メタモデル列挙依存オブジェクトは、JSON `オブジェクト` で表されます。

```
{
  "localModify": false,
  "dataspace": "BAuthors",
  "dataset": "Authors",
  "targetPath": "/root/Authors",
  "details": "http://.../BAuthors/Authors/root/Authors"
```

}

次のプロパティを保持します。

JSON プロパティ	JSON 形式	説明	必須
<code>localModify</code>	<code>Boolean</code>	列挙型依存関係フィールドが、同じレコード内のフィールドに対して実行された変更の影響を受けるかどうかを指定します。	はい
<code>dataspace</code>	<code>String</code>	列挙型依存関係のターゲットデータスペースまたはスナップショット識別子を指定します。指定しない場合は、レコードのデータスペースと同じデータスペースに対応します。	いいえ
<code>dataset</code>	<code>String</code>	列挙型依存関係のターゲットデータセットの識別子を指定します。指定しない場合は、レコードのデータセットと同じデータセットに対応します。	いいえ
<code>targetPath</code>	<code>String</code>	列挙型依存関係のターゲットパスを指定します。	はい

JSON プロパティ	JSON 形式	説明	必須
<code>details</code>	<code>String</code>	列挙型依存関係のターゲット REST リソーステーブルの URL を指定します。 以下も参照してください。: <code>includeDetails</code> パラメーター	いいえ

並べ替え基準

リクエストに適用される並べ替え基準は、`includeSortCriteria` パラメーター (デフォルトでは非アクティブ) を使用して、オンデマンドで返すことができます。有効にすると、`sortCriteria` プロパティが応答ルートノードに直接追加されます。

`sortCriteria` プロパティは、順序付けられた並べ替え基準を含む JSON 配列として表されます。各並べ替え基準は、次のプロパティを持つ JSON オブジェクトに対応しています。

JSON プロパティ	JSON 形式	説明	必須
<code>path</code>	<code>String</code>	フィールドのパス。	はい
<code>order</code>	<code>String</code>	可能な値は、<code>asc</code>、<code>lasc</code>、<code>desc</code>、または <code>ldesc</code> です。	はい

`sortByRelevancy` プロパティは JSON Array で表され、それ自体が次のプロパティを持つ JSON オブジェクトを保持します。

JSON プロパティ	JSON 形式	説明	必須
<code>order</code>	<code>String</code>	並べ替えを昇順で行うか降順で行うかを定義します。可能な	はい

JSON プロパティ	JSON 形式	説明	必須
		値は、 <code>lasc</code> または <code>ldesc</code> です。詳細については、 sortByRelevancy を参照してください。	

検証レポート

検証は、`includeValidation` パラメーターを使用してオンデマンドでアクティブ化できます (デフォルトでは非アクティブ化されています)。有効にすると、`validation` プロパティが 1 つまたは複数のメッセージとともにターゲットノードに直接追加されます。ターゲットノードのパスがないメッセージの場合、ルートノードに `validation` プロパティが追加されます。

`validation` プロパティは JSON `Array` で表され、メッセージごとに JSON `オブジェクト` を保持し、検証項目に対応し、次のプロパティを持ちます。

JSON プロパティ	JSON 形式	説明	必須
<code>level</code>	<code>String</code>	検証項目の重大度。 可能な値は、 <code>info</code> 、 <code>warn</code> 、 <code>error</code> 、 <code>fatal</code> です。	はい
<code>message</code>	<code>String</code>	検証項目の説明。	はい
<code>details</code>	<code>String</code> 絶対 URL に対応します。	検証項目に関連付けられているリソースの URL。 関連するリソースが存在し、それが含まれている場合、テーブルスコープとデータセットスコープでのみ使用できます。	いいえ

JSON プロパティ	JSON 形式	説明	必須
		以下も参照してください。: <code>includeDetails</code> パラメーター	

制約

このセクションは、ノードに読み取り/書き込み権限がある場合に自動的にアクティブになります。これは、データモデルで宣言された非プログラムの制約を提供します。

`constraints` プロパティは、JSON 配列で表されます。すべての制約は、JSON オブジェクトの形式を取ります。それらには次のプロパティがあります。

JSON プロパティ	JSON 形式	説明	必須
<code>type</code>	<code>String</code>	制約のタイプを指定します。 以下も参照してください。: 制約タイプ	はい
<code>level</code>	<code>String</code>	制約の重大度レベルを指定します。可能な値は、<code>fatal</code>、<code>error</code>、<code>warning</code>、または <code>info</code> です。	はい
<code>content</code>	<code>Object</code>	非プログラム制約の値を指定します。この値は、<code>String</code> や JSON オブジェクトなどの単純型	いいえ

JSON プロパティ	JSON 形式	説明	必須
		にすることができます。	
<code>min</code>	<code>Integer</code>	<code>excludeSegment</code> 制約タイプの最小値を指定します。	いいえ
<code>max</code>	<code>Integer</code>	<code>excludeSegment</code> 制約タイプの最大値を指定します。	いいえ
<code>message</code>	<code>String</code>	フィールドの値が制約に準拠していない場合に返される検証メッセージを指定します。	はい
<code>blocksCommit</code>	<code>String</code>	ブロッキングまたは非ブロッキング制約の制御ポリシー管理を指定します。 詳細については、 ブロッキング制約と非ブロッキング制約 を参照してください。	いいえ

制約タイプ

このセクションでは、プログラム以外の制約タイプをリストします。

タイプ	説明
<code>mandatory</code>	フィールドが必須であることを定義します。
<code>fixedLengthStatic</code>	このフィールドに必要な正確な文字数を定義します。

タイプ	説明
<code>fixedLengthDynamic</code>	これに必要な正確な文字数を提供するフィールドを指摘します。
<code>minLengthStatic</code>	このフィールドに許可される最小文字数を定義します。
<code>minLengthDynamic</code>	これに許可される最小文字数を提供するフィールドを指摘します。
<code>maxLengthStatic</code>	このフィールドに許可される最大文字数を定義します。
<code>maxLengthDynamic</code>	このフィールドに許可される最大文字数を提供するフィールドを示します。
<code>excledeSegment</code>	このフィールドに許可されていない値の包括的範囲を定義します。
<code>excludeValue</code>	このフィールドに許可されていない値のリストを定義します。
<code>fractionDigits</code>	このフィールドに許可される小数点以下の最大数を定義します。
<code>pattern</code>	フィールドの値と一致する必要がある正規表現パターンを定義します。
<code>totalDigits</code>	この整数または 10 進数フィールドに許可される最大桁数を定義します。
<code>boundaryMinInclusiveStatic</code>	このフィールドに許可される最小値 (最小値を含む) を定義します。
<code>boundaryMinExclusiveStatic</code>	このフィールドに許可される最小値 (最小値を除く) を定義します。
<code>boundaryMinInclusiveDynamic</code>	このフィールドに許可される最小値 (最小値を含む) を提供するフィールドを定義します。
<code>boundaryMinExclusiveDynamic</code>	このフィールドに許可される最小値 (最小値を除く) を提供するフィールドを定義します。

タイプ	説明
<code>boundaryMaxInclusiveStatic</code>	このフィールドに許可される最大値 (最大値を含む) を定義します。
<code>boundaryMaxExclusiveStatic</code>	このフィールドに許可される最大値 (最大値を除く) を定義します。
<code>boundaryMaxInclusiveDynamic</code>	このフィールドに許可される最大値 (最大値を含む) を提供するフィールドを定義します。
<code>boundaryMaxExclusiveDynamic</code>	このフィールドに許可される最大値 (最大値を除く) を提供するフィールドを定義します。

以下も参照してください。:

- XML スキーマでサポートされているファセット
- 拡張ファセット

コンテンツ

このセクションは、`includeContent` パラメーター (デフォルトでアクティブ) を使用して、オンデマンドで非アクティブ化できます。これは、許可されたユーザーのレコード値、データセット、またはコンテンツフィールドの 1 つのフィールドのコンテンツを提供します。また、ラベル、技術情報、URL などの追加情報もあります。

コンテンツは、サブノードごとにプロパティが設定されたJSON `オブジェクト` で表されます。

ノード値

JSON プロパティ	JSON 形式	説明	必須
<code>content</code>	単純型のコンテンツ グループとリストのコンテンツ	ノード値が含まれます。 <code>association</code> と <code>selection</code> を除くすべてのノードで使用できます。ただし、コンテンツは、	いいえ

JSON プロパティ	JSON 形式	説明	必須
		<code>details</code> プロパティで提供される URL を呼び出すことで取得できます。	
<code>details</code>	<code>String</code> 絶対 REST リソース URL に対応します。	呼び出されたときにノードの詳細を返します。 応答タイプはメタタイプによって異なります。 • <code>foreignKey</code> : ターゲットレコード (テーブル、レコード、およびフィールド操作で使用可能)。 • <code>resource</code> : ターゲットリソース (データセットノード、テーブル、レコード、およびフィールド操作で使用可能)。 • <code>association</code> : 関連するレコードを含むターゲットテーブル (テーブルおよびレコード	<code>いいえ</code>

JSON プロパティ	JSON 形式	説明	必須
		操作で使用可能)。 • <code>selection</code> : 関連するレコードを含むターゲットテーブル (テーブルおよびレコード操作で使用可能)。 • <code>group</code> : ターゲットデータセットグループノード (データセットツリー操作で使用可能)。 • <code>table</code> : ターゲットテーブル (データセットツリー操作で使用可能)。 例 <pre>http://.../ BReference/ dataset/root/ table/pk/ associationField</pre>	
<code>historyDetails</code>	<code>String</code> 絶対 REST リソース URL に対応します。	呼び出されたときにノード履歴の詳細を返します。 以下も参照してください。:	<code>いいえ</code>

JSON プロパティ	JSON 形式	説明	必須
		<code>includeHistory</code>	
<code>label</code>	<code>String</code>	現在のロケールの外部キーまたは列挙ラベルが含まれます。 現在のロケールがサポートされていない場合は、デフォルトのラベルが返されます。	いいえ
<code>inheritanceMode</code>	<code>String</code>	データセットの継承のみを考慮した、ノードの継承状態が含まれます。 <code>inheritedFieldMode</code> プロパティと <code>inheritanceMode</code> プロパティの両方を同じノードで定義することはできません。 以下も参照してください。: <code>inheritedFieldMode</code> - レコードルックアップメカニズム	いいえ
<code>inheritedFieldMode</code>	<code>String</code>	データセットとフィールドの継承を考慮した、ノードのフィールドの継承状	いいえ

JSON プロパティ	JSON 形式	説明	必須
		態が含まれます。両方の継承が使用される場合、フィールドの継承はデータセットの継承よりも優先されます。 <code>inheritedFieldMode</code> プロパティと <code>inheritanceMode</code> プロパティの両方を同じノードで定義することはできません。 以下も参照してください。: <code>inheritanceMode</code> - 値ルックアップメカニズム	
<code>selector</code>	<code>String</code> 絶対 URL に対応します。	<code>selector</code> 操作に適切な URL が含まれています。 例 <code>http://.../BReference/dataset/root/table/pk/enumField?selector=true</code>	いいえ
<code>validation</code>	<code>Array</code>	現在のノードコンテンツに関連する検	いいえ

JSON プロパティ	JSON 形式	説明	必須
		証レポートが含まれています。 以下も参照してください。: 検証レポート	

単純型のコンテンツ

単純なフィールド値は JSON `オブジェクト` に格納され、そのコンテンツは `content` プロパティの値です。

以下も参照してください。:

- 単純型形式

グループとリストのコンテンツ

XML スキーマ	JSON 形式	例	メタタイプ
グループ <code>xs:complexType</code>	<code>Object</code> サブノードごとのプロパティが含まれます。	単純オカレンスグループの例 <pre>{ "road": { "content": "11 rue scribe" }, "zipcode": { "content": "75009" }, }</pre>	<code>group</code>

XML スキーマ	JSON 形式	例	メタタイプ
		<pre>"country": { "content": "France" }</pre>	
List <code>maxOccurs > 1</code>	Array JSON オブジェクトで表されるすべてのフィールドオカレンスの配列が含まれます。 各オブジェクトは、ノード値として表されます。	<code>xs:int</code> 型の複数オカレンスフィールドの例 <pre>[{ "content": 0 }, { "content": 1 }, { "content": 2 }, { "content": 3 }]</pre>	単純型のメタ または group

XML スキーマ	JSON 形式	例	メタタイプ
		複数オカレンスグループの例 <pre>[{ "content": { "road": { "content": "11 rue scribe" }, "zipcode": { "content": "75009" }, "country": { "content": "France" } } }, { "content": { "road": { "content": "711 Atlantic Ave" }, "zipcode": {</pre>	

XML スキーマ	JSON 形式	例	メタタイプ
		<pre> "content": "MA 02111" }, "country": { "content": "United States" } } }] </pre>	

セレクトー

列挙型を提供するフィールドで `selector` プロパティで表される URL を呼び出すと、次のプロパティを含む JSON オブジェクト が返されます。

- JSON オブジェクト の `Array` に対応する `rows`。各行には返される `content` (永続化可能) と対応する `label` の 2 つのエントリが含まれています。確立された可能な項目のリストは、現在のコンテキストによって異なります。
- (オプション) `pagination` データを含む `pagination` (デフォルトでアクティブ)。

```

{

  "rows": [

    {

      "content": "F",

      "label": "feminine"

    },

    {

```

```
        "content": "M",  
  
        "label": "masculine"  
    }  
  
    ],  
  
    "pagination": {  
  
        "firstPage": null,  
  
        "nextPage": null,  
  
        "previousPage": null,  
  
        "lastPage": null  
    }  
  
}
```

以下も参照してください。:

- [includeSelector](#)

コンパクト

- [はじめに](#)
- [グローバル構造](#)
 - [JSON リクエスト本文](#)
 - [JSON 応答本文](#)
- [コンテンツ](#)
 - [ノード値](#)
 - [単純型のコンテンツ](#)

- 。グループ、リスト、列挙の内容

はじめに

JSON コンパクト形式の目的は、軽量構造を使用してマスターデータを取得することです。キーバリュー設計に従って、技術情報なしで最も簡単な方法でデータを表示します。コンパクト形式をアクティブにするには、`data-compact` や `form-data-compact` などの `compact` サフィックス REST カテゴリを URL で使用する必要があります。

グローバル構造

JSON リクエスト本文

リクエストの本文は JSON `オブジェクト` で表され、その内容は操作とカテゴリによって異なります。

データカテゴリ

- データセットノード

指定したノードの下のターミナルノードのターゲット値を指定します。このリクエストは、データセットノードの Update 操作に使用されます。

```
{  
  
  "nodeName1": true,  
  
  "nodeName2": 2,  
  
  "nodeName3": "Hello"  
  
}
```

HTTP リクエストと HTTP 応答の間で JSON 対称構造を確保するために、列挙ノードリクエストの形式は次のようになります。

```
{  
  
  "enumerationNode": {  
  
    "key": "a key value"  
  
  }  
  
}
```

```
}
```

以下も参照してください。:

- [Update 操作](#)

• レコード

各フィールドの値を設定して、ターゲットレコードの内容を指定します。欠落しているフィールドの場合、動作はリクエストパラメーター `byDelta` によって異なります。この構造体は、ローカル列挙依存関係フィールド値が宣言されている場合に、テーブルレコードの挿入、レコードの更新、またはセレクトの Select 操作に使用されます。

```
{
  "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"

]

}

```

以下も参照してください。:

- [Insert 操作](#)
- [Update 操作](#)

• レコードフィールド

各フィールドの値を設定して、レコードターミナルノードの下のフィールドのターゲット値を指定します。欠落しているフィールドの場合、動作はリクエストパラメーター `byDelta` によって異なります。この構造は、テーブルレコードの更新にのみ使用されます。

```

{

    "jobs": [

        {

            "key": "CM"

        },

    ],

}

```

```

    {
      "key": "PI"
    }
  ]
}

```

以下も参照してください。:

- [Update 操作](#)

• レコードテーブル

1 つ以上のレコードの内容を、各フィールドの値を指定して定義します。欠落しているフィールドの場合、動作はリクエスト `byDelta` のパラメーターによって異なります。この構造は、テーブルのレコードを挿入または更新するときに使用されます。

```

[
  {
    "gender": {
      "key": "M"
    },
    "lastName": "Saint-Saëns",
    "firstName": "Camille",
    "birthDate": "1835-10-09",
    ...
  },

```

```
{
  "gender": {
    "key": "M"
  },
  "lastName": "Debussy",
  "firstName": "Claude",
  "birthDate": "1862-10-22",
  ...
}
```

]

以下も参照してください。:

- [Insert 操作](#)
- [Update 操作](#)

• 削除するレコードテーブル

1 つ以上のレコードを定義します。この構造は、同じテーブルから複数のレコードを削除するときに使用されます。

```
[
  {
    "details": "http://.../root/table/1"
  },
  {
```

```

    "details": "http://.../root/table/2"

  },

  {

    "primaryKey": "./oid=3"

  },

  {

    "foreignKey": "4"

  },

  ...

]

```

以下も参照してください。:

- [Delete 操作](#)

• フィールド

ターゲットフィールドの内容を指定します。このリクエストは、フィールドの更新時に使用されます。

リクエストは、[ノード値](#)で定義されているものと同じ構造です。

以下も参照してください。:

- [Update 操作](#)

• ユーザーインターフェイスを開くまたは閉じる

ユーザーインターフェイスが開いているか閉じているか、および使用不可メッセージを指定します。

```
{  
  
  "toolStatus": {  
  
    "key": true,      // or false  
  
    "label": "Open"  
  
  },  
  
  "toolStatusCloseMessage": "Access is temporarily forbidden for  
maintenance."  
  
}
```

以下も参照してください。:

- 。ユーザーインターフェイスの操作

リクエストで言及できるのは書き込み可能なフィールドのみです。これには、次の場合は含まれません。

- 関連付けノード
- 選択ノード
- 値関数
- セッターを持たない JavaBean フィールド
- 認証されたユーザーのノードに対する書き込み不可能なアクセス許可

JSON 応答本文

応答の本文は JSON `オブジェクト` で表され、その内容は操作とカテゴリによって異なります。

データカテゴリ

- データセットツリー

テーブル ノードと非ターミナル グループ ノードの階層が含まれます。

```
{
  "hasChildren": false,
  "rootName": {
    "settings": {
      "weekTimeSheet": {},
      "vacationRequest": {}
    },
    "table1": {},
    "table2": {}
  }
}
```

以下も参照してください。:

- [Select 操作](#)

• データセットノード

指定されたノードの下のターミナルノードのリストが含まれます。

```
{
  "nodeName1": true,
  "nodeName2": 2,
  "nodeName3": "Hello"
```

```
}
```

以下も参照してください。:

- [Select 操作](#)

• データセットのリスト

それぞれ指定されたデータスペースまたはデータセットのルートデータセットまたは子データセットのリストを含む JSON `オブジェクト`。

すべてのデータセット JSON `オブジェクト` には次のプロパティがあります。

JSON プロパティ	JSON 形式	説明	必須
<code>key</code>	<code>String</code>	データセットのフォーマット設定されたキーを指定します。形式: <code>[a-zA-Z_-][a-zA-Z0-9_]*</code>	はい
<code>isSelectAllowed</code>	<code>Boolean</code>	ユーザーの権限に従ってデータセットを選択できるかどうかを指定します。	はい
<code>hasChildren</code>	<code>Boolean</code>	ユーザーの権限に従って、データセットに子があるかどうかを指定します。	はい

```
{
  "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"

}

}

```

以下も参照してください。:

- 。データセットの選択

• データセット情報

次のプロパティを含む JSON オブジェクト

JSON プロパティ	JSON 形式	説明	必須
<code>key</code>	<code>String</code>	データセットのフォーマット設定されたキー値を指定します。形式： <code>[a-zA-Z_-][-a-zA-Z0-9_.*]*</code> 、最大 64 文字に制限されます。	はい
<code>dataModel</code>	JSON オブジェクト	データセットスキーマの場所を指定します。モジュールスキーマにパッケージ化されたものとパブリケーションのみがサポートされます。	はい
<code>module</code> (<code>dataModel</code> プロパティの下にネスト)	<code>String</code>	データセットスキーマが配置されているモジュール名を指定します。	いいえ
<code>pathInModule</code> (<code>dataModel</code> プロパティの下にネスト)	<code>String</code>	データセットスキーマが配置されているモジュール内のパスを指定します。	いいえ
<code>publication</code> (<code>dataModel</code> プロパティの下にネスト)	<code>String</code>	データセットスキーマのパブリケーション名を指定します。	いいえ
<code>dataModel</code> (<code>dataModel</code> プロパティの下にネスト)	<code>String</code>	データセットスキーマパブリケーションのデータモデル名を指定します。	いいえ
<code>versionName</code> (<code>dataModel</code> プロ	<code>String</code>	データセットスキーマパブリケーション	いいえ

JSON プロパティ	JSON 形式	説明	必須
パティの下にネスト)		のバージョン名を指定します。	
<code>parent</code>	<code>String</code>	親データセットのフォーマット設定されたキー値を指定します。親がない場合は、 <code>null</code> 値が保持されます。	はい
<code>activated</code>	<code>Boolean</code>	データセットがアクティブ化されているかどうかを指定します。	はい
<code>owner</code>	<code>String</code>	所有者プロファイルを指定します。	はい
<code>creator</code>	<code>String</code>	作成者プロファイルを指定します。	はい
<code>creation_time</code>	<code>Date</code>	作成時刻を指定します。	はい
<code>updater</code>	<code>String</code>	アップデータープロファイルを指定します。	はい
<code>update_time</code>	<code>Date</code>	更新時刻を指定します。	はい

```
{
  "key": "mydataset",
  "dataModel": {
    "module": "modulename",
    "pathInModule": "/WEB-INF/ebx/schemas/mydataset.xsd"
```

```

    },

    "parent": null,

    "activated": true,

    "owner": "Beverly",

    "creator": "Uadmin",

    "creation_time": "2023-10-10T15:49:03.023",

    "updater": "Uadmin",

    "update_time": "2023-10-10T15:49:03.005"

}

```

以下も参照してください。:

- 。データセットの選択

• テーブル

次のプロパティを含む JSON オブジェクト

- 。選択したレコードの JSON 配列 に対応する 行。各レコードは、JSON オブジェクト で表されます。レコードが選択されていない場合、JSON 配列 は空です。
- 。(オプション) ページ付け データを含むページ付け。

```

{

    "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"
}
}

```

以下も参照してください。:

- [Select 操作](#)

• レコード

レコードコンテンツを含む JSON `オブジェクト`。

```
{  
  
  "pk": "pk1",  
  
  "name": "Name1",  
  
  "name-fr": "Name1",  
  
  "parent": null  
  
}
```

以下も参照してください。:

- [Select 操作](#)
- [準備操作](#)

• フィールド

選択ノードを除き、現在のノード値が含まれます。

以下も参照してください。:

- [Select 操作](#)

• ユーザーインターフェイスの状態の取得

ユーザーインターフェイスのステータスと使用不可メッセージが含まれます。

```
{
  "toolStatus": {
    "key": true,      // or false
    "label": "Open"
  },
  "toolStatusCloseMessage": "Access is temporarily forbidden for
maintenance."
}
```

以下も参照してください。:

- 。ユーザーインターフェイスの操作

注意:

ノード、レコード、フィールドのプロパティと値は、解決された権限 ([権限を参照](#)) によっては非表示になる場合があります。

コンテンツ

このセクションは常に含まれており、追加のフィールドなしでマスターデータが含まれています。

ノード値

ノード値には、データまたはラベルと、列挙の場合は詳細リンクのみが含まれます。関連付けと選択を除くすべてのノードで使用できます。

単純型のコンテンツ

ノードのコンテンツを記述するキーと値の JSON エントリに対応します。

以下も参照してください。:

- 単純型形式

グループ、リスト、列挙の内容

XML スキーマ	JSON 形式	例
グループ <code>xs:complexType</code>	<code>Object</code> サブノードごとのプロパティが含まれます。	単純オカレンスグループの例 <pre>{ "road": "11 rue scribe", "zipcode": "75009", "country": "France" }</pre>
List <code>maxOccurs > 1</code>	<code>Array</code> JSON <code>オブジェクト</code> または単純型で表されるすべてのフィールドオカレンスの配列が含まれます。 各 JSON <code>オブジェクト</code> は、<code>ノード値</code> で構成されています。	<code>xs:int</code> 型の複数オカレンスフィールドの例 <pre>[0, 1, 2, 3, 4]</pre> 複数オカレンスグループの例

XML スキーマ	JSON 形式	例
		<pre>[{ "road": "11 rue scribe", "zipcode": "75009", "country": "France" }, { "road": "711 Atlantic Ave", "zipcode": "MA 02111", "country": "United States" }]</pre>
列挙型 <code>xs:string</code>	<code>Object</code> <code>key</code>、<code>link</code>、および <code>label</code> プロパティが含まれています。	外部キーの例 <pre>{ "key": "1", "details": "http://.../Bdataspace/ "label": "Française" }</pre>

共通

- [はじめに](#)
- [グローバル構造](#)

- JSON 応答本文
- フォーム検証レポート
- コンテンツ
 - ページ付け
 - 単純型のコンテンツ
 - Insert 操作レポート
 - Delete 操作レポート
 - 技術データ
- 更新モード
- 既知の制限事項
 - フィールド値
 - インデックス作成と検索戦略

はじめに

共通の仕様は、[コンパクト形式](#)と[拡張形式](#)の両方の形式で使用されます。

グローバル構造

JSON 応答本文

フォームデータカテゴリ

応答本文には、処理されたリソースごとに JSON `オブジェクト` が含まれます。つまり、単一の処理済みリソースの JSON ルート `オブジェクト` は、複数の処理済みリソースの JSON `オブジェクト` です。いくつかのプロパティは、JSON `オブジェクト` に直接配置されます。

- テーブル/データセットリクエストの単一のレコードリクエストまたは更新フィールドの挿入/更新が成功した場合、`code` JSON プロパティが存在しない可能性があり、そうでない場合は、処理されたリソースに関連付けられた HTTP 応答コードに対応します。検証レポートは常に存在します。一部のプロパティはオプションであり、リクエストパラメーターを使用して要求されたときに追加されます。詳細については、[フォームデータ操作](#)を参照してください。

```
{
  "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"
}
]
}

```

- 複数レコードの挿入リクエストが成功した場合、`code` JSON プロパティが存在しない可能性があり、そうでない場合は、処理されたリソースに関連付けられた HTTP 応答コードに対応します。検証レポートは常に存在します。一部のプロパティはオプションであり、リクエストパラメーターを使用して要求されたときに追加されます。

```
{
  "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"

  }

]

}

]

}

```

- テーブル/データセット内の単一のレコードリクエストまたは更新フィールドの挿入/更新が失敗した場合、応答本文は JSON **例外処理** 応答に対応します。

```

{

  "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"

  }

]

}

```

- 複数レコードの挿入リクエストが失敗した場合、応答本文は JSON **例外処理** 応答に対応します。

```

{

  "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"
}
]
}

```

以下も参照してください。:

- [フォーム検証レポート](#)
- [例外処理](#)

フォーム検証レポート

このレポートは、[フォームデータ操作](#)を使用して取得できます。要求されたリソースに対するすべての検証制約を提供します。

これらの制約には、次のプロパティがあります。

JSON プロパティ	JSON 形式	説明	必須
<code>message</code>	<code>String</code>	制約の説明。	はい
<code>blocksCommit</code>	<code>String</code>	制約の制御ポリシー。可能な値は、 <code>onInsertUpdateOrDelete</code>	はい

JSON プロパティ	JSON 形式	説明	必須
		、 <code>onUserSubmit-checkModifiedValues</code> 、 <code>never</code> です。詳細については、 ブロッキング制約と非ブロッキング制約 を参照してください。	
<code>level</code>	<code>String</code>	制約の重大度。可能な値は、 <code>info</code> 、 <code>warning</code> 、 <code>error</code> 、または <code>fatal</code> です。	はい
<code>pathInDataset</code>	<code>String</code>	スキーマノードから始まる相対フィールドパス。	いいえ (**)
<code>pathInRecord</code>	<code>String</code>	テーブルノードから始まる相対フィールドパス。	いいえ (*)

(*) レコードおよびレコードフィールド操作でのみ使用できます。

(**) データセット操作でのみ使用できます。

以下も参照してください。:

- [フォームデータカテゴリ](#)

コンテンツ

ページ付け

ページ付けを使用して、パラメーター化可能な制限された数のデータを返します。ページ付けは、レコード、関連付け値、選択ノード値、セクター、データスペース、およびデータセットのデータ型に適用できます。 `pagination` という名前のコンテキストが常に返されます。このコンテキストを使用すると、UI と同様の方法でデータを参照できます。

ページ付けは常に有効になっています。

以下も参照してください。:

- [Select 操作](#)
- [データセットの選択](#)
- [ベータ機能：データスペースまたはスナップショットの選択](#)

以下に、このコンテキストに関連する詳細情報を示します。

JSON プロパティ	JSON 形式	説明	必須
<code>firstPage</code>	<code>String</code> または <code>null (*)</code>	最初のページにアクセスするための URL。	はい
<code>previousPage</code>	<code>String</code> または <code>null (*)</code>	前のページにアクセスするための URL。	はい
<code>nextPage</code>	<code>String</code> または <code>null (*)</code>	次のページにアクセスするための URL。	はい
<code>lastPage</code>	<code>String</code> または <code>null (*)</code>	最後のページにアクセスするための URL。	はい

注意:

(*) データがこのコンテキストで利用可能かどうかのみを定義し、応答ではデータが利用可能かどうかを定義しません。

単純型のコンテンツ

XML スキーマ	JSON 形式	例	メタタイプ	OPENAPI
xs:string xs:Name osd:text	String (Unicode 文 字。RFC4627 を参照)	"A text" "The escape of \"special character\" is preceded by a backslash." null	string name text	型: string 形式: n/a
osd:html	String (Unicode 文 字。RFC4627 を参照)	" An HTML tag can thus be written without trouble "	html	型: string 形式: html
osd:email	String (Unicode 文 字。RFC4627 を参照)	"employee@mycom_email_m"		型: string 形式: email
osd:locale	String (言語 タグ。 RFC1766 を参 照)	"en-US"	locale	型: string 形式: locale
xs:string (外部キー)	String フォーマット が設定された 外部キーの値 が含まれま す。	"0" "true 99"	foreignKey	型: string 形式: n/a
xs:boolean	Boolean	true false	boolean	型: boolean 形式: n/a

XML スキーマ	JSON 形式	例	メタタイプ	OPENAPI
		null		
xs:decimal	数値または null	-10.5 20.001 15 -1e-13	decimal	型: number 形式: double
xs:date	"yyyy-MM-dd" 形式の String	"2015-04-13"	date	型: string 形式: date
xs:time	以下の形式の String "HH:mm:ss" "HH:mm:ss.SSS"	"11:55:00" "11:55:00.000"	time	型: string 形式: date-time
xs:dateTime	以下の形式の String "yyyy-MM-ddTHH:mm:ss" "yyyy-MM-ddTHH:mm:ss.SSS"	"2015-04-13T11:55:00" "2015-04-13T11:55:00.000"	dateTime	型: string 形式: date-time
xs:anyURI	String (URI。 RFC3986 を参照)	"https://fr.wikipedia.org/wiki/René_Descartes"	anyURI	型: string 形式: uri
xs:int xs:integer	Number または null	1596	int	型: integer 形式: int32

XML スキーマ	JSON 形式	例	メタタイプ	OPENAPI
<code>osd:resource</code>	<code>String</code> リソースフォーマットの値が含まれます。	"ebx-tutorial:ext-images:frontpages/Bach.jpg"	<code>resource</code>	型: <code>string</code> 形式: <code>n/a</code>
<code>osd:color</code>	形式 "#[A-Fa-f0-9]{6}" の <code>String</code> 色のフォーマットが設定された値が含まれます。	"#F6E0E0"	<code>color</code>	型: <code>string</code> 形式: <code>n/a</code>
<code>osd:datasetName</code>	形式: "[a-zA-Z_][a-zA-Z0-9_]*"、最大 64 文字の <code>String</code> データセット名のフォーマット設定値が含まれます。	"ebx-tutorial"	<code>dataset</code>	型: <code>string</code> 形式: <code>n/a</code>
<code>osd:dataspaceKey</code>	[BV][a-zA-Z0-9_:.\\-\\]+ 形式、最大 33 文字の <code>String</code> データスペースのフォーマット設定	"Bebx-tutorial"	<code>dataspace</code>	型: <code>string</code> 形式: <code>n/a</code>

XML スキーマ	JSON 形式	例	メタタイプ	OPENAPI
	キー値が含まれます。			

Insert 操作レポート

レコードテーブルを使用して Insert 操作を呼び出す場合、オプションでレポートを返すことができます。レポートには、次のプロパティを含む JSON オブジェクトが含まれています。

- `count` には、JSON `Number` 型の `int` が含まれています。これは、リストされた行がすべての場合に網羅されていないため、処理されたレコードの数を定義します。
- `isPartialList` には、JSON `Boolean` 型の `boolean` が含まれます。リストされた行が処理されたレコード全体の部分的なリストにすぎない場合は `true`、そうでない場合は `false` です。
- `rows` には JSON `Array` が含まれ、各エレメントはリクエストエレメントの結果に対応します。
- `code` には JSON `Number` 型の `int` が含まれており、レコードが挿入されたか更新されたかを確認できます。このプロパティは、`updateOrInsert` パラメーターが `true` に設定されている場合にのみ含まれます。
- `foreignKey` には、このレコードの外部キーとして使用されるコンテンツに対応する、JSON `String` 型の `文字列` が含まれています。このプロパティは、パラメーター `includeForeignKey` が `true` に設定されている場合にのみ含まれます。
- `label` には JSON `String` 型の `文字列` が含まれており、レコードラベルを取得できます。このプロパティは、パラメーター `includeLabel` が `yes` に設定されている場合にのみ含まれます。
- リソース URL に対応する JSON `String` 型の `文字列` を含む `details`。このプロパティは、パラメーター `includeDetails` が `true` に設定されている場合にのみ含まれます。

```
{
  "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"

    }

]

}

```

以下も参照してください。:

- [Insert 操作](#)

Delete 操作レポート

Delete 操作を呼び出すと、レポートが返されます。レポートには、次のプロパティを含む JSON オブジェクトが含まれています。

- JSON `Number` 型の整数を含む `deletedCount` は、削除されたレコードの数に対応します。
- JSON `Number` 型の整数を含む `occultedCount` は、非表示にされたレコードの数に対応します。

- JSON `Number` 型の整数を含む `inheritedCount` は、継承されたレコードの数に対応します。

```
{
  "deletedCount": 1,
  "inheritedCount": 0,
  "occultedCount": 0
}
```

以下も参照してください。:

- [Delete 操作](#)

技術データ

返された各レコードには、技術データに対応する `ebx-metadata/system` プロパティの下にシステムレコードメタデータを含めることができます。システムメタデータは、要求および並べ替えが可能です。

レコードのメタデータを含めるには、選択操作の `includeMetadata=system` パラメーター値を参照してください。

JSON プロパティ	JSON 形式	説明	必須
<code>uuid</code>	<code>String</code>	レコードのユニバーサル固有識別子。	はい
<code>creator</code>	<code>String</code>	作成ユーザーの識別子。詳細については、 <code>UserReference</code> を参照してください。	はい
<code>creation_time</code>	<code>String</code>	作成日	はい

JSON プロパティ	JSON 形式	説明	必須
<code>updater</code>	<code>String</code>	最終更新ユーザーの識別子。詳細については、 <code>UserReference</code> を参照してください。	はい
<code>update_time</code>	<code>String</code>	最終更新日。	はい

拡張形式

```
{
  ...,
  "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"

        }

    }

}

}

}

```

コンパクト形式

```

{

    ...,

    "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"

```

```

    }

}

}

```

以下も参照してください。:

- [楽観ロック](#)

更新モード

`byDelta` モードでは、JSON ソースドキュメントから欠落しているデータモデルエレメントを無視できます。このモードは、RESTful 操作によって (デフォルトで) 有効になります。次の表は、エレメントがソースドキュメントに含まれていない場合の Insert および Update 操作の動作をまとめたものです。

詳細については、Java API の RESTful データサービス [Update 操作](#)と [Insert 操作](#)、および `ImportSpec.setByDelta` を参照してください。

JSON ソースドキュメントの状態	動作
プロパティがソースドキュメントに存在しない	<code>byDelta</code> モードがアクティブになっている場合 (デフォルト) • <code>Update</code> 操作の場合、フィールド値は変更されません。 • <code>Insert</code> 操作の場合、動作は <code>byDelta</code> モードが無効になっている場合と同じです。 <code>byDelta</code> モードがRESTful 操作パラメーターによって無効になっている場合 ターゲットフィールドは、次のいずれかの値に設定されます。

JSON ソースドキュメントの状態	動作
	• エレメントがデフォルト値を定義している場合、ターゲットフィールドはそのデフォルト値に設定されます。 • エレメントが文字列またはリスト以外のタイプの場合、ターゲットフィールド値は <code>null</code> に設定されます。 • エレメントが集約リストの場合、ターゲットフィールド値は空のリスト値に設定されます。 • エレメントが <code>null</code> と空の文字列を区別する文字列の場合、ターゲットフィールドの値は <code>null</code> に設定されます。2 つを区別しない文字列の場合は、空の文字列です。 • エレメント (単純または複合) がデータサービスで非表示になっている場合、ターゲット値は変更されません。 以下も参照してください。: ◦ データサービスでのフィールドの非表示 注意: インポートを実行するユーザーは、ターゲットフィールド値を作成または変更するために必要な権限を持っている必要があります。それ以外の場合、操作は中止されます。
エレメントが存在し、その値が <code>null</code> (例: <code>"content":null</code>)	ターゲットフィールドは、リストを除いて常に <code>null</code> に設定されます。リストの場合、サポートされません。

既知の制限事項

フィールド値

フィールド `xs:date`、`xs:time`、および `xs:dateTime` の値には、JSON プリミティブ型に関連付けられたタイムゾーンは含まれません。

インデックス作成と検索戦略

メタモデルからの `filterable` および `sortable` の値は、デフォルトの検索戦略に限定されます。

以下も参照してください。:

- [検索](#)

その他

- [はじめに](#)
- [グローバル構造](#)
 - [JSON リクエスト本文](#)
 - [JSON 応答本文](#)

はじめに

拡張および [コンパクト JSON](#) 形式はほとんどのユースケースを処理できますが、一部の操作と REST カテゴリには次のような特定の形式が必要です。

グローバル構造

JSON リクエスト本文

リクエストの本文は JSON `オブジェクト` で表され、その内容は操作とカテゴリによって異なります。

認証カテゴリ

リクエスト本文には、ルート JSON `オブジェクト` に直接配置されたいくつかのプロパティが含まれています。

- [トークンの作成](#)

認証トークンの作成試行に使用する `login` と `password` を指定します。

```
{  
  
  "login": "...",                // JSON String  
  
  "password": "...",            // JSON String  
  
}
```

`specific` 属性を指定して、認証トークンの作成を試行するために、HTTP リクエストに対してユーザー認証をアクティブにします。

```
{  
  
  "specific": true                // JSON Boolean  
  
}
```

以下も参照してください。:

- [トークンの作成操作](#)

• パスワードの変更

パスワードの変更に使用する `login`、`password`、および `passwordNew` を指定します。

```
{  
  
  "login": "...",                // JSON String  
  
  "password": "...",            // JSON String  
  
  "passwordNew": "...",         // JSON String  
  
}
```

以下も参照してください。:

- 。パスワード変更操作

JSON 応答本文

応答の本文は JSON `オブジェクト` で表され、その内容は操作とカテゴリによって異なります。

管理者カテゴリ

このカテゴリの Select 操作では、`content` プロパティで要求された値のみが提供されます。

・ システム情報

EBX® インスタンスのシステム情報が含まれています。これらのデータの表現は、フラットまたは階層的です。

フラット表現:

```
{
  "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

}

}

}

}

```

階層表現:

```

{

    "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

}

}

}

}

```

以下も参照してください。:

- システム情報の操作

認証カテゴリ

応答本文には、ルートJSON **オブジェクト** に直接配置されたいくつかのプロパティが含まれています。

• トークンの作成

トークン値とそのタイプが含まれます。

```

{

    "accessToken": "...",           // JSON String

    "tokenType": "..."           // JSON String

```

}

以下も参照してください。:

- 。 トークンの作成操作

データカテゴリ

テーブルビューの検索

応答の本文には、JSON `Array` を保持する `content` プロパティが含まれています。このプロパティ自体は、次のプロパティを持つ JSON `オブジェクト` で構成されています。

JSON プロパティ	JSON 形式	説明	必須
<code>details</code>	<code>String</code>	ビューアクセス URL に対応します。	はい
<code>label</code>	<code>String</code>	ビューのラベル。	いいえ
<code>viewPublication</code>	<code>String</code>	公開されたビューの名前。 以下も参照してください。: テーブルの <code>viewPublication</code> パラメーター	はい
<code>viewType</code>	<code>String</code>	値が次のいずれかに対応する列挙。	はい

JSON プロパティ	JSON 形式	説明	必須
		• <code>SimpleTabular</code> ：単純な表形式のビュー • <code>Hierarchy</code> : 階層ビュー • <code>Tile</code> : タイルビュー	

```
{
  "content": [
    {
      "details": "http://.../data/v1/Bebx-directory/ebx-directory/
directory/users?viewPublication=custom-directory",
      "label": "My custom directory view",
      "viewPublication": "custom-directory",
      "viewType": "SimpleTabular"
    },
    {
      ...
    }
  ]
}
```

以下も参照してください。:

- ルックアップテーブルビュー操作

ベータ機能：データスペースの選択

返される応答本文には、`rows` JSON `Array` プロパティにデータスペースが含まれます。ここで、各内部JSON `オブジェクト` は、次のプロパティを持つデータスペースに対応します。

JSON プロパティ	JSON 形式	説明	必須
<code>label</code>	<code>String</code>	現在のロケールのドキュメントラベル。	いいえ
<code>description</code>	<code>String</code>	現在のロケールでのドキュメントの説明。	いいえ
<code>details</code>	<code>String</code>	データスペースの REST リソース URL を指定します。	はい
<code>information</code>	<code>String</code>	データスペースの情報 REST リソース URL を指定します。	はい
<code>key</code>	<code>String</code>	データスペースまたはスナップショット形式のキーを指定します。 フォーマット： <code>[BV][a-zA-Z0-9_:. \-\\]</code> <code>{1,33}</code> およびその後 にエンコードされたパーセント。	はい

JSON プロパティ	JSON 形式	説明	必須
<code>isSelectAllowed</code>	Boolean	ユーザーの権限に従って、データスペースを選択できるかどうかを指定します。	はい
<code>hasChildren</code>	Boolean	ユーザーの権限に従って、データスペースに子があるかどうかを指定します。 注意: スナップショットには適用されません	はい
<code>children</code>	String	データスペースの子 REST リソース URL を指定します。 <code>hasChildren</code> プロパティキーの値が <code>false</code> の場合、戻り値は <code>null</code> です。 注意: スナップショットには適用されません	いいえ
<code>snapshots</code>	String	データスペースのスナップショット REST リソース	はい

JSON プロパティ	JSON 形式	説明	必須
		URL を指定します。 注意: スナップショットには適用されません	

```

{
  "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

  }

}

```

以下も参照してください。:

- ページ付け
- ベータ機能：データスペースまたはスナップショットの選択

ベータ機能：データスペース情報

応答本文には、次のプロパティを保持する `content` JSON オブジェクト プロパティが含まれています。

JSON プロパティ	JSON 形式	説明	必須
<code>content</code>	<code>Object</code>	データスペースまたはスナップショット情報に対応します。	はい
<code>key</code>	<code>String</code>	データスペースまたはスナップショット	はい

JSON プロパティ	JSON 形式	説明	必須
		形式のキー値を指定します。 形式： <code>[BV][a-zA-Z0-9_:\-\]+</code>。 最大 33 文字に制限されています。	
<code>documentation</code>	JSON オブジェクトの Array	ロケールごとに JSON オブジェクトを使用してローカライズされたドキュメントに対応します。	いいえ
<code>locale</code>	String	ドキュメントロケール (<code>documentation</code> プロパティの下にネストされています)。	はい
<code>label</code>	String	ドキュメントラベル (<code>documentation</code> プロパティの下にネストされています)。	いいえ
<code>description</code>	String	ドキュメントの説明 (<code>documentation</code> プロパティの下にネストされています)。	いいえ

JSON プロパティ	JSON 形式	説明	必須
<code>closed</code>	Boolean	データスペースを閉じるかどうかを指定します。	はい
<code>locked</code>	Boolean	データスペースをロックするかどうかを指定します。 注意: スナップショットには適用されません	はい
<code>parent</code>	String	親データスペースまたはスナップショット形式のキー値を指定します。	いいえ
<code>administration</code>	Boolean	データスペースまたはスナップショットが管理用のものであるかどうかを指定します。	はい
<code>owner</code>	String	所有者プロファイルを指定します。	いいえ
<code>creator</code>	String	作成者プロファイルを指定します。	はい
<code>creationDate</code>	Date	作成日を指定します。	はい

```
{
  "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"

}

```

}

以下も参照してください。:

- ベータ機能：データスペースまたはスナップショットの選択

ベータ機能：データスペースの子またはスナップショットの作成

応答本文には、次のプロパティを保持する `content` JSON オブジェクト プロパティが含まれています。

JSON プロパティ	JSON 形式	説明	必須
<code>key</code>	<code>String</code>	データスペースまたはスナップショット形式のキー値を指定します。 形式: <code>[BV][a-zA-Z0-9_:\-\]+</code>。 最大 33 文字に制限されています。 デフォルト値はタイムスタンプです。	いいえ
<code>owner</code>	<code>String</code>	所有者プロファイルを指定します。 デフォルト値は <code>null</code> です。	いいえ
<code>documentation</code>	JSON オブジェクトの <code>Array</code>	ロケールごとに JSON オブジェクトを使用してローカライズされたド	いいえ

JSON プロパティ	JSON 形式	説明	必須
		キュメントに対応します。 デフォルト値は <code>null</code> です。	
<code>locale</code>	<code>String</code>	ドキュメントロケール (<code>documentation</code> プロパティの下にネストされています)。	はい
<code>label</code>	<code>String</code>	ドキュメントラベル (<code>documentation</code> プロパティの下にネストされています)。 デフォルト値は <code>null</code> です。	いいえ
<code>description</code>	<code>String</code>	ドキュメントの説明 (<code>documentation</code> プロパティの下にネストされています)。 デフォルト値は <code>null</code> です。	いいえ
<code>dataspaceKeyToCopyFrom</code>	<code>String</code>	権限のコピー元となるデータスペース形式のキー値を指定します。 注意:	いいえ

JSON プロパティ	JSON 形式	説明	必須
		スナップショットの作成には適用されません	

```
{
  "content": {
    "key": "BMyData",
    "owner": "Beveryone",
    "documentation": [
      {
        "locale": "en-US",
        "label": "My dataspace",
        "description": "This space contains my data"
      }
    ],
    "dataspaceKeyToCopyPermissionsFrom": "BReference"
  }
}
```

以下も参照してください。:

- ベータ機能：子データスペースまたはスナップショットの作成

ヘルスカテゴリ

正常性起動および確認応答

応答本文には、**OK** 値に対応する JSON 文字列値が含まれています。

ヘルス停止応答

応答本文には、列挙値に対応する JSON 文字列値が含まれています。可能な値：**OK** または

NO

EBX® の SQL

- はじめに
- 比較演算子
- 算術演算子と関数
- 論理演算子
- 文字列の演算子と関数
- 日付と時刻の関数
- EBX® SQL 関数

はじめに

- 概要
- データモデルエンティティのマッピング
 - テーブル (データモデル内)
 - フィールド
 - グループ
 - 関連付け
- データ型のマッピング
 - 単純データ型
 - EBX® で定義された拡張単純型
 - リスト (複数値) タイプ
 - 複合型
 - TableRef 型

- メタデータフィールド
- システム列
- SQL 構文
 - サポートされている標準の演算子と関数
 - 識別子のエスケープ
 - プラン情報
- 制限とパフォーマンスのガイドライン

概要

このドキュメントでは、EBX® の構造化照会言語 (SQL) のクエリと式について説明します。EBX® は、1 つ以上のテーブルから選択された行を取得するための標準 SQL クエリをサポートしています。一部の EBX® SQL 言語機能は、標準の拡張機能です。サポートされている EBX® SQL 構文には、テーブル式 (SELECT、FROM、WHERE、GROUP BY、および HAVING 句)、DISTINCT、ORDER BY、LIMIT、および OFFSET、クエリの組み合わせ (UNION)、および WITH (RECURSIVE修飾子を含む) が含まれます。

この API の目的は、よく知られた標準を使用して EBX® からデータを取得する機能を開発者に提供することです。

EBX® SQL は Java API を介して、特に `Query` クラスからアクセスできます。クエリはパラメーターもサポートしています。 `Query.setParameter` を参照してください。

クエリには、単一のデータセットまたは複数のデータセットを含めることができます。クエリの作成方法の詳細については、 `QueryBuilder` を参照してください。

データモデルエンティティのマッピング

次のセクションでは、EBX® の概念の SQL へのマッピングについて詳しく説明します。

テーブル (データモデル内)

EBX® テーブルは、SQL テーブルに自然にマッピングされます。データモデルには、同じ名前の EBX® テーブルが複数存在する場合があります。このあいまいさは、テーブルがグループにある場合に発生する可能性があります。あいまいさを取り除くには、二重引用符で囲まれたテーブルのフルパスを使用します (たとえば、 `"my_group/my_table"` は `"other_group/my_table"` と競合しなくなります)。データモデル内で一意のテーブルのエンティティ名を使用することもできます。テーブル名は、エンティティ名または別のテーブル名と競合しない場合にのみ使用できます。

以下も参照してください。:

- [Data Model Assistant を使用したエンティティマッピング](#)

フィールド

SQL 標準では、テーブルの構造は 1 つ以上の列で構成されます。親が EBX® テーブルであるすべてのエレメント (フィールドを含む) は、列にマップされます。

グループ

SQL 標準では、テーブルの構造は 1 つ以上の列で構成されます。親が EBX® テーブルであるすべてのエレメントは、列にマップされます。これには、SQL 構造タイプとして SQL 列にマッピングされるグループが含まれます。

関連付け

SQL 標準では、複数のテーブル間でのデータのクエリは、外部キーと主キーに基づいています。EBX® のこれらの概念は、SQL の概念と似ています。したがって、SQL のテーブル間の結合は、EBX® 外部キーと主キーを使用して実行することもできます。

データ型のマッピング

SQL を介したデータの処理は、そのデータ型に大きく依存します。たとえば、述語では、列が同じ SQL データ型を持っている場合にのみ列を比較できます。SQL データ型は、データモデルの型に応じた型です。

単純データ型

サポートされている標準 SQL データ型

この表は、EBX® でサポートされている XML スキーマで定義されているすべての単純型と、それに対応する標準 SQL 型を示しています。

XML スキーマタイプ	SQL タイプ	JAVA タイプ	注意
<code>xs:string</code>	VARCHAR	<code>java.lang.String</code>	
<code>xs:boolean</code>	ブール値	<code>java.lang.Boolean</code>	値: TRUE、FALSE、UNKNOWN

XML スキーマタイプ	SQL タイプ	JAVA タイプ	注意
<code>xs:decimal</code>	10 進数	<code>java.math.BigDecimal</code>	
<code>xs:dateTime</code>	タイムスタンプ	<code>java.util.Date</code>	
<code>xs:time</code>	時間	<code>java.util.Date</code>	返される <code>Date</code> の日付部分は常に「1970/01/01」に設定されます。
<code>xs:date</code>	日付	<code>java.util.Date</code>	返される <code>Date</code> の時間部分は、常に 1 日の始まり (つまり、「00:00:00」) に設定されます。
<code>xs:anyURI</code>	VARCHAR	<code>java.net.URI</code>	
<code>xs:name</code> (<code>xs:string</code> restriction)	VARCHAR	<code>java.lang.String</code>	
<code>xs:int</code>	整数	<code>java.lang.Integer</code>	

EBX® で定義された拡張単純型

EBX® は、事前定義された単純なデータ型を提供します。これらのタイプは、内部スキーマ `common-1.0.xsd` によって定義されます。それらの定義については、[EBX® で定義された拡張単純型](#)のセクションで詳しく説明しています。

XML スキーマタイプ	SQL タイプ	JAVA クラス
<code>osd:text</code> (<code>xs:string</code> restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:html</code> (<code>xs:string</code> restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:email</code> (<code>xs:string</code> restriction)	VARCHAR	<code>java.lang.String</code>

XML スキーマタイプ	SQL タイプ	JAVA クラス
<code>osd:password</code> (<code>xs:string</code> restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:color</code> (<code>xs:string</code> restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:resource</code> (<code>xs:anyURI</code> restriction)	VARCHAR	内部クラス
<code>osd:locale</code> (<code>xs:string</code> restriction)	VARCHAR	<code>java.util.Locale</code>
<code>osd:dataspaceKey</code> (<code>xs:string</code> restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:datasetName</code> (<code>xs:string</code> restriction)	VARCHAR	<code>java.lang.String</code>

リスト (複数値) タイプ

リストは SQL 配列として処理されます。対応する Java クラスは `java.util.List` です。

注意：複数値フィールドでフィルタリングするには、まず `UNNEST` SQL 演算子を使用して「正規化」する必要があります (これはパフォーマンスの観点から推奨されず、大規模なテーブルでは非常に非効率になる可能性があることに注意してください)。

- `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'`

複合型

複合型は **SQL 構造化型**として扱われます。対応する Java クラスは `Object[]` です。これは、外部キー (以下を参照) と **グループ**に適用されます。これらは、複合型によって定義されているためです。ドット演算子を使用して、SQL 構造型内のフィールドにアクセスします。たとえば、複合型の場合は、`address.street` を使用してフィールド `address` のフィールド `street` にアクセスします。クエリで複合型のサブフィールドを参照する場合は、常にテーブル名またはエイリアスを使用する必要があります。

- `SELECT customer.address.street FROM customer`

- `SELECT c.address.street FROM customer c`

TableRef 型

EBX® では、テーブルに複数のフィールドで構成される主キーを含めることができます。外部キーは、`osd:tableRef` 宣言のある単一のフィールドによって定義されます。標準の SQL 構文が拡張され、対象の主キーフィールドの値が抽出されるようになりました。外部キーの抽出の例では、次の SQL 式が有効です。

- `tableA.fkb.id = 123`
- `YEAR(tableA.fkb.date) > 2018`

注意:

主キーが 1 つのフィールドのみで構成されている場合でも、値にアクセスするにはフィールドの名前を指定する必要があります。たとえば、主キーが単一の `id` で構成されている場合、`tableA.fkb.id = 123` のように、値にアクセスするには `fkb.id` を使用する必要があります。

tableRef の使用法の典型的な例は、SQL 結合です。

```
SELECT * FROM employee JOIN department ON employee.fkDept.id = department.id
```

または、参照されるテーブルに複合主キーがある場合

```
SELECT * FROM tableA JOIN tableB ON tableA.fkB.id1 = tableB.id1 AND  
tableA.fkB.id2 = tableB.id2
```

同様に、`FK_AS_STRING` ビルトイン関数と `$pk` システム列を使用して、キーの文字列表現を介して同じ JOIN を実行できます。

```
SELECT * FROM tableA JOIN tableB ON FK_AS_STRING(tableA.fkB) = tableB."$pk"
```

メタデータフィールド

EBX® SQL は、一部のメタデータ情報へのアクセスも提供します。これらのフィールドは、明示的に参照されない限り、SQL ステートメントによって返されません。たとえば、「`SELECT * FROM ...`」はメタデータフィールドを返しません、「`SELECT metadataField FROM ...`」は返します。

名前	説明	SQL タイプ	JAVA クラス	例
creator	レコードの作成者。	VARCHAR	java.lang.String	<pre>SELECT t."ebx- metadata"."system"."creat FROM myTable t SELECT t.* FROM myTable t WHERE t."ebx- metadata"."system"."creat = 'User1' SELECT t.* FROM myTable t ORDER BY t."ebx- metadata"."system"."creat</pre>
creation_time	レコードの作成時刻。	タイムスタンプ	java.util.Date	<pre>SELECT t."ebx- metadata"."system"."creat FROM myTable t SELECT t.* FROM myTable t WHERE t."ebx- metadata"."system"."creat > TIMESTAMP'2001-06-10 14:57:20' SELECT t.* FROM myTable t ORDER BY t."ebx- metadata"."system"."creat</pre>
updater	レコードを変更した最後のユーザー。	VARCHAR	java.lang.String	<pre>SELECT t."ebx- metadata"."system"."updat FROM myTable t SELECT t.* FROM</pre>

名前	説明	SQL タイプ	JAVA クラス	例
				<pre>myTable t WHERE t."ebx- metadata"."system"."update = 'User1' SELECT t.* FROM myTable t ORDER BY t."ebx- metadata"."system"."update</pre>
update_time	レコードの最終変更時刻。	タイムスタンプ	java.util.Date	<pre>SELECT t."ebx- metadata"."system"."update FROM myTable t SELECT t.* FROM myTable t WHERE t."ebx- metadata"."system"."update > TIMESTAMP '2001-06-10 14:57:20' SELECT t.* FROM myTable t ORDER BY t."ebx- metadata"."system"."update</pre>

システム列

テーブルに存在するフィールドとは別に、EBX® SQL はいくつかの追加のシステム列を提供します。これらの列は、明示的に参照されていない限り、SQL ステートメントによって返されません。たとえば、「`SELECT * FROM ...`」はシステム列を返しません、「`SELECT systemColumnName FROM ...`」は返します。

名前	説明	SQL タイプ	JAVA クラス	例
\$adaptation	テーブルレコードを表す	内部型	com.onwbp.adapt	<pre>SELECTdaptation t."\$adaptation"</pre>

名前	説明	SQL タイプ	JAVA クラス	例
	Adaptation 。			FROM myTable t WHERE t.value>100
\$pk	レコードの主キーの文字列表現。 以下も参照してください。: • PrimaryKey.syntax	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"

SQL 構文

サポートされている標準の演算子と関数

演算子は、比較や算術演算などの演算を実行するために、主に SQL ステートメントの WHERE 句で使用される予約語または文字です。これらの演算子は、SQL ステートメントで条件を指定し、ステートメントで複数の条件の接続詞として機能するために使用されます。EBX® は、ほとんどの

SQL 標準演算子と関数をサポートしています。一部の関数と演算子は、オプションのパラメーターを持つことができます。ドキュメントでは、それらは角括弧で囲まれています。一般に、演算子と関数には 5 つのタイプがあります。

- 比較演算子と関数
- 算術演算子と関数
- 論理演算子
- 文字列の演算子と関数
- 日付と時刻の関数

次の表は、すべての演算子の結合性と優先度を、高い順に示しています。

演算子	結合性
.	左
(配列エレメント)	左
+ - (単項プラス、マイナス)	右
* / %	左
+ -	左
BETWEEN、IN、LIKE、CONTAINS など	-
< > = <= >= <> !=	左
IS NULL、IS FALSE、IS NOT TRUE など	-
NOT	右
AND	左
OR	左

識別子のエスケープ

次の場合、識別子は二重引用符を使用してエスケープする必要があります。

- 絶対パスを使用してテーブルを識別する場合 (たとえば、「/root/myTable」)。
- 識別するフィールドが予約語の場合 (「user」、「order」など)。

- テーブルエイリアス (たとえば、t."*\$adaptation*", t."*\$pk*" でシステム列を参照する場合)。
- メタデータフィールドを参照する場合 (たとえば、t."*ebx-metadata*".*"system"*.*"creation_time"*)。

次の例は、すべてのケースを例示するためのクエリを示しています。

```
SELECT t."user", t."$pk" FROM "/root/myTable" t WHERE t."order" = 1
```

プラン情報

EBX® SQL は `EXPLAIN PLAN FOR ...` 構文をサポートして、クエリのプラン情報を取得します。

結果は `Query.explain` に似ています。

例： `EXPLAIN PLAN FOR SELECT id FROM myTable`

制限とパフォーマンスのガイドライン

- 内部結合の最適化では、RIGHT 結合と FULL 結合がサポートされていないため、可能であればこれらの結合タイプは避けてください。
- 数値または 10 進値の最大精度とスケールは 1000 です。
- `GROUP BY` や集約関数 (`MIN`、`MAX` など) を使用したクエリは最適化されませんが、`COUNT` は状況に応じて最適化されます。
- 現在、`MIN` および `MAX` 演算子は内部インデックスを利用していません。代わりに、次の同等のクエリを使用してください。多くの場合、これらはより効率的です。

`SELECT val FROM myTable ORDER BY val DESC NULLS LAST LIMIT 1` の代わりに

`SELECT MAX(val) FROM myTable`

`SELECT val FROM myTable ORDER BY val LIMIT 1` の代わりに `SELECT MIN(val) FROM myTable`

比較演算子

次の表に、EBX® でサポートされているすべての SQL 算術演算子と関数、およびそれらの標準 SQL 構文を示します。一部の関数にはオプションのパラメーターがある場合があります。それらは角括弧で囲まれています。

演算子の構文	説明と例
value1 = value2	等しい <code>SELECT 1 = 0 : false</code> <code>SELECT 4 = 4 : true</code>
value1 <> value2	等しくない <code>SELECT 1 <> 0 : true</code> <code>SELECT 4 <> 4 : false</code>
value1 > value2	より大きい <code>SELECT 1 > 0 : true</code> <code>SELECT 4 > 4 : false</code>
value1 >= value2	以上 <code>SELECT 1 >= 0 : true</code> <code>SELECT 4 >= 4 : true</code>
value1 < value2	より小さい <code>SELECT 1 < 0 : false</code> <code>SELECT 4 < 4 : false</code>
value1 <= value2	以下 <code>SELECT 1 <= 0 : false</code> <code>SELECT 4 <= 4 : true</code>
value IS NULL	値が null かどうか <code>SELECT 1 IS NULL : false</code> <code>SELECT NULL IS NULL : true</code>
value IS NOT NULL	値が null でないかどうか <code>SELECT 1 IS NOT NULL : true</code> <code>SELECT NULL IS NOT NULL : false</code>
value1 IS DISTINCT FROM value2	2つの値が等しくないかどうか (null 値を同じものとして扱う) <code>SELECT 1 IS DISTINCT FROM 1 : false</code> <code>SELECT 1 IS DISTINCT FROM 4 : true</code> <code>SELECT 1 IS DISTINCT FROM NULL : true</code> <code>SELECT NULL IS DISTINCT FROM NULL : false</code>
value1 IS NOT DISTINCT FROM value2	2つの値が等しいかどうか (null 値を同じものとして扱う) <code>SELECT 1 IS NOT DISTINCT FROM 1 : true</code> <code>SELECT 1 IS NOT DISTINCT FROM 4 : false</code> <code>SELECT 1 IS NOT DISTINCT FROM NULL : false</code> <code>SELECT NULL IS NOT DISTINCT FROM NULL : true</code>

演算子の構文	説明と例
value1 BETWEEN value2 AND value3	value1 が value2 以上、かつ value3 以下かどうか SELECT 4 BETWEEN 3 AND 10 : true SELECT 1 BETWEEN 3 AND 10 : false
value1 NOT BETWEEN value2 AND value3	value1 が value2 以上、かつ value3 以下かどうか SELECT 4 NOT BETWEEN 3 AND 10 : false SELECT 1 NOT BETWEEN 3 AND 10 : true
string1 LIKE string2	string1 がパターン string2 と一致するかどうか。ワイルドカード '%' は、0 個、1 個、または複数の文字を表します。ワイルドカード '_' は単一文字を表します。エスケープ文字を定義するには、'ESCAPE' を使用します (ワイルドカードをエスケープする場合)。string1 がシーケンスで始まるかどうかを確認するには、パターン 'sequence%' を使用します (同様に、シーケンスで終わる場合は '%sequence' を使用)。照合では大文字と小文字が区別されます。大文字と小文字を区別しない照合を実行するには、string1 と pattern で UPPER (または LOWER) を使用します。 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	string1 がパターン string2 と一致しないかどうか。ワイルドカード '%' は、0 個、1 個、または複数の文字を表します。ワイルドカード '_' は単一文字を表します。エスケープ文字を定義するには、'ESCAPE' を使用します (ワイルドカードをエスケープする場合)。string1 がシーケンスで始まらないかどうかを確認するには、パターン 'sequence%' を使用します (同様に、シーケンスで終わらない場合は '%sequence' を使用)。照合では大文字と小文

演算子の構文	説明と例
	字が区別されます。大文字と小文字を区別しない照合を実行するには、string1 と pattern で UPPER (または LOWER) を使用します。 <pre>SELECT name FROM employee WHERE name NOT LIKE 'S%' : Hamilton 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</pre>
value IN (value *)	値がリスト内の値と等しいかどうか <pre>SELECT firstname FROM employee WHERE name IN ('Smith', 'Hamilton') : John, Maria</pre>
value NOT IN (value *)	値がリスト内のすべての値と等しくないかどうか <pre>SELECT firstname FROM employee WHERE name NOT IN ('Smith', 'Hamilton') : Jennifer</pre>
value IN (sub-query)	値がサブクエリによって返される行と等しいかどうか <pre>SELECT e.firstname FROM employee e WHERE e.department.id IN (SELECT d.id FROM department d WHERE d.name='IT') : John</pre>

算術演算子と関数

次の表は、EBX® でサポートされているすべての SQL 算術演算子と関数を、それらの標準 SQL 構文と共にリストしています。一部の関数は、オプションのパラメーターを持つことができます。それらは角括弧で囲まれています。

演算子の構文	説明と例
+ numeric	数値を返す <pre>SELECT +4 : 4</pre>
- numeric	数値を返す <pre>SELECT -4 : -4</pre>

演算子の構文	説明と例
numeric1 + numeric2	numeric1 と numeric2 を加算した値を返す SELECT 18 + 4 : 22
numeric1 - numeric2	numeric1 から numeric2 を減算した値を返す SELECT 18 - 4 : 14
numeric1 * numeric2	numeric1 と numeric2 を乗算した値を返す SELECT 18 * 4 : 72
numeric1 / numeric2	numeric1 を numeric2 で除算した値を返す SELECT 18 / 4 : 4
ABS(numeric)	数値の絶対値を返す SELECT ABS(-243.5) : 243.5
ACOS(numeric)	数値の逆余弦を返す SELECT ACOS(-1) : 3.141592653589793
ASIN(numeric)	数値の逆正弦を返す SELECT ABS(0.25) : 0.25
ATAN(numeric)	数値の逆正接を返す SELECT ATAN(2.5) : 1.107148717774778
ATAN2(numeric, numeric)	数値座標の逆正接を返す SELECT ATAN2(0.50, 1) : 0.4636476090008061
CEIL(numeric)	数値を切り上げて、数値以上の最小の数値を返す SELECT CEIL(25.75) : 26
CEILING(numeric)	数値を切り上げて、数値以上の最小の数値を返す SELECT CEILING(25.75) : 26
COS(numeric)	数値のコサインを返す SELECT COS(2) : -0.4161468365471424
COT(numeric)	数値のコタンジェントを返す SELECT COT(6) : -3.436353004180128
DEGREES(numeric)	数値をラジアンから度に変換 SELECT DEGREES(1.5) : 85.94366926962348

演算子の構文	説明と例
EXP(numeric)	e の累乗を返す <code>SELECT EXP(1) :</code> <code>2.718281828459045</code>
FLOOR(numeric)	数値を切り捨て、数値以下の最大の数値を返す <code>SELECT FLOOR(25.75) : 25</code>
LN(numeric)	数値の自然対数 (底 e) を返す <code>SELECT</code> <code>LN(2) : 0.6931471805599453</code>
LOG10(numeric)	数値の 10 を底とする対数を返す <code>SELECT</code> <code>LOG10(2) : 0.3010299956639812</code>
MOD(numeric1, numeric2)	numeric1 を numeric2 で除算した剰余 (あまり) を返す。numeric1 が負の場合のみ結果は負 <code>SELECT MOD(18, 4) : 2</code>
POWER(numeric1, numeric)	numeric1 を numeric2 で累乗して返す <code>SELECT POWER(4, 2) : 16.0</code>
RADIANS(numeric)	数値を度からラジアンに変換 <code>SELECT</code> <code>RADIANS(180) : 3.141592653589793</code>
RAND()	指定された場合、シード値として数値を使用してランダムな倍精度浮動小数点数を返す <code>SELECT RAND(6) : 0.9891840064573959</code>
RAND_INTEGER(numeric)	0 から数値 -1 までのランダムな整数を生成し、シードが指定されている場合はそのシードで乱数ジェネレータを初期化 <code>SELECT</code> <code>RAND_INTEGER(100, 5) : 0</code> <code>SELECT</code> <code>RAND_INTEGER(6, 100) : 11</code>
ROUND(numeric1, numeric2)	numeric1 を小数点以下 numeric2 桁に丸めます <code>SELECT ROUND(135.375, 2) : 135.38</code>
SIGN(numeric)	数値の符号を返す <code>SELECT SIGN(255.5) :</code> <code>1</code>
SIN(numeric)	数値の正弦を返す <code>SELECT SIN(2) :</code> <code>0.9092974268256817</code>

演算子の構文	説明と例
SQRT(numeric)	数値の平方根を返す <code>SELECT SQRT(64) :</code> <code>8.0</code>
TAN(numeric)	数値の正接を返す <code>SELECT TAN(1.75) :</code> <code>-5.52037992250933</code>
TRUNCATE(numeric1)	numeric1 を小数点以下 0 桁 (numeric2 が指定 されていない場合) に切り捨てます <code>SELECT</code> <code>TRUNCATE(135.375) :</code> <code>135</code> <code>SELECT</code> <code>TRUNCATE(135.375, 2) :</code> <code>135.37</code>

論理演算子

次の表は、EBX® でサポートされているすべての SQL 論理演算子と、それらの標準 SQL 構文を示しています。

演算子の構文	説明と例
boolean1 OR boolean2	boolean1 が TRUE または boolean2 が TRUE かどうか <code>SELECT TRUE OR TRUE :</code> <code>true</code> <code>SELECT TRUE OR FALSE :</code> <code>true</code> <code>SELECT</code> <code>FALSE OR TRUE :</code> <code>true</code> <code>SELECT FALSE OR</code> <code>FALSE :</code> <code>false</code>
boolean1 AND boolean2	boolean1 が TRUE かつ boolean2 が TRUE か どうか <code>SELECT TRUE AND TRUE :</code> <code>true</code> <code>SELECT TRUE AND FALSE :</code> <code>false</code> <code>SELECT FALSE AND TRUE :</code> <code>false</code> <code>SELECT FALSE AND FALSE :</code> <code>false</code>
NOT boolean	ブール値が TRUE でないかどうか。ブール値 が UNKNOWN の場合は UNKNOWN を返しま す。 <code>SELECT NOT TRUE :</code> <code>false</code> <code>SELECT</code> <code>NOT FALSE :</code> <code>true</code> <code>SELECT NOT</code> <code>UNKNOWN :</code> <code>null</code>
boolean IS FALSE	ブール値が FALSE かどうか。ブール値が UNKNOWN の場合は FALSE を返します。 <code>SELECT TRUE IS FALSE :</code> <code>false</code> <code>SELECT</code> <code>FALSE IS FALSE :</code> <code>true</code>

演算子の構文	説明と例
boolean IS NOT FALSE	ブール値が FALSE でないかどうか。ブール値が UNKNOWN の場合は TRUE を返します。 <pre>SELECT TRUE IS NOT FALSE : true</pre> <pre>SELECT FALSE IS NOT FALSE : false</pre>
boolean IS TRUE	ブール値が TRUE かどうか。ブール値が UNKNOWN の場合は TRUE を返します。 <pre>SELECT TRUE IS TRUE : true</pre> <pre>SELECT FALSE IS TRUE : false</pre>
boolean IS NOT TRUE	ブール値が TRUE でないかどうか。ブール値が UNKNOWN の場合は FALSE を返します。 <pre>SELECT TRUE IS NOT TRUE : false</pre> <pre>SELECT FALSE IS NOT TRUE : true</pre>
boolean IS UNKNOWN	ブール値が UNKNOWN かどうか <pre>SELECT TRUE IS UNKNOWN : false</pre> <pre>SELECT FALSE IS UNKNOWN : false</pre>
boolean IS NOT UNKNOWN	ブール値が UNKNOWN ではないかどうか。 <pre>SELECT TRUE IS NOT UNKNOWN : true</pre> <pre>SELECT FALSE IS NOT UNKNOWN : true</pre>

文字列の演算子と関数

次の表は、EBX®でサポートされているすべての SQL 文字列演算子と関数を、それらの標準 SQL 構文と共に一覧表示しています。一部の関数は、オプションのパラメーターを持つことができます。それらは角括弧で囲まれています。

演算子の構文	説明と例
string	
CHAR_LENGTH(string)	文字列内の文字数を返します。 <pre>SELECT CHAR_LENGTH('Alfreds Futterkiste') : 19</pre>
CHARACTER_LENGTH(string)	As CHAR_LENGTH(string) <pre>SELECT CHARACTER_LENGTH('Alfreds Futterkiste') : 19</pre>

演算子の構文	説明と例
UPPER(string)	大文字に変換された文字列を返します。 <pre>SELECT UPPER('SQL Tutorial is FUN!') : SQL TUTORIAL IS FUN!</pre>
POSITION(string1 IN string2)	指定されたポイントから始まる string2 内の string1 の最初の出現位置を返します。 <pre>SELECT POSITION('is fun' IN 'Tutorial is FUN!') : 0</pre> <pre>SELECT POSITION('fun' IN 'Tutorial is FUN, very FUN!' FROM 17) : 0</pre>
TRIM({ BOTH	LEADING
OVERLAY(string1 PLACING string2 FROM integer)	string1 の部分文字列を string2 に置き換えます <pre>SELECT OVERLAY('Tutorial is very FUN!' PLACING 'VERY' FROM 13) : Tutorial is VERY FUN!</pre> <pre>SELECT OVERLAY('Tutorial is very FUN!' PLACING 'VERY' FROM 13 FOR 4) : Tutorial is VERY FUN!</pre>
SUBSTRING(string FROM integer)	指定されたポイントから始まる文字列の部分文字列を返します。 <pre>SELECT SUBSTRING('Tutorial is very FUN!' FROM 13) : very FUN!</pre> <pre>SELECT SUBSTRING('Tutorial is very FUN!' FROM 13 FOR 4) : very</pre>

日付と時刻の関数

次の表は、EBX®でサポートされているすべての SQL 日付および時刻関数と、それらの標準 SQL 構文を示しています。一部の関数には、オプションのパラメーターがある場合があります。それらは角括弧で囲まれています。

演算子の構文	説明と例
CURRENT_TIME	データ型 TIMESTAMP WITH TIME ZONE の値で、セッションタイムゾーンの現在の時刻を返します。

演算子の構文	説明と例
CURRENT_DATE	データ型 DATE の値で、セッションタイムゾーンの現在の日付を返します。
CURRENT_TIMESTAMP	データ型 TIMESTAMP WITH TIME ZONE の値で、セッションタイムゾーンの現在の日付と時刻を返します。
文字列をタイムスタンプにキャスト	EBX (UI およびエクスポートファイル) で表示される日時値はローカルタイムゾーンであるのに対し、SQL パーサーによって作成されたタイムスタンプリテラルは UTC であることに注意してください。したがって、サーバーが UTC でない場合は、この不一致を回避して期待どおりの結果を得るために、ローカルタイムゾーンでリテラルを作成する必要がある場合があります (3 番目の例を参照)。フィルターを作成するには、日時値がミリ秒の精度 (EBX UI では表示されない) かどうかを考慮する必要があります (最後の例を参照)。 <pre> 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- 09:56:55' AND e."ebx- metadata"."system"."update_time"<TIMESTAMP'2023-08- 09:56:56' : 1 </pre>
YEAR(date)	日時値式から年の値を抽出して返します。整数を返します。 <pre> SELECT YEAR(TIMESTAMP '1971-07-20 09:34:21') : 1971 SELECT YEAR(DATE '1968-07-20') : 1968 SELECT YEAR(hiringDate) FROM employee </pre>

演算子の構文	説明と例
	<pre>WHERE name='Smith' : 2015 SELECT YEAR(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 2017</pre>
QUARTER(date)	日時値式から四半期の値を抽出して返します。1 から 4 までの整数を返します。 <pre>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</pre>
QUARTER(date)	日時値式から月の値を抽出して返します。1 から 12 までの整数を返します。 <pre>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</pre>
WEEK(date)	日時値式から週の値を抽出して返します。1 から 53 までの整数を返します。 <pre>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</pre>
DAYOFYEAR(date)	日時値式から日付 (年単位) の値を抽出して返します。1 から 366 までの整数を返します。 <pre>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</pre>

演算子の構文	説明と例
DAYOFMONTH(date)	日時値式から日 (月単位) の値を抽出して返します。1 から 31 までの整数を返します。 <pre>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</pre>
DAYOFWEEK(date)	日時値式から曜日の値を抽出して返します。1 から 7 までの整数を返します。 <pre>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</pre>
HOUR(date)	日時値式から時間の値を抽出して返します。0 から 23 までの整数を返します。 <pre>SELECT HOUR(TIMESTAMP '1971-07-20 09:34:21') : 9 SELECT HOUR(TIMESTAMP '10:34:21') : 10 SELECT HOUR(workStart) FROM employee WHERE name='Smith' : 8 SELECT HOUR(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 16</pre>
MINUTE(date)	日時値式から分の値を抽出して返します。0 から 59 までの整数を返します。 <pre>SELECT MINUTE(TIMESTAMP '1971-07-20 09:34:21') : 34 SELECT MINUTE(TIMESTAMP '10:35:21') : 35 SELECT MINUTE(workStart) FROM employee WHERE name='Smith' : 0 SELECT MINUTE(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 6</pre>
SECOND(date)	日時値式から秒の値を抽出して返します。0 から 59 までの整数を返します。 <pre>SELECT</pre>

演算子の構文	説明と例
	<pre> 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 </pre>

EBX® SQL 関数

次の表に、すべての EBX® ビルトイン SQL 関数とその構文を示します。一部の関数には、オプションのパラメーターがある場合があります。それらは角括弧で囲まれています。

演算子の構文	説明と例
FK_AS_STRING(tableRefValue)	tableRef 値の文字列表現を返します。この関数は集約リストでは使用できません。パラメーターを 1 つ取る FK_AS_STRING は、明確なコンテキストでのみ使用できます。他のシナリオでは、たとえば UNION または UNNEST の後の特定のクエリでは、3 つのパラメーターを使用する必要があります (以下を参照)。 <pre> 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 </pre>
FK_AS_STRING('referencedDatasetName', 'referencedTableName', tableRefValue)	tableRef 値の文字列表現を返します。この関数は集約リストでは使用できません。参照されるデータセット名の詳細については、「QueryBuilder JavaDoc」を参照してください。 <pre> 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 </pre>

REST Toolkit

- はじめに
- アプリケーションの定義
- サービスと操作の定義
 - URL とサンプル
- テーブルレコードのシリアル化
 - ビルトインシリアライザー
 - カスタムシリアライザー
- 認証とルックアップのメカニズム
- REST 認証と権限
- URI ビルダー
 - ビルトインの URI ビルダー
 - リソースの URI ビルダー
 - サーバーとアプリケーションの URI ビルダー
- 例外処理
- 監視
- パッケージ化と登録
- OpenAPI
 - 概要
 - アクティベーション
 - 構成
 - 操作
 - ドキュメントの強化
 - 権限
 - ビジュアルドキュメント
- 制限事項

はじめに

TIBCO EBX® は、REST Toolkit を使用してカスタム REST サービスを開発する可能性を提供します。REST Toolkit は、[Jakarta RESTful Web Services 3.1](#) および [JSR-367: JSON-B](#) をサポートします。

REST サービスは Java クラスによって実装され、その操作は Java メソッドによって実装されます。応答は、POJO オブジェクトをシリアル化することで生成できます。リクエスト入力、POJO にシリアル化解除できます。JSON を含むさまざまな入力および出力形式がサポートされています。サポートされている形式の詳細については、[メディアタイプ](#)を参照してください。

REST Toolkit は以下をサポートします。

- 注入可能なオブジェクト

EBX® は、リクエストのユーザーの認証、EBX® リポジトリへのアクセス、または構成を気にせずに URI ([リバースプロキシモード](#)や [REST フォワードモード](#)など) を構築するために役立つ注入可能なオブジェクトを提供します。

JAX-RS 注入可能オブジェクトもサポートされています。

- アノテーション

EBX® は、リソースを説明したり、匿名アクセスを許可したり、メソッドに制限を追加したりするためのアノテーションを提供します。

JAX-RS および JSON-B アノテーションもサポートされています。

- ロギングおよびデバッグユーティリティ。

以下も参照してください。:

- [Jakarta RESTful Web Services 3.1 API 仕様](#)

アプリケーションの定義

カスタム REST サービスを含む EBX® モジュールは、少なくとも 1 つの REST Toolkit アプリケーションクラスを提供する必要があります。REST Toolkit アプリケーションクラスは、EBX®

`RESTApplicationAbstract` クラスを拡張します。最小要件は、`@ApplicationPath` アノテ

ションと REST サービスクラスをスキャンするパッケージのセットを使用して、ベース URL を定義することです。

注意:

スキャンできるのは、Web アプリケーションのクラスローダーからアクセスできるパッケージのみです。

注意:

`ApplicationConfigurator.register(java.lang.Class)` または `ApplicationConfigurator.register(java.lang.Object)` を介して、Web アプリケーションアーカイブの内部または外部にパッケージ化された REST リソースクラスまたはシングルトンを登録できます。

注意:

パッケージスコープが定義されていない場合、Web アプリケーションのクラスローダーから到達可能なすべてのクラスがスキャンされます。

アプリケーションパスを「/」にすることはできません。また、モジュールの既存のリソースと競合することはできません。「/rest」 (`RESTApplicationAbstract.REST_DEFAULT_APPLICATION_PATH` 定数の値) を使用することをお勧めします。

EBX® `Documentation` アノテーションはオプションです。これは、>、またはログ記録とデバッグ時に管理者に表示されます。

```
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()));
    }
}

```

サービスと操作の定義

REST Toolkit サービスは Java クラスによって実装され、その操作はそのメソッドによって実装されます。

クラスとメソッドに `@Path` アノテーションを付けて、アクセスパスを指定できます。クラスレベルで定義された `@Path` アノテーション値は、メソッドで定義された値の前に追加されます。警告：クラスまたはメソッドごとに許可される `@Path` アノテーションは 1 つだけです。

リソースによって受け入れられ、生成されるメディアタイプは、それぞれ `@Consumes` および `@Produces` JAX-RS アノテーションによって定義されます。サポートされているメディアタイプは次のとおりです。

- `application/json` (`MediaType.APPLICATION_JSON_TYPE`)
- `application/octet-stream` (`MediaType.APPLICATION_OCTET_STREAM_TYPE`)
- `application/x-www-form-urlencoded` (`MediaType.APPLICATION_FORM_URL_ENCODED_TYPE`)

- `multipart/form-data` (`MediaType.MULTIPART_FORM_DATA_TYPE`)
- `text/css`
- `text/html` (`MediaType.TEXT_HTML_TYPE`)
- `text/plain` (`MediaType.TEXT_PLAIN_TYPE`)

有効な HTTP(S) メソッドは、JAX-RS アノテーション `@GET`、`@POST`、`@PUT` などによって指定されます。これらのアノテーションの 1 つのみ各 Java メソッドに設定できます (これは、Java メソッドがサポートできる HTTP メソッドは 1 つだけであることを意味します)。

警告： 名前の前に `ebx-` が付いた URL パラメーターは、REST Toolkit によって予約されており、EBX® ドキュメントで明示的に許可されていない限り、カスタム REST サービスで定義しないでください。

URL とサンプル

サンプルの記述サービスにアクセスするための REST URL は次のように定義されています。

```
http[s]://<host>[:<port>]/<path to webapp>/rest/track/v1/description
```

説明

- `<path to webapp>` は、REST Toolkit アプリケーションを保持する Web アプリケーションのパスに対応し、それ自体がサンプルサービスを提供します。パスは、複数の URI セグメント、または Web アプリケーションの名前が続く URI セグメントで構成されます。

注意:

`/rest/track/v1/description` は、アプリケーションの `@ApplicationPath` とサービスの `@Path` アノテーションの連結に対応していることに注意してください。

次の REST Toolkit サービスサンプルは、トラックデータをクエリおよび管理するためのメソッドを提供します。

```
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).v
alue();

}

```

```

/**

 * 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);
    }
}

```

この REST サービスは、データ転送オブジェクト (DTO) を表す次の Java クラスを使用して、データをシリアル化および逆シリアル化します。

```

/**
 * 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;

    }

}

```

テーブルレコードのシリアル化

ビルトインシリアライザー

デフォルトの JSON シリアライザーとデシリアライザーは、DTO で `ContentHolder` として宣言されたときにテーブルレコードを処理するために提供されています。[拡張](#)と[コンパクト](#)の両方の JSON 形式のレコードがサポートされています。

```

/**

 * 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;

}

```

同じ DTO をシリアル化と逆シリアル化に使用できます。シリアル化の場合、`ContentHolderForInput` インスタンスが自動的に作成され、適切なデータが入力されます。その後、このインスタンスはそのデータを `ValueContextForUpdate` にコピーできるようになります。テーブルレコードを逆シリアル化するには、`ContentHolderForOutput` を REST 操作 JAVA メソッドで作成し、返す必要があります。提供された `Adaptation` データは、JSON の有効なピースに変換され、HTTP 応答本文に配置されます。

```

/**

 * 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); ;

aSingerDT0.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());

    }

}

}

```

応答のデフォルトの JSON 形式は、ビジネスデータフィールドのみで構成され、コンパクトと呼ばれます。

```

{

  "singer":{

    "firstname":"Frank",

    "lastname":"Sinatra"

  }

}

```

技術フィールドまたはメタデータを追加するには、`ExtendedOutput` アノテーションを `ContentHolder` フィールドの上に配置する必要があります。アノテーションは、必要なすべてのオプションまたは ALL オプションを宣言する必要があります。

```

/**

 * 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"

    }

}

}

}

```

入力 HTTP リクエストに拡張 JSON 形式を使用するには、`ExtendedInput` アノテーションを `ContentHolder` フィールドの上に配置する必要があります。

```

/**

 * 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;

}

```

カスタムシリアライザー

TIBCO EBX® は JSON-B ([JSR-367](#)) に基づいているため、カスタムシリアライザーとデシリアライザーは `JsonbTypeSerializer` および `JsonbTypeDeserializer` アノテーションを使用して定義できます。

デフォルトでは、`java.math.BigDecimal` は JSON `String` にシリアル化されます。この動作をオーバーライドするには、カスタムシリアライザーを JSON `Number` に実装します。

```
/**
 * 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 vinyl.
 */
public final class VinylDTO
{
    @JsonbTypeSerializer(BigDecimalSerializerInJsonNumber.class)
    public BigDecimal price;

    @JsonbTypeSerializer(CustomTrackDtoSerializer.class)
    @JsonbTypeDeserializer(CustomTrackDtoDeserializer.class)
    public TrackDTO track;
}
```

認証とルックアップのメカニズム

REST Toolkit で開発されたカスタム REST サービスは、ビルトインの REST データサービスと同じ認証方法とルックアップメカニズムをサポートします。ただし、「匿名認証スキーム」については、`AnonymousAccessEnabled` を使用することで範囲を広げることができるため、わずかな違いがあります。詳細については、[REST 認証と権限](#)を参照してください。

以下も参照してください。:

- [認証](#)
- [ルックアップメカニズム](#)

REST 認証と権限

デフォルトでは、すべての REST リソース Java メソッドでユーザーを認証する必要があります。

ただし、一部のメソッドは匿名でアクセスできる必要がある場合があります。これらのメソッドには、`AnonymousAccessEnabled` というアノテーションを付ける必要があります。

一部のメソッドは、特定のプロファイルに制限する必要がある場合があります。これらのメソッドには、認証ルールを指定するために `Authorization` というアノテーションを付けることができます。認証ルールは、`AuthorizationRule` インターフェイスを実装する Java クラスです。

```
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()

{

    ...

}

}

```

URI ビルダー

REST Toolkit は、URI を生成するためのユーティリティインターフェイス `URIInfoUtility` を提供します。このインターフェイスのインスタンスには、注入可能なビルトインオブジェクト `SessionContext` を介してアクセスできます。

ビルトインの URI ビルダー

いくつかの URI ビルダーインターフェイスは、ビルトイン RESTful サービスの URI ビルドを容易にするように設計されています。各インターフェイスは、同じ機能概念に関連するメソッドの集合体を構成します。この分割により、モジュール式および汎用アルゴリズムの開発が可能になります。これらのインターフェイスの一部は、それ自体が他のインターフェイスの集合体であり、メソッド呼び出しの一貫した組み合わせのみが許可されているため、ビルダーを直感的に使用できます。たとえば、REST **階層** カテゴリ用に構成された URI ビルダーは、レコードアクセスが REST **データ** カテゴリの概念の一部であるため、レコード URI ビルドメソッドの呼び出しを許可しません。

さらに、これらの URI ビルダーは、次のように事前構成されています。

- 基本構成を構成する着信 HTTP リクエストからのデータ。
- EBX® モジュールセグメントの前に追加される**モジュールのパブリックパスプレフィックス** (存在する場合)。
- **データサービスリネージ設定**は、以前の構成全体をオーバーライドします (存在する場合)。
- **REST フォワードモード**は、URI のパスのピースのみがアクティブ化されている場合に保持されるため、以前の構成を部分的にオーバーライドします。

詳細については、`URIBuilderForBuiltin` および `CategoryURIBuilder` を参照してください。

リソースの URI ビルダー

リソースアクセス URI を生成する URI ビルダーも利用できます。リソースは 1 つの機能概念としてのみ見なされるため、すべてのメソッドは同じインターフェイスで定義されています。

ビルトインデータサービスの URI ビルダーと同様に、これらのサービスは次のように事前構成されています。

- 基本構成を構成する着信 HTTP リクエストからのデータ。
- **外部リソース設定**は、以前の構成のすべてまたは一部をオーバーライドします (存在する場合)。
- EBX® モジュールセグメントが存在する場合はその先頭に追加する**モジュールのパブリックパスプレフィックス**。

サーバーとアプリケーションの URI ビルダー

以前のビルダーでカバーされていないサービスまたはリソースへの URI をビルドする必要がある場合は、デフォルトの事前構成済み URI ビルダーを使用する必要があります。終了パスセグメントのみが異なる 2 つのデフォルト URI ビルダーがあります。最初のパスはサーバーのセグメント (EBX® モジュールセグメントの直前) で終了し、2 番目のパスは REST アプリケーションの最後のセグメント (`@ApplicationPath` アノテーションで定義) で終了します。

これらのデフォルトの URI ビルダーは、次のように事前構成されています。

- 基本構成を構成する着信 HTTP リクエストからのデータ。
- EBX® モジュールセグメントが存在する場合はその先頭に追加する **モジュールのパブリックパスプレフィックス**。

例外処理

REST Toolkit Java メソッドは、JAX-RS クラス `jakarta.ws.rs.WebApplicationException` を拡張する Java 例外をスローすることで、標準の HTTP エラー応答を生成できます。JAX-RS は、さまざまな HTTP ステータスコードの例外を定義します。EBX® は、HTTP `422` (*Unprocessable entity*) コードのサポートを追加する `UnprocessableEntityException` を定義します。

プレーン Java 例外は、HTTP ステータスコード `500` (*内部サーバーエラー*) にマップされます。

カスタム REST サービスが `"application/json"` Java アノテーションで `@Produces` メディアタイプを宣言している場合、例外の内容は以下の例に示すように返されます。その他のサポートされているメディアタイプの場合、応答本文は返されず、REST サービスロギングカテゴリは関連する警告メッセージを生成します。

```
{
    "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"
    }
]
}

```

以下も参照してください。:

- [メディアタイプ](#)

REST Toolkit は [ExceptionHandler](#) の実装を提供しますが、以下の例に示すように、独自の
実装に置き換えることができます。

```

import jakarta.annotation.*;

import jakarta.ws.rs.core.*;

import jakarta.ws.rs.ext.*;

@Provider

@Priority(4999)

public class CustomExceptionHandler implements ExceptionMapper<Throwable>

{

    @Override

```

```

    public Response toResponse(final Throwable anException)

    {

        ...

    }

}

```

特定のメソッドは、JAX-RS [Response](#) を直接返すことで管理できます。

以下も参照してください。:

- `ApplicationConfigurator.register`

監視

REST Toolkit イベントの監視は、データサービスのログ構成に似ています。違いは、`ebx.log4j.category.log.restServices` にする必要があるプロパティキーです。

以下も参照してください。:

- [監視](#)
- [EBX® ログの構成](#)

ログメッセージを構成するために、いくつかの追加のプロパティを使用できます。詳細については、[REST Toolkit サービスの構成](#)を参照してください。

パッケージ化と登録

すべてのアプリケーションとコンポーネントは、モジュールの Web アプリケーション (warファイル) にパッケージ化する必要があります。

JAX-RS クライアント API を除く JAX-RS ライブラリは、すでに `ebx.jar` に含まれているため、war ファイルにパッケージ化しないでください。

詳細については、[Jakarta EE の展開](#) を参照してください。

REST Toolkit アプリケーションの登録は、EBX® モジュール登録プロセスに統合されています。登録クラスは `ModuleRegistrationListener` を継承し、サーブレット 3.0 アノテーション `@WebListener` を宣言し、`handleContextInitialized` メソッドをオーバーライドする必要があります。

詳細については、[モジュール登録](#) を参照してください。

```
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);

    }

}
```

OpenAPI

概要

TIBCO EBX® ネイティブ統合によって生成された OpenAPI ドキュメントは、[OpenAPI V3 仕様](#)に準拠しています。REST Toolkit アプリケーションの利用可能な REST リソースと関連する操作を構造化して記述することにより、これらのドキュメントは、その開発と消費を容易にします。ただし、現在サポートされているのは、JSON 形式のみです。

アクティベーション

OpenAPI ドキュメントの生成を有効にするには、次のことが必要です。

- REST Toolkit アプリケーションクラスで、`OpenApiApplicationPath` アノテーションを設定する必要があります。このアノテーションには、OpenAPI 操作のベース URI の構成で使った一意のパスの定義が必要です。

注意:

OpenAPI アプリケーションパスの後には、ジェネレーターバージョンが続きます (`/open-api-application-path/v1`)。ジェネレーターのバージョンと open-api の仕様のバージョンが混同されるため、`/open-api` のようなパスを使用しないことが推奨されます。

- ドキュメント作成時に考慮すべきは、REST Toolkit のクラスリソースに `OpenApiResource` を設定することです。

構成

パラメーター `ebx.restservices.openApi.unpublished.applications` を使用して、特定の環境で OpenAPI エンドポイントを公開しないようにすることができます。

```
#####

# Comma-separated list of canonical application class names

# whose OpenAPI descriptions will not be published.
```



```
#####

ebx.restservices.openApi.unpublished.applications=com.example.RestApplicat
ion
```

パラメーター `ebx.restservices.openapi.style.short.name` を使用して、生成されたスキーマの名前を簡略化できます。

```
#####

# 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
```

操作

OpenAPI 操作は、GET または POST HTTP メソッドを使用して、アプリケーション全体または単一のサービスの JSON ドキュメントを生成します。

アプリケーションの URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/<open-api-path>/v1
```

1 つのサービスの URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/<open-api-application-path>/v1/{resourcePath:
[&#94;:]*}
```

説明

- `<open-api-application-path>` は、`OpenApiApplicationPath` アノテーションで記述されている一意のパスに対応します。
- `{resourcePath: [^:]*}` は、ドキュメントへの REST リソースのパスに対応します。

HTTP コード

HTTP コード	説明
200 (OK)	ドキュメントは正常に生成されました。
401 (Unauthorized)	認証に失敗しました。
403 (Forbidden)	指定されたリソースの読み取り権限は、現在のユーザーに対して拒否されました。
404 (Not found)	URL で指定されたリソースが見つかりません。

ドキュメントの強化

MicroProfile OpenAPI

OpenAPI ドキュメントを生成するためにデフォルトで処理される JAX-RS アノテーションに加え、TIBCO EBX® は [Microprofile OpenAPI アノテーション](#) をサポートしています。これらのアノテーションは、REST Toolkit フレームワークによってすでにカバーされている OpenAPI ドキュメントの部分を書き換える必要なく、記述を強化するために使用されます。アノテーションは、そのタイプに応じて、アプリケーションとリソースクラスの両方で使用することができます。

注意:

リソースクラスで明示的に定義されていない場合、`OpenAPIDefinition` アノテーションは、その REST Toolkit アプリケーションクラスから継承されます。このアノテーションは、OpenAPI ドキュメントのルートドキュメントオブジェクトを記述しています。

テーブルレコードのシリアル化

`ContentHolder` を DTO で使用して `テーブルレコード` をシリアル化する場合は、さらにアクションを実行する必要があります。

- フィールドに `Table` アノテーションが設定されている必要があります。
- `Schema` アノテーションは、処理のためにフィールドにタグ付けする必要があります。

注意:

`Schema` アノテーションの `ref` 値を空にすることはできず、アプリケーションスコープ全体で一意にする必要があります。

```
/**  
  
 * 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;
```

```
}
```

注意:

制限により、この場合、メタデータフィールドは追加されないことに注意してください。

権限

開発実行モードでは、OpenAPI エンドポイントへの匿名アクセスがすべてのユーザーに許可されます。実稼働および統合実行モードでは、OpenAPI アクセス権限は **基礎となるリソース** のものから継承されます。

以下も参照してください。:

- **EBX® 実行モード**

- リソースのドキュメントには、現在のユーザーがアクセスできる操作のみが含まれます。
- 現在のユーザーが操作にアクセスできない場合は、権限が拒否されたことが返されます。
- REST Toolkit アプリケーションのドキュメントには、許可されたリソースのスコープのみが含まれます。
- 現在のユーザーがどのリソースにもアクセスできない場合は、権限が拒否されたことが返されます。

ビジュアルドキュメント

開発実行モードでは、**Swagger UI** は タブの >> から利用できます。これにより、対応する REST Toolkit アプリケーションの文書化されたリソースを表示および操作できます。ビジュアルドキュメントにより、バックエンドの実装とクライアント側での消費が容易になります。

Swagger UI は、次の URL から直接利用することもできます。

アプリケーションの OpenAPI 視覚化の URL フォーマット

```
http[s]://<host>[:<port>]/<open-api-application-path>/v1/ui
```

単一サービスの OpenAPI 視覚化の URL フォーマット

```
http[s]://<host>[:<port>]/<open-api-application-path>/v1/ui/{resourcePath:
[&#94;;:]*}
```

説明

- `<open-api-application-path>` は、`OpenApiApplicationPath` アノテーションで記述されている一意のパスに対応します。
- `{resourcePath: [^;:]*}` は、ドキュメントへの REST リソースのパスに対応します。

注意:

リバースプロキシまたは HTTPS 構成を使用する場合、OpenAPI UI は外部リソースと見なされます。つまり、[外部リソース URL](#) を正しく構成する必要があります。

制限事項

OpenAPI	テーブルレコードのシリアル化では、メタデータフィールドは追加されません。 詳細については、テーブルレコードのシリアル化を参照してください。 リストまたはマップではない Java ジェネリックを使用する DTO クラスはサポートされていません。
---------	--

レコードの権限

- [はじめに](#)
- [字句構造](#)
 - [はじめに](#)

- 文字セット
- 文字の大文字と小文字の区別
- コメント
- キーワード
- 識別子
 - 引用符なしの識別子
 - 引用符付きの識別子
- 型
 - サポートされている型
 - リテラル
- フィールドアクセス
 - 単純なフィールド
 - 外部テーブルフィールド
 - リスト (複数値) フィールドアクセス
 - 関連付け
- 演算子
- Null 値の管理
 - 算術演算子
 - 比較演算子
 - ブール演算子
 - インデックス式
 - 関数
- If ステートメント
 - 「If then」 ステートメント
 - 「If then else」 ステートメント
 - 「Then」 または 「else」 の本文ステートメント
- return ステートメント

- コンテキスト変数
 - はじめに
 - データスペース
 - データセット
 - セッション
- コンテキスト関数
 - ロール
 - セッション
- その他の関数
 - 文字列照合関数
 - 集計関数
 - 日付と時刻の関数
 - その他の関数

はじめに

レコード権限 DSL (ドメイン固有言語) を使用して、特定のテーブルのレコードに対するアクセスルールを指定します。

レコード権限 DSL の主な目的は、Java コーディングを必要とせず、プログラミングの深い知識がない人でも簡単に使用できるようにすることです。

スクリプトを使用して、任意のテーブルに対する権限を指定できます。スクリプトは、レコードの権限を示す一連の **if then else** および **return** ステートメントで構成されます。

データモデルアシスタント (DMA) を使用してスクリプトを編集できます。

字句構造

はじめに

スクリプトの構造は次のとおりです。

```

begin

    <statement 1>

    <statement 2>

    ...

    <last statement>

end

```

最後のステートメントを除くすべてのステートメントは、「if then」または「if then else」ステートメントにする必要があります。

最後のステートメントは「if then」、「if then else」または「return」ステートメントです。

例

```

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;
```

文字セット

Unicode 文字セットがサポートされています。

文字の大文字と小文字の区別

DSL では大文字と小文字が区別されます。

コメント

1 行のコメントは、`//` から現在の行の終わりまで拡張されます。

```
// This is a comment

if record.LastName = 'Doe' then // This is another comment.

    return readOnly;
```

複数行のコメントは `/*` から始まり、`*/` で終わります。

```
/* This is a sample of a multi-line

    comment */

if record.isActive then

    return readWrite;
```

キーワード

キーワードには次の 2 種類があります。

- 予約済みキーワード は、`if`、`then`、`else`、`begin`、`end`、`return`、`null`、`and`、`or`、`not`、`true`、`false`、`\_ebx` です。

- 予約されていないキーワードは、DSL によって定義されている他のすべてのキーワードです。

2 種類のキーワードの主な違いは、予約されていないキーワードは、プレーンな (引用符で囲まれていない) 識別子として使用できることです。

識別子

引用符なしの識別子

引用符で囲まれていない識別子は、文字、数字、またはアンダースコア (`_`) の長さ無制限のシーケンスです。最初の文字は文字にする必要があります。

有効な文字は `a` から `z` および `A` から `Z` です。有効な数字は `0` から `9` です。

引用符で囲まれていない識別子は、予約済みのキーワードと同じにすることはできません。

引用符付きの識別子

引用符で囲まれた識別子は、Unicode 文字の長さに制限はありません。

引用符で囲まれた識別子は、二重引用符で囲んで使用する必要があります。

引用符で囲まれていない識別子は、二重引用符で囲んで使用できます。つまり、識別子 `"a_name"` は `a_name` と等しいということです。

引用符で囲まれた識別子は、`"_ebx"` を除く予約キーワードにすることができます。

型

サポートされている型

次の型がサポートされています。

タイプ	EBX® 対応型
Boolean	xs:boolean
10 進数	xs:decimal xs:int xs:integer
String	xs:string xs:anyURI xs:Name osd:text osd:html osd:email osd:password osd:color

タイプ	EBX® 対応型
	osd:resource osd:locale osd:dataspaceKey osd:datasetName
タイムスタンプ	xs:dateTime
Date	xs:date
時間	xs:time

リテラル

文字列リテラル

文字列リテラルは、一重引用符で囲まれた Unicode 文字の任意のシーケンスにすることができます。次の表は、エスケープシーケンスで置き換える必要のある文字を示しています。

文字	エスケープシーケンス
タブ	\t
バックスペース	\b
改行	\n
キャリッジリターン	\r
フォームフィード	\f
一重引用符	\'
バックスラッシュ	\\

文字は、Unicode 文字の 16 進コード XXXX を使用した \uXXXX 形式の Unicode エスケープシーケンスで指定できます。

例

値	構文
O'Harra	'O\''Harra'

値	構文
Noël	'No\u00EBI'
été	'\u00e9\u00E9'

注意

無効なエスケープまたは Unicode シーケンスは、コンパイル時にエラーを生成します。

10 進リテラル

次の 10 進形式がサポートされています。

形式	例
Integer	546 -67
浮動小数点数	54.987 -433.876 0.00054 -0.0032
指数表記	34.654e-5 -45E+65 1.543e23

タイムスタンプリテラル

タイムスタンプリテラルの形式は **dt(yyyy-MM-dd hh:mm:ss.sss)** です。

秒はオプションです。秒が指定されていない場合、0 が想定されます。秒は、ミリ秒の精度までの小数部を持つことができます。

グレゴリオ暦で無効な日付は、コンパイル時にエラーを生成します。

例

```
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)
```

日付リテラル

日付リテラルの形式は **d(yyyy-MM-dd)** です。

グレゴリオ暦で無効な日付は、コンパイル時にエラーを生成します。

例

```
d(2010-01-02)
```

```
d(2019-2-3)
```

```
dt(2019-5-7)
```

時間リテラル

時間リテラルの形式は **t(hh:mm:ss.sss)** です。

秒はオプションです。秒が指定されていない場合、0 が想定されます。秒は、ミリ秒の精度までの小数部を持つことができます。

無効な時間は、コンパイル時にエラーを生成します。

例

```
t(00:00:00)
```

```
t(12:56:7)
```

```
t(12:56:7.5)
```

```
t(1:6)
```

ブール値リテラル

ブール値リテラルは **true** と **false** です。

フィールドアクセス

単純なフィールド

最適化/インデックス付きテーブルフィールドへのアクセスのみがサポートされています。ドット表記は、テーブルフィールドにアクセスするために使用されます。たとえば、パスが **./OfficeAddress/City** である現在のテーブルフィールドの条件は次のようになります。

```
if record.OfficeAddress.City = 'Paris' then  
  
    return readWrite;
```

エイリアス **record** は、常に現在のレコードを参照します。コンテキストによっては、他のエイリアスを使用できる場合があります。

各ステップ (ドットで区切られた部分) は識別子です。これは、次の引用表記を任意のステップに使用できることを意味します。

```
if record."OfficeAddress".City = 'Paris' then  
  
    return readWrite;
```

これは、予約済みのキーワードに相当するステップや、マイナス文字 (-) やドット (.) などの文字を使用するステップに役立ちます。これらは、引用符で囲まれていない識別子と互換性がありません。

実行時に、どのステップも **null** と評価される可能性があります。この場合、完全なフィールド式は **null** と評価されます。

外部テーブルフィールド

ドット表記を使用して外部キーを「フォロー」することにより、外部テーブルにアクセスできます。

次の例では、フィールド **Supervisor** は外部キーです。

```
if record.Supervisor.Name = 'John Doe' then  
  
    return readWrite;
```

次の例のように、複数レベルの外部キーが存在する場合があります。

```
if record.Supervisor.Supervisor.Supervisor.Name = 'John Doe' then

    return readOnly;
```

リスト (複数值) フィールドアクセス

複数值フィールドはサポートされていません。

関連付け

集計関数は、関連付けに基づいて入力式として受け取ることができます。次の例では、フィールド **ManagedUsers** は関連付けです。

```
// All users that manage at least 2 persons have read write access.

if count(record.ManagedUsers[]) >= 2 then

    return readWrite;
```

関連付けにフィルターを適用できます。関連付けからフィールドにアクセスするには、エイリアスを使用する必要があります。関連エイリアスは **:** を使用して宣言されます。次の例では、エイリアスは **u1** です。

```
// 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;
```

関連付けのフィルターは、現在のレコードのフィールドを参照できます。

```
// 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.C
```

```
ity]) then

    return readOnly;
```

関連付けのフィルターには、埋め込み集計関数を含めることができます。

```
// 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;
```

埋め込み集計関数に対する関連付けのフィルターは、現在のレコードのフィールドおよび親関連付けのフィールドを参照できます。

```
// 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[exists(t1.ManagedUsers:t2[t2.OfficeAddress.Ci
ty=record.OfficeAddress.City and
t2.OfficeAddress.City=t1.OfficeAddress.City])]) then

    return readOnly;
```

演算子

デフォルトでは、操作の評価順序は優先度と結合性に基づいています。評価の順序は、括弧を使用して明示的に示すことができます。

次の表は、優先度の高いものから低いものへのすべての演算子とそれらの関連性を示しています。

優先度	演算子	オペランド型	結果型	結合性
8	(リストのエレメントへのアクセス).(フィー	リストインデックスは 10 進数	任意の型にすることができます。	左から右へ

優先度	演算子	オペランド型	結果型	結合性
	ルドへのアクセス) () (括弧)	に必要があります。		
7	not	ブール値	ブール値	
6	* /	10 進数	10 進数	左から右へ
5	+ -	10 進数	10 進数	左から右へ
4	< <= > >=	文字列、10 進数、タイムスタンプ、日付、時刻 (3)。	ブール値	連想的ではありません。
3	= <>	文字列、10 進数、タイムスタンプ、日付、時刻、ブール値 (3)。	ブール値	
2	and	ブール値	ブール値	左から右へ
1	or	ブール値	ブール値	左から右へ

Null 値の管理

算術演算子

算術演算子 (*、/、+ および -) は **null** を返します。オペランドは **null** です。

比較演算子

比較演算子 (<、<=、>、>=、= および <>) は、いずれかのオペランドが **null** の場合、**null** を返します。

ブール演算子

ブール演算子はスレッド値ロジックを使用します。

AND の真偽 (true/false) 表は次のとおりです。

AND	TRUE	FALSE	NULL
true	true	false	null
false	false	false	false
null	null	false	null

OR の真偽 (true/false) 表は次のとおりです。

OR	TRUE	FALSE	NULL
true	true	true	true
false	true	false	null
null	true	null	null

インデックス式

null と評価されるか範囲外のインデックスを持つインデックス付き式は、**null** を返します。

関数

パラメーターが **null** の場合、関数は通常 **null** を返します。例外は関数 **isNull(value)** で、**null** を返すことはありません。

If ステートメント

「If then」ステートメント

「if then」ステートメントの構文は次のとおりです。

```
if condition-expression then

    then-body-statement
```

条件式はブール型として評価される必要があります。

式が **true** と評価された場合、「then」ステートメントが実行されます。式が **false** または **null** と評価された場合、「then」ステートメントは無視されます。

「then」ステートメントは [body ステートメント](#) です。

「If then else」ステートメント

「if then else」ステートメントの構文は次のとおりです。

```
if condition-expression then  
  
    then-body-statement  
  
else  
  
    else-body-statement
```

条件式はブール型にする必要があります。

式が **true** と評価された場合、「then」ステートメントが実行されます。式が **false** または **null** と評価された場合、「else」ステートメントが実行されます。

「then」または「else」ステートメントは [body ステートメント](#) です。

注意

式

```
if condition-expression then  
  
    statement-a;  
  
else  
  
    statement-b;
```

上記の式は、以下の式とは同等ではない可能性があります。

```
if not condition-expression then  
  
    statement-b;
```

```
else

    statement-a;
```

実際、式が null の場合、どちらの場合も「else」ステートメントが実行されます。

「Then」または「else」の本文ステートメント

本文のステートメントは次のようになります。

- `return` ステートメント
- 「if then」または「if then else」ステートメント
- ステートメントブロック

ステートメントブロックは、複数のステートメントをグループ化するために使用されます。キーワード **begin** で始まり、キーワード **end** で終わります。

```
begin

    <statement 1>

    <statement 2>

    ...

    <last statement>

end
```

最後のステートメントを除くすべてのステートメントは、「if then」または「if then else」ステートメントにする必要があります。

最後のステートメントは「if then」、「if then else」または「return」ステートメントです。

```
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

```

return ステートメント

return ステートメントは、特定の条件を満たすレコードへのアクセスを指定します。

次の表に、有効な return ステートメントを示します。

RETURN ステートメント	説明
return hidden;	レコードは非表示になっています (ユーザーにはアクセス権がありません)。

RETURN ステートメント	説明
return readOnly;	レコードは、現在のユーザーに対してのみ読み取られます。
return readWrite;	レコードは、現在のユーザーが読み取って変更できます。

特定のレコードに適用される return ステートメントがない場合、値 **hidden** が想定されます。

コンテキスト変数

はじめに

レコードの権限は、現在のデータスペース、データセット、またはセッションによって異なります。

データスペース

事前定義されたエイリアス **dataspace** は、現在のデータスペースに関する情報へのアクセスを提供します。

このエイリアスは、次のフィールドへのアクセスを提供します。

名前	タイプ	説明
name	string	データスペースの名前。データスペースの名前空間とスナップショットの名前空間は独立しているため、返される文字列は名前空間の 1 つのコンテキストにおける識別子にすぎません。グローバルデータスペースまたはスナップショット識別子には、フィールド id を使用します。
id	string	データスペースまたはスナップショットの永続的な識別子。 name と比較すると、この識別子は、この ID がデー

名前	タイプ	説明
		タスペース用かスナップショット用かをさらに指定します。
isSnapshot	boolean	データスペースがスナップショットの場合は true であり、データスペースがブランチの場合は false です。

例

```
if dataspace.name = 'branch-R' then

    return readOnly;
```

データセット

事前定義されたエイリアス **dataset** は、現在のデータセットに関する情報へのアクセスを提供します。

このエイリアスには次のフィールドがあります。

名前	タイプ	説明
name	string	データセットの名前

例

```
if dataset.name = 'TEST' then

    return readWrite;
```

セッション

事前定義されたエイリアス **session** は、現在のユーザーセッションに関する情報へのアクセスを提供します。

このエイリアスには次のフィールドがあります。

名前	タイプ	説明
<code>userId</code>	<code>string</code>	現在のユーザーの ID
<code>userEmail</code>	<code>string</code>	現在のユーザーのメール
<code>trackingInfo</code>	<code>string</code>	現在のセッションの追跡情報。

現在のユーザーのロールを確認するには、[ロール](#)を参照してください。

例

```
if session.userId = 'jdoe' then

    return readWrite;
```

コンテキスト関数

ロール

次の表では、`isMember()` 関数について説明します。

構文	説明
<code>isMember(ロール式...)</code>	現在のユーザーが指定された式から評価されたロールの少なくとも 1 つを持っている場合、<code>true</code> を返します。 ロール式は、ロールリテラルまたはフィールドパスにすることができます。ロールリテラルに関しては、次のビルトインロールは引用符なしで指定する必要があります。administrator、readOnly、everyone。カスタムロールは文字列リテラルとして指定されます。存在しないロールを指定してもエラーにはなりません (メンバーのいないロールと見なされます)。 注意 ロール administrator と 'administrator' は 2 つの異なるロールにすることができます。これは、<code>isMember(administrator)</code> が <code>isMember('administrator')</code> と同じ結果を返さ

構文	説明
	ない可能性があることを意味します。同じルールが他のビルトインロールにも適用されます。

例

```
// ロールリテラル:

if isMember('SALES', 'SUPPORT') then

    return readWrite;

// ロール管理者（引用符なし）:

if isMember(administrator) then

    return readOnly;

// フィールドパス:

if isMember(record.AuthorizedRole) then

    return readWrite;

// 異なるロール式を同時に使用できます:

if isMember('SALES', 'SUPPORT', administrator, record.AuthorizedRole) then

    return readWrite;
```

テーブルで構成されたロールに権限を依存させるには、1 つまたは複数の関連付けによってレコードに関連付ける必要があります。次に、関数 **isMember** を次のように使用する必要があります。

```
// 現在のユーザーに承認されているゾーンに割り当てられているすべてのレコード

// (つまり、ユーザーのロールの少なくとも 1 つが Roles.AuthorizedRole フィールドで構成されている場合)。

if exists(record.Zones:Z[exists(Z.Roles:R[isMember(R.AuthorizedRole)])])
then

    return readOnly;
```

セッション

次の表では、セッション関数について説明します。

構文	説明
getSessionInputParameter(parameterKey, lookupInParentSessions)	指定された parameterKey に対応するセッション入力パラメーターを返します。ワークフローインタラクションのコンテキストでは、関数はワークフロータスクの入力パラメーターでパラメーターを検索します。 lookupInParentSessions が true の場合、関数は親セッションでパラメーターを検索します。それ以外の場合は、現在のセッションでのみパラメーターを検索します。
isInWorkflowInteraction(lookupInParentSessions)	現在のセッションがワークフローインタラクションコンテキスト内にある場合は、 true を返します。 lookupInParentSessions が true の場合、親セッションがワークフローインタラクションコンテキスト内にある場合、この関数は true を返します。それ以外の場合、関数は現在のセッションのコンテキストのみを評価します。

例

```
// 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;

```

その他の関数

文字列照合関数

文字列照合関数は、次の 3 つのパラメーターを取ります。

- 文字列として評価される **stringExpression**。
- 文字列リテラルの必要がある **pattern**。
- 照合で大文字と小文字が区別されるかどうかを示すオプションのブールリテラル **isCaseSensitive**。省略した場合、照合では大文字と小文字が区別されないと見なされます。

構文	説明
<code>matches(stringExpression, pattern, isCaseSensitive)</code>	stringExpression が Java 正規表現 pattern と一致する場合、 true を返します。
<code>startsWith(stringExpression, pattern, isCaseSensitive)</code>	stringExpression が文字列 pattern で始まる場合、 true を返します。
<code>endsWith(stringExpression, pattern, isCaseSensitive)</code>	stringExpression が文字列 pattern で終わる場合、 true を返します。
<code>contains(stringExpression, pattern, isCaseSensitive)</code>	stringExpression に文字列 pattern が含まれている場合、 true を返します。
<code>containsWholeWord(stringExpression, pattern, isCaseSensitive)</code>	stringExpression に単語全体 pattern が含まれている場合、 true を返します。

例

```

// 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;
```

集計関数

次の集計関数は、関連付けで使用できます。

構文	説明
count(expression)	式の行数を返します。
exists(expression)	式が少なくとも 1 行と評価された場合、true を返します。

例

```
// 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;
```

日付と時刻の関数

次の表に、日付と時刻の関数を示します。

構文	説明
<code>datetimeNow()</code>	現在の日付と時刻を返します。
<code>dateNow()</code>	現在の日付を返します。
<code>timeNow()</code>	現在の時刻を返します。

例

```
// Give read write access if the last possible update date hasn't expired yet.

if record.LastPossibleUpdateDate >= dateNow() then

    return readWrite;

else

    return readOnly;
```

その他の関数

次の表は、ブール値を返すビルトイン関数についての説明です。

構文	説明
<code>isNull(value)</code>	値が null の場合は true を返します。値は、任意のタイプの式にすることができます。

例

```
// Give read write access if no supervisor.

if isNull(record.supervisor) then
```

```
return readWrite;
```

EBX® スクリプト

- リファレンス
- 使用方法
 - 関数フィールド
 - スクリプトタスク
 - テーブルトリガー
 - データセットトリガー
 - スクリプト化された REST サービス
 - REST サービス
 - ロールマッパー
 - 制約検証
- API
 - ユニットの概要
 - デフォルトのユニット
 - ユニット core.complex
 - ユニットコアデータ
 - ユニット core.date
 - ユニット core.datetime
 - ユニット core.dictionary
 - ユニット core.http
 - ユニット core.json
 - ユニット core.list
 - ユニット core.locale
 - ユニット core.log
 - ユニット core.math
 - ユニット core.message

- ユニット core.profile
- ユニット core.resource
- ユニット core.set
- ユニット core.string
- ユニット core.time
- ユニット core.uri
- ユニット core.validation
- ユニット core.rest

リファレンス

- はじめに
- 字句構造
 - はじめに
 - 文字セット
 - 文字の大文字と小文字の区別
 - コメント
 - キーワード
- 識別子
 - 引用符なしの識別子
 - 引用符付きの識別子
- 参照
 - データモデルへの参照
- 型
 - 定義済みタイプ
 - 複合型
 - リスト型
 - typeof キーワード
 - データモデルで宣言された型
 - レコード、テーブル、主キー

- 型修飾子
- キーワード raise
- リテラル
 - 文字列リテラル
 - 10 進リテラル
 - 整数 (32 ビット) リテラル
 - タイムスタンプリテラル
 - 日付リテラル
 - 時間リテラル
 - ブール値リテラル
 - Null リテラル
- 演算子
 - 優先度
 - 算術演算子
 - 文字列連結演算子
 - ブール演算子
 - 比較演算子
 - ビルトインおよびユニット関数
- 割り当て
- ステートメントブロック
- 変数と定数
 - はじめに
 - 割り当てステートメント
 - 型検出を使用した宣言
 - 明示的な型の定義
 - 参照変数 (ref)
 - スコープ
 - 複合型変数

- 複数値フィールド
- 定義済み変数
 - はじめに
 - データスペース
 - データセット
 - ルート
- 関数とプロシージャ
 - 関数
 - プロシージャ
- エクスポート済み関数またはプロシージャ
 - パラメーター
- If ステートメント
 - 「If then」 ステートメント
 - 「If then else」 ステートメント
 - If ステートメントと三値論理
- ループ
 - 「For in do」 ループ
 - 「While do」 ループ
- ユニット
- SQL クエリ
 - 埋め込みクエリ
 - 動的クエリ
 - クエリセッション
- ログ取得とデバッグ
- 初期スクリプト

はじめに

EBX® スクリプトは、EBX カスタマイズ (計算値、テーブルトリガー、スクリプトタスクなど) など、現在 Java が必要なあらゆる場所で使用できるように設計された、シンプルで非常に強力な言語です。

これは DSL (ドメイン固有言語) に基づいています。DSL は手続き型で、強く静的に型付けされた言語です。

スクリプトを記述するには、EBX IDE を使用できます。これは、関数フィールド、テーブルトリガー、スクリプトタスクのスクリプトの定義を支援するツールと機能を提供する統合開発環境です。構文の強調表示、コンテキストコードの補完、ソースのコンパイルと公開などのさまざまな機能を備えたエディターが含まれています。

DMA を使用して新しい機能フィールドまたはテーブルトリガーを作成すると、EBX® Script IDE にデフォルトのスクリプトが作成されます。

EBX® Script IDE を使用してスクリプト タスクを作成すると、[ワークフローモデル](#) で指定できます。

字句構造

はじめに

スクリプトの構造は次のとおりです。

```
<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>

```

詳細については、[データモデルで宣言された型](#)、[ユニット](#)、および [関数とプロシージャ](#) を参照してください。

次の例は、関数フィールドのスクリプトです。

```

// 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

```

次の例は、テーブルトリガーのスクリプトです。

```

//This trigger adds the full name of current record.

export procedure onBeforeCreate()

begin

    _ebx.record.fullName := _ebx.record.firstName | ' ' |
    _ebx.record.lastName;

end

```

文字セット

Unicode 文字セットがサポートされています。

文字の大文字と小文字の区別

DSL では大文字と小文字が区別されます。

コメント

1 行のコメントは、`//` から現在の行の終わりまで拡張されます。

```

// This is a comment

if _ebx.record.LastName = 'Doe' then // This is another comment.

    return true;

```

複数行のコメントは `/*` から始まり、`*/` で終わります。

```

/* This is an example of a multi-line

    comment */

```

```
if _ebx.record.isActive then

    return false;
```

キーワード

予約済みキーワードは、**and**、**or**、**not**、**uses**、**as**、**frommodelexport**、**typeof**、**tableof**、**primarykeyof**、**mutable**、**immutable**、**modifiable****unmodifiable**、**persisted****function**、**procedure**、**const**、**var**、**if**、**then**、**else**、**for**、**while**、**in**、**do**、**begin**、**end**、**return**、**true**、**false**、**raise**、**null** です。

予約済みのキーワードは、プレーンな (引用符で囲まれていない) 識別子として使用することはできません。

識別子

引用符なしの識別子

引用符なしの識別子は、文字、数字、またはアンダースコア (_) の無制限の長さのシーケンスです。最初の文字は、文字またはアンダースコアにする必要があります。

有効な文字は **a** から **z** および **A** から **Z** です。有効な数字は **0** から **9** です。

引用符なしの識別子は、予約済みのキーワードと等しくない場合があります。

引用符付きの識別子

引用符付き識別子は、二重引用符 (")を除く任意の Unicode 文字の無制限の長さです。

引用符で囲まれた識別子は、二重引用符で囲んで使用する必要があります。

引用符なしの識別子は、二重引用符で囲んで使用できます。これは、識別子 "a_name" が a_name と等しいことを意味します。

引用符付きの識別子は予約済みのキーワードにすることができます。

参照

データモデルへの参照

スクリプトはデータモデルへの参照を宣言できます。

XSD ファイルとして展開されたデータモデルの場合、サポートされる構文は次のとおりです。

```
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;
```

短縮名は、パスと拡張子を除いたファイル名です。たとえば、**/WEB-INF/ebx/schemas/Directory.xsd** の短縮名は **Directory** です。

エイリアスが指定されていない場合、エイリアスは短い名前またはファイルパスと同じであると見なされます。

次のいずれかの構文を使用して、現在のモジュールからデータモデルを宣言できます。

```
references model short_name from module . as alias;

references model short_name from module .;

references model file_path from module . as alias;
```

埋め込みモデルの場合は、次のいずれかの構文を使用します。

```
references model short_name as alias;

references model short_name;
```

データモデル参照の例

```
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;

```

型

定義済みタイプ

次の定義済みタイプがサポートされています。

タイプ	キーワード	プロパティ	EBX® 対応型
Boolean	boolean		xs:boolean
Decimal (無制限の精度)	decimal		xs:decimal
Integer (32ビット)	int		xs:int xs:integer
String	string	length (0 以上の整数)	xs:string xs:Name osd:text osd:html osd:email osd:password 1 osd:color osd:dataspaceKey osd:datasetName
Timestamp (ミリ秒の精度でタイムゾーンなし)	timestamp	year month (1 から 12) day (1 から 31) hour (0 から 23) minute (0 から 59) second (0.000 から 59.999)	xs:dateTime

タイプ	キーワード	プロパティ	EBX® 対応型
Date (タイムゾーンなし)	date	year month (1 から 12) day (1 から 31)	xs:date
Time (ミリ秒の精度)	time	hour (0 から 23) minute (0 から 59) second (0.000 から 59.999)	xs:time
Locale	locale		osd:locale
URI (Uniform Resource Identifier)	uri		xs:anyURI
Resource	resource		osd:resource
Dataspace	dataspace	name identifier (先頭に「B」または「V」が付いた名前) isSnapshot (データスペースがスナップショットの場合は true)	
Dataset	dataset	name dataspace	
Transaction	transaction	dataspace isAllPrivileges (読み取り/書き込みブール型プロパティ) executionInformation (読み取り/書き込み文字列プロパティ) executionTimestamp executionUUIDString	

1 タイプ **osd:password** の関数フィールドを定義することはできません。

複合型

複合型は、EBX® スキーマで定義されたグループ (複合) ノードの型です。

詳細については、[複合型の変数](#)の章を参照してください。

リスト型

DSL はリストをサポートしています。次の構文を使用してリストを宣言します。

```
list<item_type>
```

リストはデフォルトでは変更できません。次の構文を使用して変更可能なリストを宣言できます。

```
modifiable list<item_type>
```

リストが変更不可の場合、項目を追加、削除、または置換することはできません。

インデックス付き表記を使用して、値を設定または取得します (最初のインデックスは 0)。

```
// Set first value of cities.  
  
cities[0] := 'Paris';  
  
// Get the second item of cities.  
  
return cities[1];
```

インデックスは、int 型または 10 進数の任意の式にすることができます。インデックスが 10 進数の場合、小数部分は無視されます。

インデックスが null、負の値、または範囲外の場合、get 式は null を返します。

インデックスが null、負の値、または範囲外の場合、set 式は実行時にエラーを報告します。

プロパティ **size** は、リストのサイズを返します。

```
var size := cities.size;
```

リストを反復処理できます。詳細については、[for ループ](#)を参照してください。

複数値スキーマフィールド

複数値のスキーマフィールドは、リスト型と見なされます。

複数値スキーマフィールドは、値の最大数 (maxOccurs) が 1 より大きいか、「unbounded」に設定されているフィールドです。

インデックス付き表記は、値にアクセスまたは設定するために使用されます。最初のインデックスは 0 です。次の例では、レコードフィールド「OfficeAddress/street」は複数値です。

```
// Return the second line of the street part of the address.

return OfficeAddress.street[1].
```

typeof キーワードは、多値フィールドで使用できます ([typeof キーワード](#)の章を参照)。

typeof キーワード

フィールドまたは変数に応じて型を指定するには、キーワード **typeof** を使用します。

たとえば、レコードフィールド **OfficeAddress** の型を参照するには、次の構文を使用します。

```
typeof record.OfficeAddress
```

フィールドが複数値の場合、リストと見なされます。次の構文を使用して、リストの項目の型を参照できます。

```
typeof record.Addresses[*]
```

データモデルで宣言された型

パス **/root/User/Address** を持つ **current** データモデル内のフィールドの型を参照するには、次の構文を使用します。

```
var address: .User.Address;
```

「/」が「.」に置き換えられ、最初のステップが省略されていることに注意してください。

エイリアス 'hist' で宣言されたデータモデル内のパス **/root/Sales/Amount** を持つデータモデルからフィールドの型を取得するには、次の構文を使用します。

```
var amount: hist.Sales.Amount
```

フィールドが複数値の場合、リストと見なされます。次の構文を使用して、リストの項目の型を参照できます。

```
var products: hist.Sales.products[*]
```

レコード、テーブル、主キー

データモデル **sample** のパス "/root/test/table" がモデル用である場合、レコードを指定するには次のいずれかの構文を使用します。

```
var immutableRecord1: sample.test.table;

var immutableRecord2: immutable sample.test.table

var mutableRecord: mutable sample.test.table
```

修飾子が指定されていない場合、型は不変です。

テーブルを指定するには、次の構文を使用します。

```
var aTable: tableof sample.test.table;

aTable := db.findTable<.directory.Customer>(_ebx.dataset);
```

主キーを指定するには、次のいずれかの構文を使用します。

```
var immutablePrimaryKey1: primarykeyof sample.test.table;

var immutablePrimaryKey: immutable primarykeyof sample.test.table

var mutablePrimaryKey: mutable primarykeyof sample.test.table
```

修飾子が指定されていない場合、主キーは不変です。

型修飾子

型は次のようになります。

- **immutable**: この型の項目を変更することはできません。
- **mutable**: この型の項目は、変更できる可能性があります。

単純型は常に不変です。

互換性のある不変型の変数または戻り値に変更可能な値を割り当てることはできますが、変更可能な変数または戻り値に変更不可能な値を割り当てると、コンパイル時にエラーが発生します。

次のいずれかの構文を使用して不変型を指定できます。

```
immutable typeof record.OfficeAddress

immutable .User.Address

immutable hist.Sales.Amount
```

次のいずれかの構文を使用して、変更可能な型を指定できます。

```
mutable typeof record.OfficeAddress

mutable .User.Address

mutable hist.Sales.Amount
```

一部の型では **immutable** または **mutable** キーワードがサポートされておらず、コンパイル時にエラーが発生する場合があります。

リストは次のようになります。

- **unmodifiable**: 挿入時に項目を削除することはできません。項目は不変の場合もそうでない場合もあります。
- **modifiable**: 項目を挿入時に削除できます。項目は不変の場合もそうでない場合もあります。

変更可能なリストを、互換性のある変更不可能なリスト型の変数または戻り値に割り当てることはできますが、変更不可能なリストを変更可能な変数または戻りリストに割り当てると、コンパイル時にエラーが発生します。

可変複合型またはリストオブジェクトを変更できますが、オブジェクトの一部の属性の変更を妨げる可能性のある他の制約が適用される場合があります。

次の構文を使用して、変更不可能なリストタイプを指定できます。

```
unmodifiable typeof _ebx.record.Addresses[*]

unmodifiable .User.children[*]

unmodifiable hist.Sales.products[*]
```

次の構文を使用して、変更可能なリストタイプを指定できます。

```
modifiable typeof _ebx.record.Addresses[*]

modifiable .User.children[*]

modifiable hist.Sales.products[*]
```

キーワード **raise**

トリガーまたは関数フィールドの実行中にエラーを報告するには、キーワード **raise** を使用します。

メッセージなしでエラーを発生させるには、次の構文を使用します。

```
raise code;
```

メッセージとともにエラーを発生させるには、次の構文を使用します。

```
raise code, message;
```

- `code` はエラー識別子です。32 の整数値にする必要があります。
- `message` は表示されるエラーメッセージを指定します。これはオプションの文字列値です。

関数フィールドでキーワード **raise** を使用する例

```
// 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

```

テーブルトリガーでキーワード **raise** を使用する例

```

// 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

```

リテラル

文字列リテラル

文字列リテラルは、一重引用符で囲まれた Unicode 文字の任意のシーケンスにすることができます。次の表は、エスケープシーケンスで置き換える必要のある文字を示しています。

文字	エスケープシーケンス
タブ	\t

文字	エスケープシーケンス
バックスペース	\b
改行	\n
キャリッジリターン	\r
フォームフィード	\f
一重引用符	\'
バックスラッシュ	\\

\uXXXX 形式の Unicode エスケープシーケンスを使用して文字を指定します。XXXX は Unicode 文字の 16 進数コードです。

例

値	構文
O'Harra	'O\'Harra'
Noël	'No\u00EBI'
été	'\u00e9t\u00E9'

注意

無効なエスケープまたは Unicode シーケンスは、コンパイル時にエラーを生成します。

10 進リテラル

次の 10 進形式がサポートされています。

形式	例
小数部なし	546 -67
浮動小数点数	54.987 -433.876 0.00054 -0.0032
指数表記	34.654e-5 -45E+65 1.543e23

整数 (32ビット) リテラル

整数 32 ビットリテラルの形式は、**90324i** または **-543i** のようになります。

値は **-2147483648i** から **2147483647i** までの範囲でなければなりません。

注意：i 文字がない場合、値は小数と見なされます。

タイムスタンプリテラル

タイムスタンプリテラルの形式は **dt(yyyy-MM-dd hh:mm:ss.sss)** です。

秒はオプションです。指定されていない場合は 0 と見なされます。秒は、ミリ秒の精度までの小数部を持つことができます。

グレゴリオ暦で無効な日付は、コンパイル時にエラーを生成します。

例

```
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)
```

日付リテラル

日付リテラルの形式は **d(yyyy-MM-dd)** です。

グレゴリオ暦で無効な日付は、コンパイル時にエラーを生成します。

例

```
d(2010-01-02)
```

```
d(2019-2-3)
```

```
dt(2019-5-7)
```

時間リテラル

時間リテラルの形式は **t(hh:mm:ss.sss)** です。

秒はオプションです。指定されていない場合は 0 と見なされます。秒は、ミリ秒の精度までの小数部を持つことができます。

無効な時間は、コンパイル時にエラーを生成します。

例

```
t(00:00:00)
```

```
t(12:56:7)
```

```
t(12:56:7.5)
```

```
t(1:6)
```

ブール値リテラル

ブール値リテラルは、キーワード **true** またはキーワード **false** のいずれかです。

Null リテラル

null リテラルはキーワード **null** です。

このリテラルは、次の状況でのみ使用してください。

- 変数またはフィールドを設定する
- 戻り値

注意

このキーワードを使用して、変数またはフィールドが **null** かどうかをテストすることはできません。代わりに、関数 **isNull()** を使用してください。

演算子

優先度

デフォルトでは、操作の評価順序は優先度と結合性に基づいています。括弧を使用して、評価の順序を明示的に示します。

次の表は、優先度の高いものから低いものへのすべての演算子と、それらの関連性を示しています。

優先度	演算子	オペランド型	結果型	結合性
9	(リストのエレメントへのアクセス) . (フィールドへのアクセス) () (括弧)	リストインデックスは int または 10 進数でなければなりません。	任意の型にすることができます。	左から右へ
8	not	boolean	boolean	
7	* /	小数または整数	decimal	左から右へ
6	+ -	decimal または int	decimal	左から右へ
5		(文字列の連結)	string	string
4	< <= > >=	文字列、小数点、整数、タイムスタンプ、日付、時刻 (3)。	boolean	連想的ではありません。
3	= <>	string、decimal、int、timestamp、date、time、boolean (3)	boolean	
2	and	boolean	boolean	左から右へ
1	or	boolean	boolean	左から右へ

算術演算子

すべての算術演算子 (*、/、+、-) は、34 桁の精度 (IEEE 754R Decimal128 形式) の **decimal** で評価されます。int 型の値は、比較の前に常に **decimal** に変換されます。

いずれかのオペランドが **null** の場合、結果は **null** になります。

int 型変数に割り当てられると、**decimal** 値は自動的に変換されます。値が整数ではない、2147483647 より大きい、または -2147483648 より小さいために値を変換できない場合、エラーが発生します。

文字列連結演算子

文字列連結演算子 (||) オペランドは任意の型にすることができ、連結前に自動的に文字列に変換されます。**null** 値は空の文字列に置き換えられます。

文字列連結演算子 (||) は、連結を実行する前に、すべての **null** オペランドを空の文字列に置き換えます。

ブール演算子

ブール演算子は、三値論理 (ternary logic) と呼ばれる **三値論理** を使用します。この論理は SQL でも使用されます。

ブール変数は、**true**、**false**、**null** の 3 つの真値のいずれかを持つことができます。

二値論理と同様に、三値論理には **and**、**or**、**not** などの演算子があります。ただし、これらの演算子の動作は、**null** 値を次のように処理するように拡張されています。

- **not**: **true** を **false** に、**false** を **true** に反転しますが、**null** は変更しません。
- **and**: すべてのオペランドが **true** の場合、結果は **true** です。いずれかのオペランドが **false** の場合、結果は **false** です。少なくとも 1 つのオペランドが **null** で、かつ **false** のオペランドがない場合、結果は **null** です。これは、1 つ以上の **false** オペランドがすべての **null** オペランドを上書きすることを意味します。
- **or**: いずれかのオペランドが **true** の場合、結果は **true** です。すべてのオペランドが **false** の場合、結果は **false** です。少なくとも 1 つのオペランドが **null** で、かつ **true** のオペランドがない場合、結果は **null** です。これは、1 つ以上の **true** オペランドがすべての **null** オペランドを上書きすることを意味します。

比較演算子

比較演算子 (<、<=、>、=>、= および <>) は、同じ型の 2 つのオペランドを比較します。**int** 型の値は、比較の前に常に **decimal** に変換されます。

コンパレータの戻り値は **boolean** 型です。この値は **true** または **false** のみで、すべてのオペランドが **null** でない場合に限り、オペランドのいずれかが **null** の場合、それは **null** です。

ビルトインおよびユニット関数

ビルトイン関数とユニット関数は通常、パラメーターが **null** の場合、**null** を返します。

例外もあるため、必ず [API ドキュメント](#) を参照してください。

割り当て

次の表に、すべての割り当て演算子を示します。

演算子	オペランド型	説明
<code>:=</code>	あらゆる型に使用できます。	標準割り当て。
	<code>=</code>	string
<code>+=</code>	decimal、int	<code>a += b</code> は <code>a := a + b</code> と同等
<code>-=</code>	decimal、int	<code>a -= b</code> は <code>a := a - b</code> と同等
<code>*=</code>	decimal、int	<code>a *= b</code> は <code>a := a * b</code> と同等
<code>/=</code>	decimal、int	<code>a /= b</code> は <code>a := a / b</code> と同等

ステートメントブロック

ステートメント ブロックは、複数のステートメントをグループ化するために使用されます。キーワード **begin** で始まり、キーワード **end** で終わります。ほとんどのステートメントは ; で終了する必要があります。

```
begin

    statement_1;

    statement_2;

end
```

ステートメントブロックには、ステートメントブロックを含めることもできる 1 つ以上のステートメントブロックを含めることができます。

```
begin

    statement_1;

    begin

        statement_2_1;
```

```

    statement_2_3;

begin

    statement_3_1;

    statement_3_2;

end

end

statement_3;

end

```

変数と定数

はじめに

ブロック内の任意のステートメントは、変数または定数を宣言できます。

変数または定数の型は常に、変数が宣言されたときに固定されます。

代入演算子 `:=` を使用して、値を変数に割り当てることができます。

間違った型の値を変数に割り当てたり、宣言後に定数の値を変更したりすると、エラーになります。

割り当てステートメント

型検出を使用した宣言

変数または定数型は、宣言時に初期化すると自動的に検出できます。この機能は型推論と呼ばれます。

変数の構文は次のとおりです。

```
var variable_name := a_value;
```

定数の構文は次のとおりです。

```
const const_name := a_value;
```

例

```
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
```

注意

定数は必ずしも不変ではありません。

明示的な型の定義

変数の型を明示的に指定することができます。

変数の構文は次のとおりです。

```
// 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;
```

定数の構文は次のとおりです。

```
const const_name : variable_1_type := a_value;
```

例

```
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
```

初期化されていない変数の値は null です。

```
begin

    var firstName : string;
```



```

    if isNull(firstName) then

        doSomething(); // Statement will be executed.

    end

```

次のステートメントにはエラーがあります。

```

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

```

参照変数 (ref)

参照変数は、コンポーネント型の値への可変参照を格納します。これらはボックス化された値のように動作します。

- 読み取りは参照値に自動的に逆参照されます。
- 代入は参照値に書き込みます。

構文

ref 宣言は、形状において **var** を反映しています。初期化子が提供されている場合、型はオプションです。

```
// 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;
```

型が指定されていない場合、null 以外の型の初期化が必要です。そうでない場合、コンパイル時エラーが報告されます (**var** と同じルール)。

使用法

ref 変数への読み取りと書き込みは、参照値を直接ターゲットとします。**.value** フィールドにアクセスする必要はありません。

```
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
```

リストとインデックスで **ref** を使用することもできます。代入は通常のリスト要素の格納のように動作します。

```
uses core.list as list;

begin

  ref numbers: modifiable list<int> := list.of<int>();

  list.add(numbers, 1i);

  list.add(numbers, 2i);

end
```

注意: **ref** という単語は、グローバルに予約されたキーワードではありません (下位互換性のため)。**ref** を引用符なしで識別子として使用できます。宣言コンテキストで、**ref** がステートメントの先頭に現れる場合、それは参照変数宣言として解析されます。その位置で文字通り **ref** という名前の変数が必要な場合は、引用符付き識別子 (例: "**ref**") を使用してください。

トランザクションでの ref の使用

ref は、**execute transaction** ブロック内で計算された値をキャプチャし、後でそれを返すのに特に役立ちます。

```
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

```

スコープ

変数または定数範囲は、その宣言から、宣言されたステートメントブロックの終わりまでに及びます。

```

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
```

同じ名前の変数を同じブロックまたはサブブロックに存在させることはできません。

```
begin

  var firstName := 'John';

  begin

    var firstName := 'Bob'; // Error! Variable already declared.

    ....

  end

end
```

以下は正しい記述です。

```
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
```

複合型変数

複合型の変数または定数は、型検出または **typeof** キーワード を使用して宣言できます。

ドット表記は、複合型の変数のフィールドにアクセスするために使用されます。

例

```
// Declaration using type detection.

var address1 := _ebx.record.OfficeAddress;

// Declaration using typeof notation.

var address2 : typeof _ebx.record.OfficeAddress;

address2 := address1;
```

各ステップ (ドットで区切られた部分) は **識別子** です。これは、次の引用表記をどのステップにも使用できることを意味します。

```
var city := _ebx.record."OfficeAddress".City;
```

これは、予約済みのキーワードに相当するステップや、マイナス文字 (-) やドット (.) などの引用符で囲まれていない識別子と互換性のない文字を使用するステップに役立ちます。

実行時に、どのステップも **null** と評価される可能性があります。この場合、完全なフィールド式は **null** と評価されます。

複数値フィールド

複数値フィールドは、順序付きリストとして扱われます。インデックス表記は、リストの項目にアクセスするために使用されます。インデックスは 0 ベースです (最初のインデックスは 0)。

次の例では、フィールド「Address」は複数値です。

```
var city := _ebx.record.Address[1].City;
```

インデックスが範囲外の場合、インデックス付きの式は **null** を返します。この場合、完全なフィールド式は **null** と評価されます。

インデックスは **decimal** または **int expression** になります。**decimal`** の場合、小数部分は無視されます。

定義済み変数

はじめに

定義済み変数を使用すると、コンテキストの情報にアクセスできます。このセクションで説明する次の定義済み変数は、コンテキスト (関数フィールド、テーブルトリガー、データセットトリガー) に共通です。これらは定数であり、不変の複合型です。

各コンテキスト (関数フィールド、テーブルトリガー、スクリプト タスク) には、特定の定義済み変数がいくつかあることに注意してください。

これらの特定の変数の詳細については、[関数フィールド](#)、[テーブルトリガー](#)、および [スクリプト タスク](#) を参照してください。

データスペース

定義済み変数 `\_ebx.dataspace` は、現在のデータスペースに関する情報へのアクセスを提供します。

この変数には次のフィールドがあります。

名前	タイプ	説明
name	string	データスペースの名前。データスペースの名前空間とスナップショットの名前空間は独立しているため、返される文字列は名前空間の 1 つのコンテキストにおける識別子にすぎません。グローバル データスペースまたはスナップショット識別子には、フィールド id を使用します。
identifier	string	データスペースまたはスナップショットの永続的な識別子。 name と比較すると、この識別子は、この ID がデータスペース用かスナップ

名前	タイプ	説明
		ショット用かをさらに指定します。
isSnapshot	boolean	データスペースがスナップショットの場合は true であり、データスペースがブランチの場合は false です。

例

```
// 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
end
```

データセット

定義済み変数 `\_ebx.dataset` は、現在のデータセットに関する情報へのアクセスを提供します。

この変数には次のフィールドがあります。

名前	タイプ	説明
name	string	データセットの名前
dataspace	dataspace	データセットのデータスペース

例


```
// Returns the current dataset name.

export function getValue(): string

begin

    return _ebx.dataset.name;

end
```

注意: `\_ebx.dataset` 定義済み変数は、トリガーが **afterDelete** タイプの場合、データセットトリガーのコンテキストでは使用できません。

ルート

定義済み変数 `\_ebx.root` は、現在のスクリプトがテーブルトリガー、データセットトリガー、または関数フィールド用の場合に使用できます。この変数は、現在のデータセットのフィールドへの読み取り専用アクセスを提供します。

そのフィールドは、現在の EBX® スキーマによって定義されています。

例

```
// 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

```

関数とプロシージャ

関数

関数は、値を返すメソッドです。

パラメーターのない関数の構文は次のとおりです。

```

function function_name(): return_value_type

begin

    ....

    return a_value;

end

```

パラメーターを使用する関数の構文は次のとおりです。

```

function function_name(

    parameter_1 : parameter_1_type,

```

```

    parameter_2 : parameter_2_type...,): return_value_type

begin

    ....

    return a_value;

end

```

const キーワードを使用してパラメーター定数を宣言することができます。

```

function function_name(const parameter_1 : parameter_1_type):
return_value_type

begin

    ....

    return a_value;

end

```

関数の最後のステートメントは常に return ステートメントにする必要があります。この関数には、必要な数の return ステートメントを含めることができます。

例

```

function getFullName(firstName: string, lastName: string): string

begin

    if isNull(firstName) then

        return lastName;

    return firstName | ' ' | lastName;

end

```

```
end
```

宣言された値とは異なる値を返すことは違法です。

```
function getValue(): string

begin

    return 0; // Error! Return type is decimal, not string.

end
```

プロシージャ

プロシージャは、値を返さないメソッドです。

パラメーターのないプロシージャの構文は次のとおりです。

```
procedure procedure_name()

begin

    ....

end
```

パラメーターを使用するプロシージャの構文は次のとおりです。

```
procedure procedure_name(

    parameter_1 : parameter_1_type,

    parameter_2 : parameter_2_type...)

begin

    ....

end
```

const キーワードを使用してパラメーター定数を宣言することができます。

```
function procedure_name(const parameter_1 : parameter_1_type):
    return_value_type

begin

    ....

    return a_value;

end
```

プロシージャに `return` ステートメントを含めることはできませんが、関数には必要な数の `return` ステートメントを含めることができます。プロシージャステートメントは値を返すことができません。

エクスポート済み関数またはプロシージャ

エクスポートされた関数は、EBX® によって直接呼び出すことができます。REST サービスおよび制約検証用のスクリプトには、複数のエクスポートされた関数を含めることができます。その他のすべてのスクリプトタイプでは、スクリプトごとに 1 つのエクスポートされた関数またはプロシージャのみが許可されます。そのシグネチャ (名前、戻り値の型) はコンテキストによって異なります。

各コンテキストのエクスポートされた関数またはプロシージャの詳細については、[関数フィールド](#) と [テーブルトリガー](#) を参照してください。

パラメーター

単純型パラメーターは値によって渡されます。これは、関数またはプロシージャが元の値のコピーを受け取ることを意味します。

複合型とリスト型は参照によって渡されます。これは、関数またはプロシージャが元のオブジェクトを受け取ることを意味します。関数またはプロシージャがオブジェクトを変更すると、元のオブジェクトが変更されます。

If ステートメント

「If then」 ステートメント

「if then」ステートメントの構文は次のとおりです。

```
if condition-expression then

    then-statement
```

条件式はブール型として評価される必要があります。

then ステートメントは [ステートメントブロック](#)にすることができます。これは、条件式が **true** の場合にのみ実行されます。

「If then else」ステートメント

「if then else」ステートメントの構文は次のとおりです。

```
if condition-expression then

    then-statement

else

    else-statement
```

条件式はブール型にする必要があります。

「then」または「else」ステートメントは、[ステートメントブロック](#)にすることができます。**then** ステートメントは、条件式が **true** の場合にのみ実行されます。**else** ステートメントは、条件式が **false** または **null** の場合にのみ実行されます。

If ステートメントと三値論理

次のコード:

```
if condition-expression then

    statements-a;

else

    statements-b;
```

次のコードとは同等ではありません。

```

if not condition-expression then

    statements-b;

else

    statements-a;

```

実際、式が **null** の場合、どちらの場合も **else** ステートメントが実行されます。

次の例では、a または b のいずれかが **null** の場合、**reportError()** ステートメントは**実行されません**。

```

if a <> b then

begin

    reportError();

    return;

end

doSomething();

return;

```

これは意図した動作ではない可能性があります。代わりに、次のコードを使用できます。

```

if a = b then

    doSomething();

else

    reportError();

return;

```

または

```
if isTrueOrNull(a <> b) then  
  
begin  
  
    reportError();  
  
    return;  
  
end  
  
doSomething();  
  
return;
```

上記のコード例では、**null** 値を持つことは常にエラーであると見なしています。**a** と **b** の両方が **null** であることがエラーではない場合、次のようなコードを使用できます。

```
if (isNull(a) and isNull(b)) or (a = b) then  
  
    doSomething();  
  
else  
  
    reportError();  
  
end
```

ループ

「For in do」ループ

「for in do」ループステートメントは、リストの各項目を選択し、ブロックステートメントを実行するために使用されます。構文は次のとおりです。

```
for item_variable_name in list do  
  
begin  
  
    statement_a;  
  
end
```



```

    statement_b;

    ...

end

```

ブロックステートメントは、リストの値ごとに 1 回実行されます。各反復で、読み取り専用アイテム変数は、リストの順序でリストの値を取ります。

item 変数の名前は、現在の範囲内で一意にする必要があります。一意でない場合、コンパイル時にエラーが生成されます。

項目ごとに 1 つのステートメントを実行する必要がある場合は、次のより単純な構文を使用できます。

```
for item_variable_name in list do statement;
```

次の例は、複合体のリストを繰り返します。

```

// 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」 ループ

「while do」 ループステートメントは、条件が **true** になるまでステートメントブロックを実行するために使用されます。次の構文があります。

```

while condition do

begin

    statement_a;

    statement_b;

    ...

end

```

単一のステートメントを実行する必要がある場合は、より単純な、次の構文を使用できます。

```
while condition do statement_a;
```

次の例では、値の階乗を計算します。

```

function factorial(value : decimal): decimal

begin

    var factorial := value;

    while(value > 1) do

        begin

            value -= 1;

            factorial *= value;

        end

    return factorial;

end

```

範囲外のエラーを回避するために、**int** 型の代わりに **decimal** 型が使用されます。

ユニット

EBX® は、「ユニット」にパッケージ化された API を提供します。

ユニットは、複数の機能またはプロシージャを定義できます。

デフォルトのものを除いて、「Unit」は使用前に宣言する必要があります。宣言はスクリプトの先頭にあり、次の型の構文に従う必要があります。

```
uses package_name.unit_name_a;  
  
uses package_name.unit_name_b as alias_b;
```

現在、**package\name** は常に **core** です。ユニットエイリアスは、スクリプト内で一意にする必要があります。

メソッドは、次の構文を使用して参照できます。

```
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);
```

次の例では、**core.list** ユニットを使用してリストを作成します。

```

uses core.list as list;

export function getValue(): typeof _ebx.record.Cities

begin

    return list.of('Paris', 'Bruxelles', 'Berlin');

end

```

提供されているユニットの詳細については、[API ドキュメント](#) を参照してください。

SQL クエリ

EBX® スクリプトは SQL クエリ (select 式) を実行できます。

SQL クエリには、**埋め込み** と **動的** の 2 種類があります。

埋め込みクエリ

可能な限り埋め込みクエリを使用してください。これらはコンパイル時に検証されるため、実行時ではなくコンパイル時にエラーを検出できます。

埋め込みクエリはバッククォートで囲み、**sql** をプレフィックスとして付ける必要があります。

サンプル:

```

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.findDataspaces('myDataspaces'),
'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.

```

動的クエリ

動的クエリは実行時に構築されるため、コンパイル時に検証することはできません。不正なクエリは実行時にエラーを生成します。

サンプル:

```
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.findDataspaces('myDataspaces'),
'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.

```

クエリセッション

スクリプトが SQL クエリにセッションを明示的に設定できるように、プロパティ **query.session** が提供されています。SQL クエリのアクセス制御を無効にするには、クエリセッションに **null** 値を設定できます。

```

// 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;

```

注意: トリガーおよびスクリプトタスクのコンテキストでは、現在のユーザーセッションはデフォルトで SQL クエリに設定されます。関数値のコンテキストでは、SQL クエリにデフォルトのユーザーセッションは設定されません。

ログ取得とデバッグ

ユニット `unit.log` は、メッセージをログに記録するために使用できる関数を提供します。

スクリプトによってログに記録されたメッセージは、>> の EBX® メニューを使用して表示できます。

もう 1 つの便利な機能は、スクリプトの実行中にランタイムでエラーが発生した場合、通常、エラーが発生したスクリプトの行とともにログに記録されることです。

初期スクリプト

`DMA` を使用して新しい機能フィールドまたはテーブルトリガーが作成されると、初期スクリプトが作成されます。

次の例は、テーブルトリガー用に作成されたスクリプトです。

```
uses core.datetime as datetime;

export procedure onBeforeCreate()

begin

    record.inscriptionDate := datetime.now();

end
```

次の例は、文字列型の関数フィールド用に作成されたスクリプトです。

```
// Returns the full name.

export function getValue(): string

begin

    return _ebx.record.FirstName | ' ' | _ebx.record.LastName;
```



```
end
```

初期スクリプトの詳細については、[関数フィールド](#) および [テーブルトリガー](#) を参照してください。

使用方法

- [関数フィールド](#)
- [スクリプトタスク](#)
- [テーブルトリガー](#)
- [データセットトリガー](#)
- [スクリプト化された REST サービス](#)
- [REST サービス](#)
- [ロールマッパー](#)
- [制約検証](#)

関数フィールド

- [はじめに](#)
- [初期スクリプト](#)
 - [はじめに](#)
 - [エクスポートされた関数](#)
- [定義済み変数](#)
 - [はじめに](#)
 - [レコード](#)

はじめに

DSL (ドメイン固有言語) は、関数フィールドを定義するために使用されます。DSL の詳細については、[リファレンス](#) を参照してください。

関数フィールドを作成および変更するには、[データモデルアシスタント \(DMA\)](#) を使用します。

データモデルアシスタント (DMA) には、コンテキストコードの補完を提供するスクリプトエディターが含まれています。

初期スクリプト

はじめに

データモデルアシスタント (DMA) を使用して関数フィールドを初めて編集すると、EBX® はエクスポートされた関数の正しい定義を含む初期スクリプトを提供します。

次の例は、文字列型の関数フィールド用に作成されたスクリプトです。

```
// 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
```

エクスポートされた関数

エクスポートされた関数は、EBX®によって直接呼び出すことができます。スクリプトごとに1つのエクスポートされた関数のみが許可されます。そのシグネチャ (名前と戻り値の型) は関数フィールドの型によって異なり、変更しないでください。

レコードのフィールドの場合、定義は次のようにする必要があります。

```
export function getValue(): typeof _ebx.record.<path>

begin

    ...

end
```

データセットのフィールドの場合、定義は次のようにする必要があります。

```
export function getValue(): typeof _ebx.root.<path>

begin

    ...

end
```

定義済み変数

はじめに

定義済み変数を使用すると、関数フィールドのコンテキストにアクセスできます。次のコンテキスト変数は、関数フィールドに固有のものです。定数であり、不変の複合型です。一般的な定義済み変数の詳細については、[リファレンス](#) を参照してください。

レコード

定義済み変数 `_ebx.record` は、現在のスクリプトがテーブルの関数フィールド用である場合にのみ使用できます。これにより、現在のレコードへの読み取り専用アクセスが可能になります。

そのフィールドは、現在の EBX® スキーマによって定義されています。

例

```
// Returns the full name.

export function getValue(): string

begin

    return _ebx.record.FirstName | ' ' | _ebx.record.LastName;

end
```

スクリプトタスク

- [はじめに](#)
- [初期スクリプト](#)
 - [はじめに](#)
 - [エクスポートされたプロシージャ](#)

- 定義済み変数
 - はじめに
 - `\_ebx.parameters`
- トランザクション
 - はじめに
 - ステートメントの実行

はじめに

ワークフローモデルは、一連のステップによって定義されます。ステップはスクリプトタスクである場合があります。スクリプトタスクは、人間のユーザーの介入なしに実行される自動タスクです。カスタム操作を定義する手続き型言語 DSL (ドメイン固有言語) に基づいています。DSL の詳細については、[リファレンス](#) を参照してください。

タスクの開始準備が整うと、エンジンはスクリプトを実行します。スクリプトが完了すると、タスクも完了します。

スクリプト タスクを作成および変更するには、コンテキストコード補完を提供する EBX IDE エディターを使用します。

初期スクリプト

はじめに

スクリプト IDE を使用してスクリプトタスクを初めて編集すると、EBX® はエクスポートされたプロシージャの正しい定義を含む初期スクリプトを提供します。

初期スクリプトタスクの構造は次のとおりです。

```
<unit usage statement 1>

<unit usage statement 2>

...

<unit usage statement N>

export procedure executeScriptTask()
```

```
begin

    ...

    <statement 1>

    <statement 2>

    ...

end
```

次の例は、メッセージをログに記録するワークフロータスクのスクリプトです。

```
uses core.log as log;

export procedure executeScriptTask()

begin

    log.info('A new script task is executed');

end
```

エクスポートされたプロシージャ

エクスポートされたプロシージャは、EBX® によって直接呼び出すことができます。スクリプトタスクごとに許可されるエクスポートされたプロシージャは 1 つだけです。

スクリプトタスクプロシージャのシグネチャは次のとおりです。

```
export procedure executeScriptTask()
```

このシグネチャは変更できません。

事前定義された変数

はじめに

定義済み変数を使用すると、スクリプトタスクのコンテキストにアクセスできます。次のコンテキスト変数 `\_ebx.parameters` は、スクリプトタスクに固有のものです。定数であり、変更不可能な

複合型です。リファレンスで説明されているコンテキストの一般的な定義済み変数 (関数フィールド、テーブルトリガー) は、スクリプトタスクのコンテキストでは使用できません。

`\_ebx.parameters`

定義済み変数 `\_ebx.parameters` は、スクリプトタスク コンテキストのパラメーターへのアクセスを提供します。スクリプトタスクのパラメーターは、**input** および/または **output** です。これらは、Script IDE を使用して定義されます。定義済み変数 `\_ebx.workflow` は、識別子、作成者、最終変更日などのワークフローモデルに関する情報へのアクセスを提供します。これらのパラメーターは読み取り専用です。

変数 `\_ebx.parameters` のパラメーターには、次のプロパティがあります。

プロパティ	説明
name	パラメーターの名前。
type	パラメーターの型は、文字列、ブール値、ロケールなどになります。
isInput	このプロパティがチェックされている場合、パラメーターは入力として定義されます。パラメーターは変更できません。
isOutput	このプロパティがチェックされている場合、パラメーターは出力として定義されます。パラメーターはスクリプトによって読み取りおよび変更できます。出力パラメーターが入力でない場合、その初期値は null になります。

input パラメーターの値を取得するための構文は次のとおりです。

```
var a_value := _ebx.parameters.input_parameter_name;
```

ここで、`input\parameter\_name` は **input** パラメーターの名前です。

出力 パラメーターの値を設定する構文は次のとおりです。

```
_ebx.parameters.output_parameter_name := a_value;
```

ここで、`output\parameter\_name` は **output** パラメーターの名前です。

例

```

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

定義済み変数 **_ebx.workflow** は、現在のワークフローモデルの情報への読み取り専用アクセスを提供します。このコンテキスト変数は定数であり、不変の複合型です。

次のフィールドがあります。

名前	タイプ	説明
id	string	ワークフローモデルの識別子
publicationId	string	ワークフローモデルの公開識別子
label	message	ワークフローモデルのラベル
description	message	ワークフローモデルの説明
creationDate	date	ワークフローモデルの作成日
lastModificationDate	date	ワークフローモデルの最終変更日

例

```
// 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
```

トランザクション

はじめに

スクリプトタスクのコンテキストでは、ユーザーはトランザクションの下でデータスペースに対して何らかのアクション (レコードの更新、変更、削除など) を実行する必要がある場合があります。したがって、1 つ以上のアクションが失敗した場合、他のすべてのアクションは取り消され、データスペースの状態は変更されません。トランザクションにより、データの整合性状態が損なわれることはありません。

ステートメントの実行

トランザクションを宣言する構文は次のとおりです。

```
execute transaction transaction_variable_name on dataspace_variable

begin

    // Transaction block.

    ...

    <statement_1>

    <statement_2>

    ...

end
```

- **transaction_variable_name** はトランザクション変数の名前です。
- **dataspace_variable** は、トランザクションが実行されるデータスペースです。タイプは 'dataspaceType' です。

包含スコープ内で定義された定数 はトランザクションブロックからアクセスできますが、非定数変数はアクセスできません。

Return ステートメント はトランザクション ブロックでは許可されません。

例

```
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
```

注意

スクリプトタスクごとに複数のトランザクションステートメントを定義できます。トランザクション変数の名前は、現在のスコープ内で一意にする必要があります。そうでない場合、コンパイル時にエラーが生成されます。

```
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
```

スクリプトタスクプロシージャでは、**ネストされた トランザクションは 許可されていません**。コンパイル時にエラーが発生します。

```
export procedure executeScriptTask()  
  
begin  
  
    execute transaction tr on myDataspace  
  
    begin  
  
        <statement_a>;  
  
    end  
  
end
```

```

        // nested transaction is not allowed here !

        execute transaction tr on myDataspace

        begin

        <statement_b>;

        end

    end

end

```

テーブルトリガー

- はじめに
- 初期スクリプト
 - はじめに
 - エクスポート済みプロシージャ
- 定義済み変数
 - はじめに
 - セッション
 - レコード
 - NextRecord
 - PreviousRecord
 - トランザクション

はじめに

トリガーは、作成、更新、削除などの操作がテーブルに対して実行されるときに暗黙的かつ自動的に実行されるプロシージャを定義します。

トリガーは、イベント (挿入、削除、更新) の直前または直後に実行できます。

DSL (ドメイン固有言語) はテーブルトリガーを定義するために使用されます。DSL の詳細については、[リファレンス](#) を参照してください。

テーブルトリガーを作成するには、[データモデルアシスタント \(DMA\)](#) を使用します。

テーブルトリガーのスクリプトを編集および変更するには、コンテキストコード補完を提供する EBX IDE を使用します。

初期スクリプト

はじめに

[DMA](#) を使用してテーブルトリガーを初めて編集すると、EBX® はエクスポートされたプロシージャの正しい定義を含む初期スクリプトを提供します。

テーブルトリガーの初期スクリプトの構造は次のとおりです。

```
<unit usage statement 1>

<unit usage statement 2>

...

<unit usage statement N>


export procedure trigger_signature_name()

begin

    ...

    <statement 1>

    <statement 2>

    ...

end
```

次のスクリプトは、**before create** トリガーの例です。

```
// 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
```

次のスクリプトは、**before modify** トリガーの例です。

```
// this trigger updates automatically the last modification date.

uses core.datetime as datetime;

export procedure onBeforeModify()

begin

_ebx.record.lastModificationDate := datetime.now();

end
```

エクスポート済みプロシージャ

エクスポート済みプロシージャは、EBX® によって直接呼び出すことができます。テーブルトリガー内のスクリプトごとに、エクスポート済みプロシージャは1つだけ許可されます。

テーブルトリガープロシージャのシグネチャは次のとおりです。

- **onNewContext()** : レコードの新しいコンテキストが作成された時に呼び出されます。
- **onBeforeCreate()** : レコードが作成される前に呼び出されます。
- **onAfterCreate()** : レコードの作成後に呼び出されます。
- **onBeforeModify()** : レコードが変更される前に呼び出されます。
- **onAfterModify()** : レコードの変更後に呼び出されます。

- **onBeforeDelete()** :レコードを削除する前に呼び出されます。
- **onAfterDelete()** :レコードを削除した後に呼び出されます。
- **onBeforeTransactionCancel()** :キャンセルの前に呼び出されます。
- **onBeforeTransactionCommit()** :コミット前に呼び出されます。

注意：エクスポートされたプロシージャのシグネチャ (名前) はトリガーのタイプによって異なり、変更しないでください。

事前定義された変数

はじめに

定義済み変数を使用すると、テーブルトリガーのコンテキストにアクセスできます。次のコンテキスト変数は、テーブルトリガーに固有のものです。一般的な定義済み変数の詳細については、[リファレンス](#) を参照してください。

セッション

定義済み変数 `\_ebx.session` は、現在のセッションに関する情報へのアクセスを提供します。このコンテキスト変数は定数であり、不変の複合型です。

次のフィールドがあります。

名前	タイプ	説明
userId	string	現在のユーザーの識別子
userEmail	string	現在のユーザーのメールアドレス
locale	string	現在のセッションのロケール

例

```
// 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

```

レコード

定義済み変数 `\_ebx.record` を使用すると、作成、更新、または削除する現在のレコードへの読み取り/書き込みアクセスが可能になります。そのフィールドは、現在の EBX® スキーマによって定義されます。

例

```

// 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

定義済み変数 `\_ebx.nextRecord` を使用すると、次に更新するレコードへの読み取り/書き込みアクセスが可能になります。この変数は、**onBeforeModify** トリガーでのみ使用できます。この変数は、現在の EBX® スキーマによって定義されます。

例


```

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

end

```

PreviousRecord

定義済み変数 `\_ebx.previousRecord` を使用すると、更新後に前のレコードへの読み取り専用アクセスが可能になります。これは **OnAfterModify** トリガーでのみ使用できます。これは現在の EBX® スキーマによって定義されます。

例

```

// 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

```

トランザクション

定義済み変数 `_ebx.transaction` は、トリガーの現在のトランザクションへの読み取り専用アクセスを提供します。これは、テーブルトリガーとデータセットトリガーのコンテキストでのみ使用できます。この変数は定数であり、変更不可能な複合型です。トランザクションは、データスペースでいくつかの操作 (レコードの更新、変更、削除など) を実行するために必要です。これにより、アクションが失敗しても、データの整合性状態が損なわれることはありません。

例

```

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

```

データセットトリガー

- はじめに
- 初期スクリプト
 - はじめに
 - エクスポートされたプロシージャ
- 事前定義済み変数
 - はじめに
 - セッション
 - originalDataset
 - トランザクション

はじめに

トリガーは、データセットに対して何らかの操作が実行されたときに、暗黙的かつ自動的に実行されるプロシージャを定義します。トリガーは、イベント（作成、更新、または複製）の直前または直後に実行できます。

DSL (Domain Specific Language) は、データセットトリガーを定義するために使用されます。DSL の詳細については、「[リファレンス](#)」を参照してください。

データセットトリガーを作成するには、「[データモデルアシスタント \(DMA\)](#)」を使用します。

データセットトリガーのスクリプトを編集、変更、および公開するには、コンテキストに応じたコード補完を提供する EBX® IDE を使用します。

初期スクリプト

はじめに

データセットトリガーが EBX® IDE で初めて編集されると、エクスポートされたプロシージャの正しい定義を含む初期スクリプトが提供されます。

データセットトリガーの初期スクリプトは、次の構造を持ちます。

```
<unit usage statement 1>

<unit usage statement 2>
```

```

...

<unit usage statement N>

export procedure trigger_signature_name()

begin

    ...

    <statement 1>

    <statement 2>

    ...

end

```

次のスクリプトは、**変更前**トリガーの例です。

```

// this trigger updates root fields before modifying the dataset.

export procedure onBeforeModify()

begin

    _ebx.root.fieldA:='xx';

    _ebx.root.fieldB:='yy';

end

```

エクスポートされたプロシージャ

エクスポートされたプロシージャは、EBX® によって直接呼び出すことができます。データセットトリガーのスクリプトごとに許可されるエクスポートされたプロシージャは1つだけです。

データセットトリガープロシージャのシグネチャーは次のとおりです。

- **onBeforeCreate()** : データセットの作成前に呼び出されます

- **onAfterCreate()** : データセットの作成後に呼び出されます
- **onBeforeModify()** : データセットの変更前に呼び出されます
- **onAfterModify()** : データセットの変更後に呼び出されます
- **onBeforeDelete()** : データセットの削除前に呼び出されます
- **onAfterDelete()** : データセットの削除後に呼び出されます
- **onAfterDuplicate()** : データセットの複製後に呼び出されます

注: エクスポートされたプロシージャのシグネチャ (名前) はトリガーのタイプによって異なり、変更しないでください。

事前定義済み変数

はじめに

事前定義済み変数を使用すると、データセットトリガーのコンテキストにアクセスできます。次のコンテキスト変数は、データセットトリガーに固有のものです。共通の事前定義済み変数の詳細については、「[リファレンス](#)」を参照してください。

セッション

事前定義済み変数 `\_ebx.session` は、現在のセッションに関する情報へのアクセスを提供します。このコンテキスト変数は定数であり、不変の複合型です。この変数は、テーブルトリガーのコンテキストでも使用できます。

次のフィールドがあります。

名前	タイプ	説明
<code>userId</code>	<code>string</code>	現在のユーザーの識別子
<code>userEmail</code>	<code>string</code>	現在のユーザーのメールアドレス
<code>locale</code>	<code>string</code>	現在のセッションのロケール

例:

```
// 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

事前定義済み変数 `\_ebx.originalDataset` は、元のデータセットの情報へのアクセスを提供します。これは、複製後のトリガーでのみ使用できます。

この変数には次のフィールドがあります。

名前	タイプ	説明
name	string	元のデータセットの名前
dataspace	dataspace	データセットのデータスペース

例:

```

// 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

```

トランザクション

トランザクションは、作成、削除、複製などのデータセットに対する操作を実行するために必要です。事前定義済み変数 `_ebx.transaction` は、データセットトリガーの現在のトランザクションへの読み取り専用アクセスを提供します。これは、データセットトリガーの **afterCreate**、**afterModify**、**afterDelete**、および **afterDuplicate** でのみ使用できます。この変数は定数であり、不変の複合型です。

例:

次のスクリプトは、複製後トリガーの例です。

```
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
```

次のスクリプトは、**作成後トリガー**の例です。

```
// 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
```

スクリプト化されたRESTサービス

- はじめに
- エンドポイントマッピングとoperationId
- エクスポートされた関数
- パラメーター: 解析と検証
- レスポンスの生成
- スキーマ、型、およびOpenAPI
- サポートされるOpenAPIサブセット
- 認証とセッション
- トランザクションとデータアクセス
- エラー処理
- 関連項目

はじめに

スクリプト化されたRESTサービスは、EBX®におけるRESTエンドポイントとその関連スクリプトを定義します。これらのサービスはスクリプトIDEで作成および保守します。IDEからサービスをコンパイルして公開すると、RESTクライアントで利用できるようになります。

スクリプト化されたRESTサービス内の各操作は、HTTPリクエストを介して操作が呼び出されたときに実行されるEBX®スクリプトのエクスポートされた関数として実装されます。エクスポートされた関数は** rest_response**値を返します。**core.rest**ユニットは、レスポンスを構築するためのユーティリティを提供します。リクエストとレスポンスはJSONコンテンツのみを使用します。

エンドポイントマッピングとoperationId

スクリプト化されたRESTサービスは、1つ以上のエンドポイントを定義します。各エンドポイントにはURLパスと1つ以上のHTTPメソッドがあります。各メソッドには、OpenAPIの** operationIdとして解釈される Operation id**を割り当てます。この値は、クライアントがそのメソッドでエンドポイントを呼び出したときに実行される、エクスポートされたEBX®スクリプト関数を識別します。**Operation id**の値は、対応するエクスポートされた関数の名前と完全に一致する必要があります。

URLパスには、パスパラメーター（中括弧で囲まれたもの）とクエリパラメーターを含めることができます。パラメーターは、関連する関数によって次のように使用されます。

- パラメーターの型はOpenAPIスキーマから推論されます。[スキーマ](#)、[型](#)、および[OpenAPI](#)で詳細を参照してください。
- 各パスパラメーターは関数パラメーターになります。
 - パラメーター名はパスパラメーター名と一致します。
 - パラメーターの型はOpenAPIスキーマから推論されます。
 - 関数内のパスパラメーターの順序は、パステンプレート内の順序と一致します。
- 操作用に定義されたクエリパラメーターも関数パラメーターになり、OpenAPIドキュメントで宣言された順序でパスパラメーターの後に続きます。

例 (名前は** operationId**を反映):

OpenAPI (抜粋):

```
{
  "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" } } }

    }

}

}

```

同等のエクスポートされたメソッド:

```

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

```

エクスポートされた関数

スクリプト化されたRESTサービスのコンテキストでは、エクスポートされた関数はREST操作ハンドラーです。各エクスポートされた関数は** rest_response**を返し、サービス契約で定義されているパス、クエリ、またはボディのパラメーターを受け入れることができます。

- パスおよびクエリパラメーターは、**string**、**int**、**decimal**、**boolean**、**date**、**time**、または**datetime**などのスカラー型を使用して宣言します。
- リクエストボディが存在する場合、通常は**components.schemas** APIから生成された型付きオブジェクトとして、または非構造化JSONの場合は**json_value**として宣言されます。
- 関数名はOpenAPIの**operationId**と一致する必要があります。
- パラメーターの順序: パスパラメーターが最初（パス順）、次にクエリパラメーター（OpenAPIドキュメント順）です。

完全な例については、上記のOpenAPIとエクスポートされたメソッドの組み合わせを参照してください。

パラメーター: 解析と検証

スクリプト化されたRESTサービスは、パラメーターの解析と検証を自動的に処理します。

- パスおよびクエリパラメーターは、宣言された型に従って解析されます。

- 無効な数値や日付は、標準のエラーペイロードとともに400を返します。たとえば、「Path parameter 'year' is not valid.」または「Query parameter 'date' is not valid.」のように適用されます。
- ブール値のパスパラメーターの場合、「true」以外の値は拒否されるのではなく、falseに強制されます。
- 空のパスセグメントは、スクリプトに到達する前にコンテナーによって400で拒否されます。

レスポンスの生成

エクスポートされた関数からレスポンスを生成するには、**core.rest**ユーティリティを使用します。

- **rest.responseOf(status: int, content: any)**は、オプションのステータスコードを持つJSONレスポンスを作成します。
- **status**がnullの場合、200が想定されます。
- **content**は、JSONオブジェクト型、任意の**json_value**、またはこれらのリストにすることができます。
- **rest.emptyResponse(status: int)**は、ボディなしのレスポンスを作成します。
- **status**がnullの場合、204が想定されます。

スキーマ、型、およびOpenAPI

OpenAPI 3は操作とデータシェイプを記述します。スクリプト化されたRESTサービスでは、次のようになります。

- 各**components.schemas**エントリは、同じ名前のEBX®スクリプト型になります（例: **User**、**Error**、**Meeting**、**Query**）。フィールドはスキーマプロパティに1対1でマッピングされます。
- スカラーは次のようにマッピングされます。
 - integer (format: int32) → **int**
 - number → **decimal**
 - boolean → **boolean**
 - string → **string**
 - string (format: date-time) → **datetime**
 - string (format: date) → **date**
 - string (format: time) → **time**

- レスポンス/ボディは、型付きオブジェクト、型付きオブジェクトのリスト、または**json_value**を介した汎用JSONにすることができます。

例:

```
// 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
```

OpenAPI駆動型テンプレートを使用する場合、オブジェクト型とフィールド名はコンポーネントスキーマから派生します。明確なコンパイル時検証のために、返される構造を宣言された型と一致させてください。

サポートされるOpenAPIサブセット

スクリプト化されたRESTサービスは、OpenAPI 3.0機能のサブセットをサポートしています。

- パス: キーはパステンプレートとして検証されます。ベースルート（スクリプト名がベースパスを定義します）を含まないようにしてください。

- メソッド: **GET**、**POST**、**PUT**、**DELETE**がサポートされています。レスポンスキーは、有効な範囲の整数HTTPコードである必要があります。
- セキュリティ: **security**: は匿名アクセスを許可します。その他の **security**オブジェクトはサポートされていません。
- パラメーター:
 - パスレベル: **in: path**のみが処理されます。名前はパステンプレートに存在する必要があります。
 - メソッドレベル: **in: query**のみが処理されます。**path**はパスレベルで宣言する必要があります。
 - 各パラメーターは、サポートされているプリミティブ型 (**string**、**integer**、**number**、**boolean**) とオプションのフォーマット (**date**、**time**、**date-time**) を持つ** schemaを宣言する必要があります。パラメーターは配列またはオブジェクトにすることはできず、**\$ref****を使用することはできません。
- リクエストボディ: **application/json**のみが考慮されます。スキーマは、コンポーネントスキーマへの**\$ref**、または適用可能なインラインスカラー/配列/オブジェクトである場合があります。
- レスポンス: **application/json**のみが考慮されます。各ステータスは、オプションでJSONスキーマ (スカラー、配列、または**\$ref**) を宣言できます。
- コンポーネントスキーマ: プロパティを持つ** object型のみがサポートされます。**additionalProperties**は存在しないか、**false**である必要があります。**\$ref**ターゲットは#/**components/schemas****の下にある必要があります。

認証とセッション

スクリプト化されたRESTサービスを、匿名アクセスを許可するか、認証を要求するように構成します。エンドポイントの設計で必要とされない限りユーザーコンテキストを想定しないようにハンドラーを実装し、アクセスを制限する必要がある場合は適切なステータスコードを返します。

- 実行時、**_ebx.anonymous**はリクエストが匿名であるかどうかを示します。
- 匿名フレンドリーなエンドポイントの場合、OpenAPI操作は通常** security: **を設定します。
- ユーザーセッションが存在する場合は**_ebx.session.userId**を使用します。

例:

```

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;

    end if

    var content := list.of(json.ofString(userId));

    return rest.responseOf(200, content);

end

```

トランザクションとデータアクセス

通常のEBX®スクリプトトランザクションおよびデータAPIを使用して、EBX®データを読み取りおよび変更します。次の例には、トランザクションブロックと、ブロック全体でレスポンスを伝達するための** ref**を含む典型的なパターンが含まれています。

```

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.findDataspaces('rest-test') as system
    begin
        const directory := db.findDataset(tr.dataspaces, '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
        end
    end
end

```

```

        end

    else

    begin

        record.firstName := user.firstName;

        record.lastName := user.lastName;

        db.saveRecord(record);

        response := rest.responseOf(200, user);

    end

end

return response;

end

```

リクエストを昇格された権限で実行する必要がある場合は、**execute transaction ... as system**を使用します。それ以外の場合は、認証されたユーザーコンテキストを使用するために** as system**を省略します。

エラー処理

スクリプト化されたRESTサービスは、エラーを次のように処理します。

- 無効なパスまたはクエリパラメーターは、**400**と標準のエラーペイロードとともに自動的に拒否されます。
- ビジネスエラー（見つからない、既に存在する、検証）の場合、適切なステータスコード（例: **404**、**400**）とともに型付きエラーコンテンツを返します。

例:

```
var err := instanceOf<Error>();  
  
err.message := 'User not found';  
  
return rest.responseOf(404, err);
```

関連項目

- [RESTサービス](#) (DSLからの外部サービスの呼び出し)
- [リファレンス](#) (言語リファレンスと型)
- [RESTサービスチュートリアル](#) (スクリプト化されたRESTサービスの作成に関する詳細なチュートリアル)

RESTサービス

- [はじめに](#)
- [設定と認証](#)
- [RESTサービスの呼び出し](#)
 - [リクエストの作成](#)
 - [リクエストの送信](#)
 - [レスポンスの読み取り](#)
 - [レスポンスエラーの処理](#)

はじめに

EBX® スクリプトは、**HTTP API** を使用して外部RESTサービスを呼び出すことができます。このAPIは以下のメソッドを提供します。

- [HTTPリクエストの作成](#)
- [ヘッダーへのパラメーターの追加](#)
- [レスポンスの読み取り](#)（メッセージボディ、HTTPコードなど）

サーバーリソースに対して操作を実行するには、以下のHTTPメソッドを使用します。

- **GET** : サーバーからリソースまたはコレクションを取得
- **POST** : サーバー上に新しいリソースを作成するためのデータを送信
- **PUT** : サーバー上の既存のリソースを更新するためのデータを送信
- **DELETE** : サーバー上のリソースを削除

RESTサービスは、EBX® スクリプトIDEを使用して、[テーブルトリガー](#)および[スクリプトタスク](#)から呼び出すことができます。

設定と認証

スクリプトで呼び出される各RESTサービスは、以下の[テーブル](#)で設定します。 [管理](#) > [統合](#) > [外部サービス](#) > [設定](#)。以下のプロパティが必要です。

- **設定ID** : REST設定の一意的識別子。APIでHTTPリクエストを作成するために必要です。
- **ベースURL** : 呼び出すサーバーのURL。クライアントとサーバー間の通信を暗号化するにはHTTPSを使用します。URLの形式は次のとおりです。

```
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.

- **使用方法** : サービスの利用方法を定義します。AIアシスタントおよび/またはRESTサービス。
- **認証タイプ** : リソースへのアクセスに使用される認証タイプを指定します。以下のタイプがサポートされています。

タイプ	説明
Basic	有効なユーザー名とパスワードを必要とする、シンプルで高速なHTTP認証方法です。資格情報はBase64でエンコードされ、HTTPリクエストヘッダーに含まれます。認証コードをリクエストのヘッダーに追加する必要はありません。サーバーへの各アクセス前に自動的に含まれます。

タイプ	説明
EBX	EBX組み込みRESTfulサービスへのアクセスを可能にします。認証トークンを取得するには、有効なユーザー名とパスワードを提供する必要があります。トークンにはタイムアウト期間があります。EBX® サーバーへの各アクセス時に、新しいトークンがリクエストされ、HTTP リクエストヘッダーに含まれます。スクリプトからEBXトークンをヘッダーに追加する必要はありません。EBX® サーバーへの各アクセス前に自動的に含まれます。
Custom	有効なキー名とキー値を提供することで、カスタム認証を可能にします。キーの値は秘密にすることができ、キーストアに基づいて暗号化されます。このカスタムタイプは、APIキー、トークン、OAuthなどの認証を使用してサービスを呼び出すために使用できます。
None	認証は不要です。

RESTサービスの呼び出し

EBX® スクリプトからRESTサービスを呼び出すには、次の手順を実行します。

- 管理 > 統合 > 外部サービス > 設定 に移動し、サーバーを設定します。設定ID、URL、認証タイプ、および資格情報を設定します。
- EBX® スクリプトIDE内でテーブルトリガーまたはスクリプトタスクにアクセスします。
- 指定された設定IDを使用してRESTサービスを呼び出します。

リクエストの作成

core.http APIを使用して、新しいHTTPリクエストを作成できます。このAPIは、リクエストの設定、パス、パラメーター、およびヘッダーフィールドの追加のためのメソッドを提供します。

以下の例は、RESTサービスを呼び出すいくつかのスクリプトを示しています。

例：特定のカテゴリのすべての製品を取得するリクエストの作成

GET https://api.example.com/v1/products/?sort=desc&category=jewelery

以下の設定が使用されます。

- 設定ID: ebx-1
- URL: <https://api.example.com>
- 使用方法: Rest
- 認証タイプ: basic
- 資格情報: ユーザー名、パスワード

```
uses core.http as client;

...

procedure getJewelleryProducts()
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', 'jewellery');

    client.addParameter(request, 'sort', 'desc');
```



```

...

// sends a request to get products of a specific category

...

end

```

例：結果を制限してすべての製品を取得するリクエストの作成

GET <https://api.example.com/v1/products?limit=15>

以下の設定が使用されます。

- 設定ID: ebx-2
- URL: <https://api.example.com>
- 使用方法: Rest
- 認証タイプ: custom
- キー名: api_key
- キー値: 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
end

```

例：EBX組み込みRESTfulサービス呼び出してユーザーメールを更新するリクエストの作成

PUT https://ebx-example.io/test/ebx-dataservices/rest/data/v1/Bdataspace/dataset/root/user/
ID6766

以下の設定が使用されます。

- 設定ID: ebx-3
- URL: https://ebx-example.io/test/ebx-dataservices
- 使用方法: Rest
- 認証タイプ: ebx
- 資格情報: ユーザー名、パスワード

```

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

```

例：都市に関する情報を取得するリクエストの作成

GET <https://geo.api.gouv.fr/communes?codePostal=92160&fields=nom,departement,region>

以下の設定が使用されます。

- 設定ID: ebx-4
- URL: <https://geo.api.gouv.fr/communes>
- 使用方法: Rest
- 認証タイプ: 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

```

リクエストの送信

[core.http](#) APIは、RESTサービス呼び出すためのいくつかのHTTPメソッド（GET、PUT、POST、DELETE）を提供します。以下の例は、HTTPメソッドを使用して製品に対する操作を実行するいくつかのスクリプトを示しています。

GET

例：サーバーから単一の製品を取得

GET <https://api.example.com/v1/products/P87RZ63>

以下のスクリプトは、識別子によって単一の製品を取得するためのGETリクエストを送信します。

```
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

例：サーバー上に新しい製品を作成

POST <https://api.example.com/v1/products>

以下のスクリプトは、新しい製品を追加するためのPOSTリクエストを送信します。

```
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

例：サーバー上の既存の製品を更新

PUT <https://api.example.com/v1/products/L32LQ>

以下のスクリプトは、特定の製品を更新するためのPUTリクエストを送信します。

```

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

例：サーバー上の既存の製品を削除

DELETE <https://api.example.com/v1/products/L32LQ>

以下のスクリプトは、特定の製品を削除するためのDELETEリクエストを送信します。

```

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

```

レスポンスの読み取り

[core.http](#) APIを使用して、レスポンスのステータスコード、ヘッダー、およびボディを読み取ります。[json.http](#)を使用して、レスポンスボディを解析し、JSONデータを管理します。

例：特定の製品に関する情報を取得するリクエストのボディレスポンスの読み取り

```

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
```

レスポンスエラーの処理

エラーが発生した場合は、ステータスコードとレスポンスボディで報告される追加情報を確認してください。core.http APIは、リクエストURLの取得、ステータスコードの取得、およびレスポンスボディ内のエラーメッセージの読み取りのためのメソッドを提供します。

例：EBX組み込みRESTfulサービスにユーザーメールを更新するリクエストを送信する際、レスポンスエラーをログに記録

```
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
```

ロールマッパー

- はじめに
- 設定

- 初期スクリプト
- 関数シグネチャ
 - パラメーター
 - 戻り値
- 利用可能な関数
- 例
 - 基本的なロールマッピング
 - 属性ベースのロールマッピング
- トラブルシューティング

はじめに

ロールマッパースクリプトは、IDプロバイダーからの外部ロールを、[同期ディレクトリ](#)機能の一部として、内部EBX®ロールにマッピングするための柔軟なアプローチを提供します。

ロールマッパースクリプトを作成、変更、および公開するには、EBX® Script IDEを使用します。

設定

ロールマッパースクリプトは、**ebx.properties** の同期ディレクトリセクションで設定されます。

```
# Enable synchronized directory

ebx.directory.synchronized.source=LDAP

...


# Set role mapper to use scripts

ebx.directory.synchronized.role.mapper=SCRIPT
```

初期スクリプト

EBX® Script IDEを使用してロールマッパースクリプトが初めて編集されると、EBX®はエクスポートされた関数の正しい定義を含む初期スクリプトを提供します。

初期ロールマッパースクリプトは次の構造を持ちます。

```

<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

```

以下の例は、基本的なロールマップスクリプトです。

```

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

    end

    return result;

end

```

関数シグネチャ

mapRoles 関数は次のシグネチャを持つ必要があります。

```

export function mapRoles(userAttributes: unmodifiable
dictionary<set<string>>,

                                externalRoles: unmodifiable set<string>):
set<profile>

```

パラメーター

- **userAttributes** : IDプロバイダーからのユーザー属性を含む辞書。各キーは属性名であり、各値は属性値を表す文字列のセット

例: {'department': {'IT', 'Engineering'}, 'location': {'Paris'}, 'title': {'Senior Developer'}}

- **externalRoles** : IDプロバイダーによってユーザーに割り当てられた外部ロールのセット

例: {'admin', 'developer', 'manager'}

戻り値

この関数は、ユーザーに割り当てられる内部ロールを表す** profile**オブジェクトのセットを返す必要があります。

注: 関数が空のセットを返した場合、ユーザーにはロールが割り当てられません。

利用可能な関数

以下のプロファイル関数は、[core.profile](#) ユニットを使用して利用できます。

- **profile.ofAdministrator()** : 組み込みの管理者プロファイルを返します
- **profile.ofReadOnly()** : 組み込みの読み取り専用プロファイルを返します
- **profile.ofOwner()** : 組み込みの所有者プロファイルを返します
- **profile.ofEveryone()** : 組み込みの全員プロファイルを返します
- **profile.forSpecificRole(roleName)** : 指定された名前の特定のロールプロファイルを返します

例

基本的なロールマッピング

この例は、外部ロールを内部EBX®プロファイルに直接マッピングする方法を示します。

```
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

```

属性ベースのロールマッピング

この例は、ユーザー属性（部署など）を使用してロールの割り当てを決定する方法を示します。

```

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

```

トラブルシューティング

スクリプトが実行されない場合:

- **ebx.properties** で **ebx.directory.synchronized.role.mapper=SCRIPT** が設定されていることを確認します
- スクリプトがEBX® Script IDEで公開され、アクティブであることを確認します
- ランタイムエラーがないかスクリプトログを確認します

制約の検証

- はじめに
- 初期スクリプト
 - 構造
 - エクスポートされたプロシージャ
- 事前定義変数

- データセット
- データスペース
- ルート
- 制約タイプ
 - テーブルに対する制約
 - フィールドに対する制約
 - Nullに対する制約
- 制約の依存関係
 - 挿入、削除、および変更の依存関係
 - 挿入および削除の依存関係
 - 変更の依存関係
 - 別のデータセットへの依存関係

はじめに

EBX® Scriptを使用すると、データモデルに検証制約を定義できます。制約は、ユーザーが有効で正確かつ論理的に一貫性のあるデータのみを入力するようにするデータモデル上のルールです。制約は、テーブル、レコード、テーブルフィールド、データセットフィールドなど、さまざまなレベルで定義できます。

制約が無効なデータを検出すると、検証レポートにカスタムメッセージを追加できます。メッセージテキストと重大度レベルを選択できます。

制約は、高度なコントロールビューのデータモデルアシスタント (DMA)で作成します。EBX®は、コンテキストに応じたコード補完を提供するEBX® Script IDEで編集できる初期スクリプトを生成します。

制約内のローカライズされたユーザーメッセージを管理するには、EBX®スクリプトAPIの `core.message` を使用します。このAPIは、検証レポートにユーザーフレンドリーなメッセージを作成および追加するためのいくつかのメソッドを提供します。

初期スクリプト

高度なコントロールを使用してDMAで検証制約を作成すると、EBX®は初期スクリプトを生成します。初期スクリプトには、制約タイプに応じた正しい構造と必要なエクスポートされたプロシージャが含まれています。

構造

制約の初期スクリプトは次の構造を持ちます。

```
<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
```

次のスクリプト例は、部署内の従業員に対する制約を示しています。従業員番号が100未満であるかどうかをチェックします。100未満の場合、検証レポートのコンテキストに警告メッセージが追加されます。

```
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
```

エクスポートされたプロシージャ

EBX®は、制約を評価する際にエクスポートされたプロシージャを直接呼び出します。各制約スクリプトは、そのタイプに関連するエクスポートされたプロシージャのみを宣言できます（制約タイプに応じて2つまたは3つのプロシージャ）。エクスポートされたプロシージャ **setup()** は必須

です。EBX®は、制約コンテキストを初期化するためにこれを呼び出します。制約の依存関係の宣言など、初期化タスクを実行するために使用します。

EBX®には次の制約タイプが含まれています。

- **テーブルに対する制約:** テーブル全体が特定の条件を満たすことを保証するためのテーブルレベルのチェックを定義します。また、レコードが特定の条件を満たすことを保証するためのレコードレベルのチェックの定義も可能です。
- **フィールドに対する制約:** フィールドが特定の条件を満たすことを保証するためのフィールドレベルのチェックを定義します。
- **Nullに対する制約:** フィールドが必須であり、Nullであってはならないことを保証するためのフィールドレベルのチェックを定義します。

制約プロシージャのシグネチャは次のとおりです。

- **checkTable()** および **checkRecord()** : 制約タイプがテーブルに対する制約の場合に呼び出されます。
- **check()** : 制約タイプがフィールドに対する制約の場合に呼び出されます。
- **checkNull()** : 制約タイプがNullに対する制約の場合に呼び出されます。
- **setup()** : 制約を初期化するために呼び出されます。このプロシージャは、すべての制約タイプで必須です。

注: エクスポートされたプロシージャのシグネチャ（名前）は制約のタイプに依存するため、変更しないでください。エクスポートされたプロシージャ **setup()** は必須であり、変更しないでください。

事前定義変数

事前定義変数を使用すると、制約コンテキストで利用可能な情報にアクセスできます。制約タイプに応じて、次の事前定義変数を使用できます。

データセット

事前定義変数 `\_ebx.dataset` は、現在のデータセットに関する情報へのアクセスを提供します。これは、すべてのEBX® Scriptの使用法（トリガー、関数フィールドなど）に共通の事前定義変数です。詳細については、[リファレンス](#)セクションを参照してください。また、すべての制約タイプで利用可能です。

例:

```

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

```

データスペース

事前定義変数 `_ebx.dataspace` は、現在のデータスペースへの読み取りアクセスを許可します。これは、すべてのスクリプトの使用法（トリガー、関数フィールドなど）に共通の事前定義変数です。詳細については、[リファレンス](#)を参照してください。また、すべての制約タイプで利用可能です。

例:

```

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

```

ルート

事前定義変数 `\_ebx.root` は、データセットの現在のルートへの読み取り専用アクセスを提供します。これはすべての制約タイプで利用可能です。ルート下のフィールドは、現在のEBX®スキーマによって定義されます。

例:

```

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
```

制約タイプ

検証制約は、データモデル内のデータの一貫性を確保するための特定のルールを定義します。これは、データモデルのさまざまなレベル（テーブル、レコード、またはフィールド）に適用できます。検証制約が違反された場合、ユーザー定義の重大度レベルを持つカスタマイズされたエラーメッセージを検証レポートに追加できます。以下に、さまざまなレベルでの検証制約の例を示します。

テーブルに対する制約

テーブルレベルのチェック

この制約タイプはテーブル全体をチェックし、エラーが検出された場合にメッセージを追加します。テーブル全体が一度にチェックされます。SQLクエリとXPath式を使用して、チェックする特定のレコードを選択できます。

例:

```
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
```

レコードレベルのチェック

この制約は、テーブルレベルのチェックに加えて、レコードレベルのチェックをサポートします。特定のレコードに対するカスタム検証を定義できます。これにより、ユーザーがデータを入力したり、レコードを変更したりする際のエラーを検出するのに役立ちます。

checkRecord メソッドの **record** パラメーターは、現在のレコードへの読み取り専用アクセスを提供します。これは、制約タイプ **テーブルに対する制約** のレコードレベルチェックで利用可能です。レコードのフィールドは、現在のEBX®スキーマによって定義されます。

例 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

```

例 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

```

フィールドに対する制約

この制約タイプはフィールドレベルのチェックを定義します。フィールドが特定の条件を満たすことを保証します。フィールドが無効な場合、ローカライズされたメッセージを追加できます。この制約タイプは、データセットフィールドまたはテーブルフィールドに定義できます。

データセットフィールドレベルのチェック

この制約タイプは、データセットフィールドレベルのチェックを定義します。他のデータセットフィールドにアクセスするには、事前定義変数 `_ebx.root` を使用します。

例:

```
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

end

export procedure setup()

begin

    ...

end
```

テーブルフィールドレベルのチェック

この制約タイプは、テーブルフィールドレベルのチェックを定義します。テーブル内の各レコードに対してフィールドが特定の条件を満たすことを保証します。フィールドが無効な場合、ローカライズされたメッセージを追加できます。変数 **record** は **check()** のパラメーターとして渡されます。これは、現在のフィールドを含むレコードへの読み取り専用アクセスを提供します。

例:

```
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

end

export procedure setup()

begin

end
```

Nullに対する制約

この制約タイプは、フィールドが値を含むかどうかをデータセットまたはテーブルフィールドレベルでチェックします。フィールドがNullの場合、メッセージを追加できます。

データセットフィールドレベルのチェック

例:

```
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
```

テーブルフィールドレベルのチェック

例:

```
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
```

制約の依存関係

制約の検証結果は、特定のフィールドの更新、挿入、または削除に依存する場合があります。これらの場合、制約の初期化中にフィールドの依存関係を明示的に宣言できます。制約の依存関係を宣言することで、検証プロセス中の制約への呼び出し回数を最小限に抑えることができます。

依存関係を指定する必要はありません。ただし、明示的な依存関係は、大量のデータを検証する場合や、検証ロジックが高コストである場合にパフォーマンスを向上させます。

EBX® Script APIの[core.validation](#)は、制約の依存関係を宣言するためのいくつかのメソッドを提供します。

警告: 計算値または継承されたフィールドに対する依存関係は指定できません。

挿入、削除、および変更の依存関係

制約の検証結果は、指定されたフィールドの挿入、変更、および削除に依存します。

例:

```
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

```

挿入および削除の依存関係

制約の検証結果は、指定されたテーブルの挿入および削除に依存します。

例:

```

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

```

変更の依存関係

制約の検証結果は、指定されたフィールドの値に依存します。

例:

```

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

```

別のデータセットへの依存関係

制約の検証結果は、指定されたデータセットおよびデータスペース内の指定されたフィールドまたはテーブルの挿入、変更、および削除に依存します。

例:

```

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
```

API

- ユニットの概要
- デフォルトのユニット
- ユニット core.complex
- ユニットコアデータ
- ユニット core.date
- ユニット core.datetime
- ユニット core.dictionary
- ユニット core.http
- ユニット core.json
- ユニット core.list
- ユニット core.locale
- ユニット core.log
- ユニット core.math
- ユニット core.message
- ユニット core.profile
- ユニット core.resource
- ユニット core.set
- ユニット core.string
- ユニット core.time
- ユニット core.uri
- ユニット core.validation
- ユニット core.rest

ユニットの概要

ユニット	説明
default	デフォルトのユニット
core.complex	複雑な値を作成および更新するためのメソッドを提供するスクリプトユニット
core.data	データベース操作 (挿入、変更、更新、検索) を提供するスクリプトユニット
core.date	日付関数を提供するスクリプトユニット
core.datetime	日時関数を提供するスクリプトユニットです
core.dictionary	辞書を作成および操作するためのメソッドを提供するスクリプトユニット
core.http	外部 REST サービスを呼び出すためのメソッドを提供するスクリプトユニット
core.json	JSON データを管理するためのメソッドを提供するスクリプトユニット
core.list	リストを作成および更新するためのメソッドを提供するスクリプトユニットです
core.locale	ロケールを管理するための機能を提供するスクリプトユニット
core.log	メッセージをログに記録するメソッドを提供するスクリプトユニット
core.math	数学関数と定数を提供するスクリプトユニットです
core.message	ユーザーメッセージを作成および管理するためのメソッドを提供するスクリプトユニット
core.number	数値関数と定数を提供するスクリプトユニット

ユニット	説明
core.profile	プロフィールを管理する際に役立つメソッドを提供するスクリプトユニット
core.resource	このスクリプトユニットは、スキーマ <code>ods:resource</code> と互換性のあるリソース参照を生成するメソッドです
core.rest	REST サービスを作成する際に役立つメソッドを提供するスクリプトユニット
core.set	セットを作成および操作するためのメソッドを提供するスクリプトユニット
core.string	文字列操作機能を提供するスクリプトユニット
core.time	時間関数を提供するスクリプトユニットです
core.uri	URI (Uniform Resource Identifier) 参照を管理するための関数を提供するスクリプトユニットです
core.validation	検証制約のコンテキストで依存関係を設定するためのメソッドを提供するスクリプトユニット

デフォルトのユニット

デフォルトのユニット。このユニットは自動的にスクリプトに含まれます。

このスクリプトのメソッドは直接呼び出すことができ、`uses` ステートメントは必要ありません。

メソッド
<code>instanceOf<type>(): type</code> 指定された型のオブジェクトインスタンスを作成します。
<code>function isFalseOrNull(value: boolean): boolean</code> 値が <code>false</code> または <code>null</code> の場合は <code>true</code> を返します。
<code>function isNotNull(value: any_type): boolean</code> 値が <code>null</code> でない場合は <code>true</code> を返します。
<code>function isNull(value: any_type): boolean</code> 値が <code>null</code> の場合は <code>true</code> を返します。

メソッド

`function isTrueOrNull(value: boolean): boolean` 値が true または null の場合、true を返します。

- `instanceOf`
- `isFalseOrNull`
- `isNotNull`
- `isNull`
- `isTrueOrNull`

instanceOf

`instanceOf<type>(): type`

指定された型のオブジェクトインスタンスを作成します。

例:

```
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
```

関数型:

type: オブジェクト型。必須です。リテラル (文字列、数値など) を使用して作成できる型や、EBX によってのみインスタンス化できる型など、一部の型はサポートされていません。

戻り値:

新しいオブジェクト。常に指定された型の可変バージョンです。

使用可能な場所:

isFalseOrNull

function isFalseOrNull(value: boolean): boolean

値が false または null の場合は true を返します。

パラメーター:

value: ブール値。

戻り値:

値が null または false の場合は true、それ以外の場合は false。null を返すことはありません。

使用可能な場所:

isNotNull

function isNotNull(value: any_type): boolean

値が null でない場合は true を返します。

パラメーター:

value: 任意の型の値。

戻り値:

値が null でない場合は true、それ以外の場合は false。null を返すことはありません。

使用可能な場所:

isNull

function isNull(value: any_type): boolean

値が null の場合は true を返します。

パラメーター:

value: 任意の型の値。

戻り値:

値が null の場合は true、そうでない場合は false。null を返すことはありません。

使用可能な場所:

isTrueOrNull

function isTrueOrNull(value: boolean): boolean

値が true または null の場合、true を返します。

パラメーター:

value: ブール値。

戻り値:

値が null または true の場合は true、それ以外の場合は false です。null を返すことはありません。

使用可能な場所:

ユニット core.complex

複雑な値を作成および更新するためのメソッドを提供するスクリプトユニット。

メソッド
<code>foreignKeyOf<foreignKeyType>(): foreignKeyType</code> 外部キータイプのインスタンスを作成します。
<code>formatPrimaryKey(primaryKey: primaryKeyType): string</code> 主キーの内部 tableRef 文字列表現を返します。
<code>of<complexType>(): complexType</code> 複合型のインスタンスを作成します。
<code>parsePrimaryKey<primaryKeyType>(value: string): primaryKeyType</code> 文字列を主キーに解析します。
<code>primaryKeyOf<primaryKeyType>(): primaryKeyType</code> 主キー型のインスタンスを作成します。

- `foreignKeyOf`
- `formatPrimaryKey`
- `of`

- [parsePrimaryKey](#)
- [primaryKeyOf](#)

foreignKeyOf

foreignKeyOf<foreignKeyType>(): foreignKeyType

外部キータイプのインスタンスを作成します。

注意：外部キーフィールドには、参照先のテーブルの主キーが含まれます。

このメソッドは `complex.primaryKeyOf` メソッドと同等です。 `complex.primaryKeyOf` メソッドを優先してください。

例

```
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
```

関数の型

`primaryKeyType`: 主キーの型。外部キーの型または式 `foreignKeyOf` にすることができます。必須です。

戻り値

新しい外部キーオブジェクト。常に変更可能です。

使用可能な場所：

formatPrimaryKey**formatPrimaryKey(primaryKey: primaryKeyType): string**

主キーの内部 tableRef 文字列表現を返します。

関数の型

string: 主キーの内部表現。

パラメータ：

primaryKey: 主キー

戻り値

文字列が解析可能な場合は新しい可変主キーオブジェクト、それ以外の場合は null。

使用可能な場所：**of****of<complexType>(): complexType**

複合型のインスタンスを作成します。

例

```
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

```

関数の型

complexType: オブジェクト タイプ。必須です。識別子の式タイプにする必要があります。

戻り値

新しいオブジェクト。常に指定されたタイプの可変バージョンです。

使用可能な場所：

parsePrimaryKey

parsePrimaryKey<primaryKeyType>(value: string): primaryKeyType

文字列を主キーに解析します。

例

```

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

```

関数の型

primaryKeyType: 主キーの型。外部キーの型または式 primaryKeyOf にすることができます。必須です。

パラメータ：

value: 解析する主キーの内部表現を含む文字列

戻り値

文字列が解析可能な場合は新しい可変主キーオブジェクト、それ以外の場合は null。

使用可能な場所：

primaryKeyOf

primaryKeyOf<primaryKeyType>(): primaryKeyType

主キー型のインスタンスを作成します。

注意：外部キーフィールドには、参照先のテーブルの主キーが含まれます。

例

```

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

```

関数の型

primaryKeyType: 主キーの型。外部キーの型または式 primaryKeyOf にすることができます。必須です。

戻り値

新しい主キーオブジェクト。常に変更可能です。

使用可能な場所：

ユニット **core.data**

データベース操作 (挿入、変更、更新、検索) を提供するスクリプトユニット。

メソッド
<code>procedure addDataset(sql: queryType, alias: string, dataset: datasetType)</code> SQL クエリにデータセットを追加します。
<code>function createQuery(sql: string, datasets: string...):</code> SQL クエリを作成します。
<code>procedure deleteRecord(transaction: transactionType, record: recordType)</code> レコードを削除します。
<code>procedure deleteRecordByPrimarykey(transaction: transactionType, table: tableType, primaryKey: primaryKeyType)</code> 主キーで指定されたレコードを削除します。
<code>function executeQuery(query: query): list of tupleType.</code> SQL クエリを実行します。
<code>function explain(query: query): string.</code> SQL クエリプランの意味のある文字列表現を返します。
<code>function fetchFirst(query: query): tuple.</code> SQL クエリを実行し、最初のタプルを返します。
<code>function fetchOne(query: query): tuple.</code> SQL クエリを実行し、単一のタプルを返します。タプルの数が 1 でない場合は null を返します。
<code>function field<fieldType>(tuple: tupleType, alias: string): fieldType.</code> 指定されたエイリアスによる SQL タプルのフィールドを返します。
<code>function fieldByIndex<fieldType>(tuple: tupleType, index: int): fieldType.</code> 指定された位置による SQL タプルのフィールドを返します。
<code>function fieldNames(result: query_result): list<string></code> SQL クエリの結果のフィールドの名前を返します。
<code>function findDataset(dataspace: dataspaceType, name: string): datasetType</code> 名前で指定されたデータセットを返します。
<code>function findDataspace(name: string): dataspaceType</code> 名前で指定されたデータスペースを返します。

メソッド

`function findSnapshot(name: string): snapshotType` 名前で指定されたスナップショットを返します。

`function findTable<tableType>(dataset: datasetType): tableType` 指定されたデータセットのテーブルを返します。

`function lookupDataset(dataspace: dataspaceType, name: string): datasetType` 名前で指定されたデータセットを返します。

`function lookupDatasetField(dataset: datasetType): fieldType` このメソッドは非推奨であり、`lookupDatasetRoot` に置き換えられました。

`function lookupDatasetFieldForUpdate(transaction: transactionType, dataset: datasetType): updatableFieldType` このメソッドは非推奨であり、`lookupDatasetRootForUpdate` に置き換えられました。

`procedure lookupDatasetRoot<modelAlias>(dataset: datasetType)` データセットのルートを返します。

`procedure lookupDatasetRootForUpdate<modelAlias>(transaction: transactionType, dataset : datasetType)` `updatableDatasetRootType` 更新可能なデータセットのルートを返します。

`function lookupDataspace(name: string): dataspaceType` 名前で指定されたデータスペースを返します。

`function lookupRecord(table: tableType, primaryKey: PrimaryType): recordType` 主キーで指定された読み取り専用レコードを返します。

`function lookupRecordByFullXPath(table: tableType, xPath: string): recordType` XPath で指定された読み取り専用レコードを返します。

`function lookupRecordByFullXPathForUpdate(transaction: transactionType, table: tableType, xPath: string): recordType` XPath で指定された更新可能なレコードを返します。

`function lookupRecordForUpdate(transaction: transactionType, table: tableType, tableRef: string): updatableRecordType` 主キーで指定された更新可能なレコードを返します。

`function lookupRecordForUpdate(transaction: transactionType, table: tableType, primaryKey: PrimaryKeyType): updatableRecordType` 主キーで指定された更新可能なレコードを返します。

`function lookupSnapshot(name: string): snapshotType` 名前で指定されたスナップショットを返します。

メソッド

`function record<recordType>(tuple: tupleType, alias: string): record`. このメソッドは非推奨であり、`field` に置き換えられました。

`function recordByIndex<recordType>(tuple: tupleType, index: int): recordType`. このメソッドは非推奨であり、`fieldByIndex` に置き換えられました。

`function recordOf(transaction: transactionType, table: tableType): recordType` テーブルに新しいレコードを作成します。

`procedure saveDatasetRoot(root: updatableDatasetRootType)` すべてのデータセットフィールドを保存します。

`procedure saveField(field: updatableFieldType)` このメソッドは非推奨であり、`saveDatasetRoot` に置き換えられました。

`procedure saveRecord(record: updatableRecordType)` レコードを保存します。

`procedure setFetchSize(query: query, number: int)` SQL クエリから一度に取得するレコードの数を基になるデータにヒントを提供します。

`procedure setLocale(query: query, locale: locale)` SQL クエリのロケールを設定します。

`procedure setParameter(query: query, position: int, value: fieldType)` SQL クエリ内の位置パラメーターに値を設定します。

- `addDataset`
- `createQuery`
- `deleteRecord`
- `deleteRecordByPrimaryKey`
- `executeQuery`
- `explain`
- `fetchFirst`
- `fetchOne`
- `field`
- `fieldByIndex`
- `fieldNames`

- findDataset
- findDataspaces
- findSnapshot
- findTable
- lookupDataset
- lookupDatasetField
- lookupDatasetFieldForUpdate
- lookupDatasetRoot
- lookupDatasetRootForUpdate
- lookupDataspaces
- lookupRecord
- lookupRecordByFullXPath
- lookupRecordByFullXPathForUpdate
- lookupRecordByRefForUpdate
- lookupRecordForUpdate
- lookupSnapshot
- record
- recordByIndex
- recordOf
- saveDatasetRoot
- saveField
- saveRecord
- setFetchSize
- setLocale
- setParameter

addDataset

procedure addDataset(sql: queryType, alias: string, dataset: datasetType)

SQL クエリにデータセットを追加します。このメソッドは、executeQuery または explain の前に呼び出す必要があります。

例:

```
uses core.data as d;

export procedure onAfterCreate()

begin

    var sql := 'select t.colA, 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
```

パラメーター

query: SQL クエリ

alias: クエリに追加するデータセットのエイリアス

dataset: クエリに追加するデータセット

使用可能な場所：

createQuery

function createQuery(sql: string, datasets: string...):

SQL クエリを作成します。SQL 文字列が有効な選択ではない場合、たとえば、構文エラーが含まれている場合、欠落しているテーブルや欠落している列を参照している場合など、エラーが発生します。

例:

```
uses core.data as d;

export procedure onAfterCreate()

begin
```

```

var sql := 'select t.colA, t.colB from tableA t where t.id=xx';

var query := d.createQuery(sql);

end

```

パラメーター

sql: SQL クエリの文字列表現

戻り値

クエリ。null になることはありません。

使用可能な場所：

deleteRecord

procedure deleteRecord(transaction: transactionType, record: recordType)

レコードを削除します。削除が失敗した場合はエラーが発生します。

例:

```

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

```

パラメーター

transaction: トランザクション

record: レコード

使用可能な場所：

deleteRecordByPrimaryKey

procedure deleteRecordByPrimaryKey(transaction: transactionType, table: tableType, primaryKey: primaryKeyType)

主キーで指定されたレコードを削除します。削除が失敗した場合はエラーが発生します。存在しないレコードを削除してもエラーにはなりません。

例 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
```

例 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
```

パラメーター

transaction: トランザクション

table: テーブル

primaryKey: 主キー

使用可能な場所：

executeQuery

function executeQuery(query: query): list of tupleType.

SQL クエリを実行します。クエリで使用されるノードが禁止されている場合はエラーが発生します。

例:

```
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

```

パラメーター

query: SQL クエリ

戻り値

クエリの結果。null になることはありません。

使用可能な場所：

explain

function explain(query: query): string.

SQL クエリプランの意味のある文字列表現を返します。

例:

```

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

```

パラメーター

query: SQL クエリ

戻り値

クエリプランの表現

使用可能な場所：

fetchFirst

function fetchFirst(query: query): tuple.

SQL クエリを実行し、最初のタプルを返します。クエリで使用されているノードが禁止されている場合はエラーが発生します。

例:

```

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

```

パラメーター

query: SQL クエリ

戻り値

クエリの最初のタプル。クエリが結果を返さない場合は null になります。

使用可能な場所：

fetchOne

function fetchOne(query: query): tuple.

SQL クエリを実行し、単一のタプルを返します。タプルの数が 1 でない場合は null を返します。クエリで使用されているノードが禁止されている場合はエラーが発生します。

例:

```

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

```

パラメーター

query: SQL クエリ

戻り値

クエリの最初のタプル。クエリが結果を返さない場合は null になります。

使用可能な場所：

field

function field<fieldType>(tuple: tupleType, alias: string): fieldType.

指定されたエイリアスによる SQL タプルのフィールドを返します。

注: コンパイラが戻り値の型をチェックできないため、このメソッドは埋め込み SQL クエリからのタプルには推奨されません。

例:

```

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

```

関数の型

fieldType: フィールドタイプ。必須。

パラメーター

tuple: クエリ結果のタプル。

alias: フィールドのエイリアス。

使用可能な場所：

fieldByIndex

function fieldByIndex<fieldType>(tuple: tupleType, index: int): fieldType.

指定された位置による SQL タプルのフィールドを返します。

注: コンパイラが戻り値の型をチェックできないため、このメソッドは埋め込み SQL クエリからのタプルには推奨されません。

例:

```

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
end

```

関数の型

fieldType: フィールドタイプ。必須。

パラメーター

tuple: クエリ結果のタプル。

index: フィールドの位置。最初の位置は 0 です。

使用可能な場所：

fieldNames

function fieldNames(result: query_result): list<string>

SQL クエリの結果のフィールドの名前を返します。

例:

```
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

end
```

パラメーター

result: SQL クエリ

戻り値

クエリ結果のフィールドの名前。null になることはありません。

使用可能な場所：

findDataset

function findDataset(dataspace: dataspaceType, name: string): datasetType

名前で指定されたデータセットを返します。データセットが見つからない場合はエラーが発生します。

```

uses core.data as d;

procedure doSomething()

begin

    var dataspace := d.findDataspace('dataspace-name');

    var dataset := d.findDataset(dataspace, 'dataset-name');

    ...

end

```

パラメーター

dataspace: データセットを所有するデータスペース。

name: データセットの名前。

戻り値

名前で指定されたデータセット。null になることはありません。

使用可能な場所：

findDataspace

function findDataspace(name: string): dataspaceType

名前で指定されたデータスペースを返します。データスペースが見つからない場合はエラーが発生します。

例:

```

uses core.data as d;

begin

    var dataspace := d.findDataspace('dataspace-name');

```

```
...
end
```

パラメーター

name: データスペースの名前。

戻り値

データスペース。null になることはありません。

使用可能な場所：

findSnapshot

function findSnapshot(name: string): snapshotType

名前で指定されたスナップショットを返します。スナップショットが見つからない場合はエラーが発生します。

例:

```
uses core.data as d;

procedure doSomething()

begin

    var snapshot := d.findSnapshot('snapshot-name');

    ...

end
```

パラメーター

name: スナップショットの名前。

戻り値

スナップショット。null になることはありません。

使用可能な場所：

findTable

function findTable<tableType>(dataset: datasetType): tableType

指定されたデータセットのテーブルを返します。テーブルが見つからない場合はエラーが発生します。

例:

```
uses core.data as d;

procedure doSomething()

begin

    var customerTable := db.findTable<.Customer>(_ebx.dataset);

    ...

end
```

関数の型

complexType: テーブルの種類。

パラメーター

dataset: テーブルを所有するデータセット。

戻り値

テーブル。null になることはありません。

使用可能な場所：

lookupDataset

function lookupDataset(dataspace: dataspaceType, name: string): datasetType

名前で指定されたデータセットを返します。データセットが見つからない場合は null を返します。

```
uses core.data as d;

procedure doSomething()

begin

    var dataspace := d.lookupDataspace('dataspace-name');

    var dataset := d.lookupDataset(dataspace, 'dataset-name');

    ...

end
```

パラメーター

dataspace: データセットを所有するデータスペース。

name: データセットの名前。

戻り値

名前で指定されたデータセット。見つからない場合は null になります。

使用可能な場所：

lookupDatasetField

function lookupDatasetField(dataset: datasetType): fieldType

このメソッドは非推奨であり、lookupDatasetRoot に置き換えられました。

データセットによって指定された読み取り専用フィールドを返します。

パラメーター

dataset: データセット。

戻り値

読み取り専用フィールド。

使用可能な場所：

lookupDatasetFieldForUpdate

**function lookupDatasetFieldForUpdate(transaction: transactionType, dataset: datasetType):
updatableFieldType**

このメソッドは非推奨であり、lookupDatasetRootForUpdate に置き換えられました。

指定されたデータセットの更新可能なフィールドを返します。データセットがトランザクションの作成に使用されたデータスペースによって所有されていない場合はエラーが発生します。

注: 現在、ルート直下の単純なデータセットフィールドにアクセスすることはできません。

パラメーター

transaction: トランザクション

dataset: データセットは、トランザクションの作成に使用されるデータスペースによって所有されている必要があります。

戻り値

更新可能なフィールド。

使用可能な場所：

lookupDatasetRoot

procedure lookupDatasetRoot<modelAlias>(dataset: datasetType)

データセットのルートを返します。操作が失敗した場合はエラーが発生します。

例:

```
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

```

パラメーター

dataset: データセット。

戻り値

データセットのルート。null になることはありません。

使用可能な場所：

lookupDatasetRootForUpdate

**procedure lookupDatasetRootForUpdate<modelAlias>(transaction: transactionType,
dataset : datasetType) updatableDatasetRootType**

更新可能なデータセットのルートを返します。操作が失敗した場合はエラーが発生します。

例:

```

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

```

パラメーター

transaction: トランザクション。

dataset: データセット。

戻り値

更新可能なデータセットのルート。null になることはありません。

使用可能な場所：

lookupDataspace

function lookupDataspace(name: string): dataspaceType

名前で指定されたデータスペースを返します。データスペースが見つからない場合は null を返します。

例:

```
uses core.data as d;

procedure doSomething()

begin

    var dataspace := d.lookupDataspace('dataspace-name');

    ...

end
```

パラメーター

name: データスペースの名前。

戻り値

データスペース。見つからない場合は null になります。

使用可能な場所：

lookupRecord

function lookupRecord(table: tableType, primaryKey: PrimaryType): recordType

主キーで指定された読み取り専用レコードを返します。レコードが存在しない場合は null を返します。

例:

```
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

```

パラメーター

table: テーブル

primaryKey: 主キー

戻り値

主キーで指定されたレコード。見つからない場合は null になります。

使用可能な場所：

lookupRecordByFullXPath

function lookupRecordByFullXPath(table: tableType, xpath: string): recordType

XPath で指定された読み取り専用レコードを返します。レコードが存在しない場合は null を返します。

XPath では、各主キーフィールドの値を指定する必要があります。

XPath の例は次のとおりです。

```
/root/table[./pk-field1='a' and ./pk-field2='b']
```

例:

```

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

```

パラメーター

table: テーブル

xPath: XPath 式。

戻り値

xPath で指定されたレコード。見つからない場合は null になります。

使用可能な場所：

lookupRecordByFullXPathForUpdate

function lookupRecordByFullXPathForUpdate(transaction:transactionType,table: tableType, xpath: string): recordType

XPath で指定された更新可能なレコードを返します。レコードが存在しない場合は null を返します。

XPath では、各主キーフィールドの値を指定する必要があります。

XPath の例は次のとおりです。

```
/root/table[./pk-field1='a' and ./pk-field2='b']
```

例:

```
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
```

パラメーター

transaction: トランザクション

table: テーブル

xPath: XPath 式。

戻り値

xPath で指定されたレコード。見つからない場合は null になります。

使用可能な場所：

lookupRecordByRefForUpdate

function lookupRecordForUpdate(transaction: transactionType, table: tableType, tableRef: string): updatableRecordType

主キーで指定された更新可能なレコードを返します。レコードが存在しない場合、または tableRef が有効な形式でない場合は null を返します。

パラメーター

transaction: トランザクション。

table: テーブル。

tableRef: テーブル参照。

戻り値

tableRef で指定された更新可能なレコード。見つからない場合、または tableRef が有効な形式でない場合は null になります。

使用可能な場所：

lookupRecordForUpdate

function lookupRecordForUpdate(transaction: transactionType, table: tableType, primaryKey: PrimaryKeyType): updatableRecordType

主キーで指定された更新可能なレコードを返します。レコードが存在しない場合は null を返します。

例:

```

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

```

パラメーター

transaction: トランザクション

table: テーブル

primaryKey: 主キー

戻り値

主キーで指定された更新可能なレコード。見つからない場合は null になります。

使用可能な場所：

lookupSnapshot

function lookupSnapshot(name: string): snapshotType

名前で指定されたスナップショットを返します。指定されたスナップショットが見つからない場合は null を返します。

例:

```

uses core.data as d;

procedure doSomething()

begin

    var snapshot := d.lookupSnapshot('snapshot-name');

    ...

end

```

パラメーター

name: スナップショットの名前。

戻り値

スナップショット。見つからない場合は null になります。

使用可能な場所：

record

function record<recordType>(tuple: tupleType, alias: string): record.

このメソッドは非推奨であり、field に置き換えられました。

指定されたエイリアスによる読み取り専用レコードを返します。

関数の型

recordType: レコードタイプ。必須。

パラメーター

tuple: クエリ結果のタプル。

alias: フィールドのエイリアス。

使用可能な場所：

recordByIndex

function recordByIndex<recordType>(tuple: tupleType, index: int): recordType.

このメソッドは非推奨であり、fieldByIndex に置き換えられました。

インデックス位置にある読み取り専用レコードを返します。

関数の型

complexType: レコードの種類。必須。

パラメーター

tuple: クエリ結果のタプル。

index: フィールドの位置。

使用可能な場所：

recordOf

function recordOf(transaction: transactionType, table: tableType): recordType

テーブルに新しいレコードを作成します。デフォルト値はスキーマで指定されたとおりに設定されます。

例:

```
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

```

パラメーター

transaction: トランザクション

table: テーブル

戻り値

新しいレコード。null になることはありません。

使用可能な場所：

saveDatasetRoot

procedure saveDatasetRoot(root: updatableDatasetRootType)

すべてのデータセットフィールドを保存します。操作が失敗した場合はエラーが発生します。

例:

```

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

```

パラメーター

root: 保存するデータセットのルート。

使用可能な場所：

saveField

procedure saveField(field: updatableFieldType)

このメソッドは非推奨であり、saveDatasetRoot に置き換えられました。

データセットフィールドを保存します。操作が失敗した場合はエラーが発生します。

references model "myModel" as m;

パラメーター

field: 保存するデータセットフィールド。

使用可能な場所：

saveRecord

procedure saveRecord(record: updatableRecordType)

レコードを保存します。操作が失敗した場合はエラーが発生します。

レコードは新しい値で更新されます。例:

```
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
```

パラメーター

record: 保存するレコード。

使用可能な場所：

setFetchSize

procedure setFetchSize(query: query, number: int)

SQL クエリから一度に取得するレコードの数を基になるデータにヒントを提供します。

例:

```

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

```

パラメーター

query: クエリ。

number: 取得するレコードの数。

使用可能な場所：

setLocale

procedure setLocale(query: query, locale: locale)

SQL クエリのロケールを設定します。executeQuery メソッドの前に呼び出す必要があります。

例:

```

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'));

end

```

```

    var queryResult := d.executeQuery(query);

    ...

end

```

パラメーター

query: クエリ。

locale: ロケール。

使用可能な場所：

setParameter

procedure setParameter(query: query, position: int, value: fieldType)

SQL クエリ内の位置パラメーターに値を設定します。例:

```

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

```

パラメーター

query: クエリ。

position: クエリ内のパラメーターの位置。

value: パラメーターの値。

使用可能な場所：

ユニット **core.date**

日付関数を提供するスクリプトユニット。

メソッド
<code>function addDays(value: date, days: decimal): date</code> 日付に日を追加します
<code>function addMonths(value: date, months: decimal): datetime</code> 日付に月を追加します
<code>function addYears(value: date, years: decimal): date</code> 日付に年を追加します
<code>function days(startValue: date, endValue: date): decimal</code> 2 つの日付の間の日数を返します
<code>function fromDatetime(value: datetime): date</code> 日付時刻を日付に変換します
<code>function fromIsoString(value: string): date</code> ISO 形式 (yyyy-MM-dd) の文字列から日付を解析します
<code>function months(startValue: date, endValue: date): decimal</code> 2 つの日付間の月数を返します
<code>function now(): date</code> 現在の日付を返します
<code>function dateOf(year: decimal, month: decimal, day: decimal): date</code> 年、月、日を指定して日付を作成します
<code>function toIsoString(value: date): string</code> 日付を ISO 形式 (yyyy-MM-dd) の文字列に変換します
<code>function years(startValue: date, endValue: date): decimal</code> 2 つの日付間の年数を返します

- `addDays`
- `addMonths`
- `addYears`
- `days`
- `fromDatetime`

- fromIsoString
- months
- now
- of
- toIsoString
- years

addDays

function addDays(value: date, days: decimal): date

日付に日を追加します

パラメーター：

value: 入力日付

days: 追加される日数。整数にする必要があります

戻り値：

新しい日付。パラメーターが null の場合は null

使用可能な場所：

addMonths

function addMonths(value: date, months: decimal): datetime

日付に月を追加します

パラメーター：

value: 入力日付

months: 追加される月数。整数にする必要があります

戻り値：

新しい日付。パラメーターが null の場合は null

使用可能な場所：

addYears

function addYears(value: date, years: decimal): date

日付に年を追加します

パラメーター：

value: 入力日付

years: 追加される年数。整数にする必要があります

戻り値：

新しい日付。パラメーターが null の場合は null

使用可能な場所：

days

function days(startValue: date, endValue: date): decimal

2 つの日付の間の日数を返します

パラメーター：

startValue: 入力開始日

endValue: 入力終了日

戻り値：

日数。パラメーターが null の場合は null

使用可能な場所：

fromDatetime

function fromDatetime(value: datetime): date

日付と時刻を日付に変換します。入力の時間部分は無視されます

パラメーター：

value: 入力日時

戻り値：

新しい日付。パラメーターが null の場合は null

使用可能な場所：

fromIsoString

function fromIsoString(value: string): date

ISO 形式 (yyyy-MM-dd) の文字列から日付を解析します

パラメーター：

value: 入力文字列

戻り値：

日付。値が null であるか、形式が無効な場合は null

使用可能な場所：

months

function months(startValue: date, endValue: date): decimal

2 つの日付間の月数を返します

パラメーター：

startValue: 入力開始日

endValue: 入力終了日

戻り値：

月数、またはいずれかのパラメーターが null の場合は null

使用可能な場所：

now

function now(): date

現在の日付を返します

戻り値：

現在の日付

使用可能な場所：

of

function dateOf(year: decimal, month: decimal, day: decimal): date

年、月、日を指定して日付を作成します

パラメーター：

year: 入力年

month: 入力月

day: 入力日

戻り値：

新しい日付。パラメーターが null の場合は null

使用可能な場所：

toIsoString

function toIsoString(value: date): string

日付を ISO 形式 (yyyy-MM-dd) の文字列に変換します

パラメーター：

value: 入力日付

戻り値：

文字列。値が null の場合は null

使用可能な場所：

years

function years(startValue: date, endValue: date): decimal

2 つの日付間の年数を返します

パラメーター：

startValue: 入力開始日

endValue: 入力終了日

戻り値：

年数、またはいずれかのパラメーターが null の場合は null

使用可能な場所：

ユニット core.datetime

日時関数を提供するスクリプトユニットです。

メソッド
function addDays(value: datetime, days: decimal): datetime 日付と時刻に日を追加します。
function addHours(value: datetime, hours: decimal): datetime 日付と時刻に時間を追加します。
function addMinutes(value: datetime, minutes: decimal): datetime 日付と時刻に分を追加します。
function addMonths(value: datetime, months: decimal): datetime 日付と時刻に月を追加します。
function addSeconds(value: datetime, seconds: decimal): datetime 日付と時刻に秒を追加します。
function addYears(value: datetime, years: decimal): datetime 日付と時刻に年を追加します。
function days(startValue: datetime, endValue: datetime): decimal 2 つの日付と時刻の間の日数を返します。
function fromDate(value: date): datetime 時刻を 00:00:00 に設定して、日付を日付と時刻に変換します。
function fromDateAndTime(dateValue: date, timeValue: time): datetime 日付と時刻の値を指定して日付と時刻を作成します。
function fromIsoString(value: string): date ISO 形式 (yyyy-MM-dd'T'HH:mm:ss) の文字列から日時を解析します。
function months(startValue: date, endValue: date): decimal 2 つの日付と時刻の間の月数を返します。
function now(): datetime 現在の日付と時刻を返します。
function of(year: decimal, month: decimal, day: decimal, hour: decimal, minute: decimal, second: decimal): datetime 年、月、日、時、分、秒を指定して日付と時刻を作成します。

メソッド

`function seconds(startValue: datetime, endValue: datetime): decimal` 2 つの日付と時刻の間の秒数を返します。

`function toIsoString(value: date): string` 日時を ISO 形式 (yyyy-MM-dd'T'HH:mm:ss) の文字列に変換します。

`function years(startValue: date, endValue: date): decimal` 2 つの日付と時刻の間の年数を返します。

- `addDays`
- `addHours`
- `addMinutes`
- `addMonths`
- `addSeconds`
- `addYears`
- `days`
- `fromDate`
- `fromDateAndTime`
- `fromIsoString`
- `months`
- `now`
- `of`
- `seconds`
- `toIsoString`
- `years`

addDays

`function addDays(value: datetime, days: decimal): datetime`

日付と時刻に日を追加します。

パラメーター

value: 入力日時

days: 追加される日数。整数にする必要があります。

戻り値

新しい日時。パラメーターが null の場合は null。

使用可能な場所：

addHours

function addHours(value: datetime, hours: decimal): datetime

日付と時刻に時間を追加します。

パラメーター

value: 入力日時

hours: 追加される時間数。整数にする必要があります。

戻り値

新しい日時。パラメーターが null の場合は null。

使用可能な場所：

addMinutes

function addMinutes(value: datetime, minutes: decimal): datetime

日付と時刻に分を追加します。

パラメーター

value: 入力日時

minutes: 追加する分数。整数にする必要があります。

戻り値

新しい日時。パラメーターが null の場合は null。

使用可能な場所：

addMonths

function addMonths(value: datetime, months: decimal): datetime

日付と時刻に月を追加します。

パラメーター

value: 入力日時

months: 追加される月数。整数にする必要があります。

戻り値

新しい日時。パラメーターが null の場合は null。

使用可能な場所：

addSeconds

function addSeconds(value: datetime, seconds: decimal): datetime

日付と時刻に秒を追加します。秒の小数部はミリ秒の精度でサポートされています。

パラメーター

value: 入力日時

seconds: 追加される秒数。小数部は小数点以下 3 桁に丸められます。

戻り値

新しい日時。パラメーターが null の場合は null。

使用可能な場所：

addYears

function addYears(value: datetime, years: decimal): datetime

日付と時刻に年を追加します。

パラメーター

value: 入力日時

years: 追加される年数。整数にする必要があります。

戻り値

新しい日時。パラメーターが null の場合は null。

使用可能な場所：

days

function days(startValue: datetime, endValue: datetime): decimal

2 つの日付と時刻の間の日数を返します。この関数は、days(toDate(startValue), toDate(endValue)) と同等です。

パラメーター

startValue: 入力開始日時

endValue: 入力終了日時

戻り値

日数。パラメーターが null の場合は null。

使用可能な場所：

fromDate

function fromDate(value: date): datetime

時刻を 00:00:00 に設定して、日付を日付と時刻に変換します。

パラメーター

value: 入力日付

戻り値

新しい日時。パラメーターが null の場合は null。

使用可能な場所：

fromDateAndTime

function fromDateAndTime(dateValue: date, timeValue: time): datetime

日付と時刻の値を指定して日付と時刻を作成します。

パラメーター

dateValue: 入力日

timeValue: 入力時間

戻り値

新しい日時。パラメーターが null の場合は null。

使用可能な場所：

fromIsoString

function fromIsoString(value: string): date

ISO 形式 (yyyy-MM-dd'T'HH:mm:ss.SSS) の文字列から日時を解析します。ミリ秒部分はオプションです。

パラメーター

value: 入力文字列

戻り値

日付。値が null の場合、または形式が無効な場合は null。

使用可能な場所：

months

function months(startValue: date, endValue: date): decimal

2 つの日付と時刻の間の月数を返します。

パラメーター

startValue: 入力開始日時

endValue: 入力終了日時

戻り値

月数、またはいずれかのパラメーターが null の場合は null。

使用可能な場所：

now

function now(): datetime

現在の日付と時刻を返します。

戻り値

現在の日付と時刻。

使用可能な場所：

of

function of(year: decimal, month: decimal, day: decimal, hour: decimal, minute: decimal, second: decimal): datetime

年、月、日、時、分、秒を指定して日付と時刻を作成します。秒の小数部はミリ秒の精度でサポートされています。

パラメーター

year: 入力年

month: 入力月

day: 入力日

hour: 入力時間

minute: 入力分

second: 入力秒。小数部は小数点以下 3 桁に丸められます。

戻り値

新しい日時。パラメーターが null の場合は null。

使用可能な場所：

seconds

function seconds(startValue: datetime, endValue: datetime): decimal

2 つの日付と時刻の間の秒数を返します。秒の小数部はミリ秒の精度でサポートされています。

パラメーター

startValue: 入力開始日時 (含む)

endValue: 入力終了日時 (除く)

戻り値

秒数。パラメーターが null の場合は null。

使用可能な場所：

tolsoString

function toIsoString(value: date): string

日時を ISO 形式 (yyyy-MM-dd'T'HH:mm:ss.SSS) の文字列に変換します。

パラメーター

value: 入力日時

戻り値

文字列。値が null の場合は null。

使用可能な場所：

years

function years(startValue: date, endValue: date): decimal

2 つの日付と時刻の間の完全な年数を返します。

パラメーター

startValue: 入力開始日時

endValue: 入力終了日時

戻り値

年数、またはいずれかのパラメーターが null の場合は null。

使用可能な場所：

ユニット core.dictionary

辞書を作成および操作するメソッドを提供するスクリプトユニットです。辞書は、文字列キーと指定された型の値を持つマップです。

メソッド
<code>procedure clear(dict: dict)</code> 辞書からすべてのマッピングを削除します。
<code>function clone(dict: dict): dict</code> 辞書のシャロークローンを返します。

メソッド

`function containsKey(dict: dict, key: string): boolean` 指定されたキーのマッピングが辞書に含まれている場合に true を返します。

`function containsValue(dict: dict, value: T): boolean` 辞書が指定された値に 1 つ以上のキーをマッピングしている場合に true を返します。

`function equals(dict1: dict, dict2: dict): boolean` 2 つの辞書が等しいかどうかを比較します。

`function get(dict: dict, key: string): T` 指定されたキーがマッピングされている値を返します。辞書にそのキーのマッピングが含まれていない場合は null を返します。

`function getOrDefault(dict: dict, key: string, defaultValue: T): T` 指定されたキーがマッピングされている値を返します。辞書にそのキーのマッピングが含まれていない場合はデフォルト値を返します。

`function isEmpty(dict: dict): boolean` 辞書が null または空の場合に true を返します。

`function keys(dict: dict): set` 辞書に含まれるキーのセットビューを返します。

`function of(key1: string, value1: T [, key2: string, value2: T]...): dict` 指定されたキーと値のペアを持つ辞書を作成します。

`procedure put(dict: dict, key: string, value: T)` 辞書内で指定された値を指定されたキーに関連付けます。

`procedure putAll(dest: dict, src: dict)` ソース辞書から宛先辞書にすべてのマッピングをコピーします。

`function remove(dict: dict, key: string): T` キーのマッピングが存在する場合、辞書からそのマッピングを削除します。

`function size(dict: dict): integer` 辞書内のキーと値のマッピングの数を返します。

`function values(dict: dict): collection` 辞書に含まれる値のコレクションビューを返します。

- `clear`
- `clone`
- `containsKey`
- `containsValue`
- `equals`

- `get`
- `getOrDefault`
- `isEmpty`
- `keys`
- `of`
- `put`
- `putAll`
- `remove`
- `size`
- `values`

clear

procedure clear(dict: dict)

辞書からすべてのマッピングを削除します。

パラメーター:

dict: クリアする辞書。

使用可能箇所:

clone

function clone(dict: dict): dict

辞書のシャロークローンを返します。

パラメーター:

dict: クローンする辞書。

戻り値:

入力辞書と同じエントリを持つ新しい辞書。

使用可能箇所:

containsKey

function containsKey(dict: dict, key: string): boolean

指定されたキーのマッピングが辞書に含まれている場合に true を返します。

パラメーター:

dict: 辞書。

key: 辞書内に存在するかどうかをテストするキー。

戻り値:

指定されたキーのマッピングが辞書に含まれている場合は true、それ以外の場合は false。辞書が null の場合は null を返します。

使用可能箇所:

containsValue

function containsValue(dict: dict, value: T): boolean

辞書が指定された値に 1 つ以上のキーをマッピングしている場合に true を返します。

パラメーター:

dict: 辞書。

value: 辞書内に存在するかどうかをテストする値。

戻り値:

辞書が指定された値に 1 つ以上のキーをマッピングしている場合は true、それ以外の場合は false。辞書が null の場合は null を返します。

使用可能箇所:

equals

function equals(dict1: dict, dict2: dict): boolean

2 つの辞書が等しいかどうかを比較します。

パラメーター:

dict1: 最初の辞書。

dict2: 2 番目の辞書。

戻り値:

両方の辞書が同じキーと値を持つ場合は true、それ以外の場合は false。

使用可能箇所:

get

function get(dict: dict, key: string): T

指定されたキーがマッピングされている値を返します。辞書にそのキーのマッピングが含まれていない場合は null を返します。

パラメーター:

dict: 辞書。

key: 関連付けられた値が返されるキー。

戻り値:

指定されたキーがマッピングされている値。辞書にそのキーのマッピングが含まれていない場合は null。辞書が null の場合は null を返します。

使用可能箇所:

getOrDefault

function getOrDefault(dict: dict, key: string, defaultValue: T): T

指定されたキーがマッピングされている値を返します。辞書にそのキーのマッピングが含まれていない場合はデフォルト値を返します。

パラメーター:

dict: 辞書。

key: 関連付けられた値が返されるキー。

defaultValue: キーが見つからない場合に返すデフォルト値。

戻り値:

指定されたキーがマッピングされている値。辞書にそのキーのマッピングが含まれていない場合はデフォルト値。辞書が null の場合はデフォルト値を返します。

使用可能箇所:

isEmpty

function isEmpty(dict: dict): boolean

辞書が null または空の場合に true を返します。

パラメーター:

dict: 入力辞書。

戻り値:

辞書が null または空の場合は true、それ以外の場合は false。

使用可能箇所:

keys

function keys(dict: dict): set

辞書に含まれるキーのセットビューを返します。

パラメーター:

dict: 辞書。

戻り値:

辞書に含まれるキーのセットビュー。辞書が null の場合は空のセットを返します。

使用可能箇所:

of

function of(key1: string, value1: T ...): dict

指定されたキーと値のペアを持つ辞書を作成します。

パラメーター:

keysAndValues: キー (文字列) と値 (型 T) の交互のシーケンス。

戻り値:

指定されたキーと値のペアを含む新しい辞書。

使用可能箇所:

put**procedure put(dict: dict, key: string, value: T)**

辞書内で指定された値を指定されたキーに関連付けます。

パラメーター:

dict: 辞書。

key: 指定された値が関連付けられるキー。

value: 指定されたキーに関連付けられる値。

使用可能箇所:**putAll****procedure putAll(dest: dict, src: dict)**

ソース辞書から宛先辞書にすべてのマッピングをコピーします。

パラメーター:

dest: 宛先辞書。

src: ソース辞書。

使用可能箇所:**remove****function remove(dict: dict, key: string): T**

キーのマッピングが存在する場合、辞書からそのマッピングを削除します。

パラメーター:

dict: 辞書。

key: 辞書から削除されるマッピングのキー。

戻り値:

キーに関連付けられていた以前の値。キーのマッピングがなかった場合は null。

使用可能箇所:

size**function size(dict: dict): integer**

辞書内のキーと値のマッピングの数を返します。

パラメーター:

dict: 入力辞書。

戻り値:

辞書内のキーと値のマッピングの数。辞書が null の場合は 0。

使用可能箇所:

values**function values(dict: dict): collection**

辞書に含まれる値のコレクションビューを返します。

パラメーター:

dict: 辞書。

戻り値:

辞書に含まれる値のコレクションビュー。辞書が null の場合は空のコレクションを返します。

使用可能箇所:

ユニット core.http

外部RESTサービスを呼び出すメソッドを提供するスクリプトユニットです。

メソッド
<code>procedure addParameter(request: http_request, parameter: string, value: string)</code> リクエストにクエリパラメーターを追加します。
<code>procedure addPath(request: http_request, path: string)</code> リクエストのURLにパスを追加します。
<code>procedure addSegment(request: http_request, segment: string)</code> リクエストのURLにセグメントを追加します。
<code>function createRequest(id: string): http_request</code> 構成IDで指定されたリクエストを作成します。

メソッド

`function delete(request: http\_request): http\_response` DELETEリクエストを送信し、レスポンスを処理します。

`function get(request: http\_request): http\_response` GETリクエストを送信し、レスポンスを処理します。

`function getConfigNames(request: http\_request): list<string>` REST構成で指定されたキー名を取得します。

`function getConfigValue(request: http\_request, name: string): string` REST構成で指定された特定のキー名の値を取得します。

`function getHeaderFirstValue(response: http\_response, headerName: string): string` 指定された名前のヘッダーの最初のレスポンスヘッダー値を返します。

`function getHeaderNames(response: http\_response): list<string>` レスポンスヘッダー名を返します。

`function getHeaderValues(response: http\_response, headerName: string): list<string>` 指定された名前のヘッダーのレスポンスヘッダー値を返します。

`function getJsonBody(response: http\_response): json\_value` ボディレスポンスのJSON値を返します。

`function getRequestUrl(request: http\_request): string` リクエストのURLを取得します。

`function getResponseUrl(response: http\_response): string` レスポンスのURLを取得します。

`function getStatusCode(response: http\_response): int` レスポンスのステータスコードを返します。

`function getStringBody(response: http\_response): string` レスポンスの文字列ボディを返します。

`function hasBody(response: http\_response): boolean` レスポンスにボディがある場合、trueを返します。

`function post(request: http\_request, body: string): http\_response` 指定されたボディを設定してPOSTリクエストを送信し、レスポンスを処理します。

`function put(request: http\_request, body: string): http\_response` 指定されたボディを設定してPUTリクエストを送信し、レスポンスを処理します。

メソッド

procedure setHeader(request: http_request, headerField: string, value: string) リクエストのヘッダーフィールドに値を設定します。

procedure setTimeout(request: http_request, timeout: int) リクエストにミリ秒単位でタイムアウトを設定します。

- addParameter
- addPath
- addSegment
- createRequest
- delete
- get
- getConfigNames
- getConfigValue
- getHeaderFirstValue
- getHeaderNames
- getHeaderValues
- getJsonBody
- getRequestUrl
- getResponseUrl
- getStatusCode
- getStringBody
- hasBody
- post
- put
- setHeader
- setTimeout

addParameter

procedure addParameter(request: http_request, parameter: string, value: string)

リクエストにクエリパラメーターを追加します。パラメーター値はパーセンテージエンコードされます。

例:

```
uses core.http as client;

...

begin

    var request := client.createRequest('ebx-1');

    client.addPath(request, 'book');

    client.addParameter(request, 'status', 'sold');

    ...

end
```

パラメーター:

request: リクエスト

parameter: クエリパラメーター

value: パラメーター値

使用可能な場所:

addPath

procedure addPath(request: http_request, path: string)

リクエストのURLにパスを追加します。このメソッドは指定されたパスをエンコードしないことに注意してください。

例:

```
uses core.http as client;

...
```

```

begin

    var request := client.createRequest('ebx-1');

    client.addPath(request, 'book/findByStatus');

    ...

end

```

パラメーター:

request: リクエスト

path: 追加するパス

使用可能な場所:**addSegment**

procedure addSegment(request: http_request, segment: string)

リクエストのURLにセグメントを追加します。指定されたセグメントはパーセンテージエンコードされることに注意してください。

例:

```

uses core.http as client;

...

begin

    var request := client.createRequest('ebx-1');

    client.addSegment(request, 'search by term');

    ...

end

```

パラメーター:

request: リクエスト

segment: 追加するセグメント

使用可能な場所:

createRequest

function createRequest(id: string): http_request

構成IDで指定されたリクエストを作成します。構成IDが見つからない場合、エラーが発生します。IDはEBX管理の構成テーブルで指定されます。

例:

```
uses core.http as client;

...

begin

    var request := client.createRequest('ebx-1');

    ...

end
```

パラメーター:

id: 構成のID

戻り値:

リクエスト

使用可能な場所:

delete

function delete(request: http_request): http_response

DELETEリクエストを送信し、レスポンスを処理します。I/Oエラーが発生した場合、またはリクエストがタイムアウトした場合、エラーが発生します。返されるレスポンスには、ステータス、ヘッダー、および場合によってはボディが含まれます。

例:

```
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
```

パラメーター:

request: リクエスト

戻り値:

受信したレスポンス

使用可能な場所:

get

function get(request: http_request): http_response

GETリクエストを送信し、レスポンスを処理します。I/Oエラーが発生した場合、またはリクエストがタイムアウトした場合、エラーが発生します。返されるレスポンスには、ステータス、ヘッダー、およびボディが含まれます。

例:

```
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

```

パラメーター:

request: リクエスト

戻り値:

受信したレスポンス

使用可能な場所:**getConfigNames**

function getConfigNames(request: http_request): list<string>

REST構成で指定されたキー名を取得します。キー名が見つからない場合、空のリストを返します。

例:

```

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
```

パラメーター:

request: リクエスト

戻り値:

文字列値

使用可能な場所:

getConfigValue

function getConfigValue(request: http_request, name: string): string

REST構成で指定された特定のキー名の値を取得します。シークレットキーの値は常に復号化されます。キー名が見つからない場合、nullを返します。指定されたキーの値が復号化できない場合、エラーが発生します。

例:

```
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
```

```
end
```

パラメーター:

request: リクエスト

戻り値:

文字列値

使用可能な場所:**getHeaderFirstValue**

function getHeaderFirstValue(response: http_response, headerName: string): string

指定された名前のヘッダーの最初のレスポンスヘッダー値を返します。ヘッダー名が見つからない場合、nullを返します。

例:

```
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
```

パラメーター:

response: レスポンス

headerName: ヘッダー名

戻り値:

ヘッダー値

使用可能な場所:

getHeaderNames

function getHeaderNames(response: http_response): list<string>

レスポンスヘッダー名を返します。名前付きヘッダーが見つからない場合、空のリストを返します。

例:

```
uses core.http as client;

...

begin

    var request := client.createRequest('ebx-1');

    var response := client.get(request);

    var headerNames := client.getHeaderNames(response);

    ...

end
```

パラメーター:

response: レスポンス

戻り値:

ヘッダー名のリスト

使用可能な場所:

getHeaderValues

function getHeaderValues(response: http_response, headerName: string): list<string>

指定された名前のヘッダーのレスポンスヘッダー値を返します。名前が見つからない場合、空のリストを返します。

例:

```
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
```

パラメーター:

response: レスポンス

headerName: ヘッダー名

戻り値:

ヘッダー値のリスト

使用可能な場所:

getJSONBody

function getJSONBody(response: http_response): json_value

ボディレスポンスのJSON値を返します。レスポンスにボディがない場合、nullを返します。

例:

```

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

end

```

パラメーター:

response: レスポンス

戻り値:

ボディレスポンスのJSON値。レスポンスのボディが有効なJSONでない場合、またはnullの場合、nullを返します。

使用可能な場所:**getRequestUrl**

function getRequestUrl(request: http_request): string

リクエストのURLを取得します。

例:


```

uses core.http as client;

...

begin

    var request := client.createRequest('ebx-1');

    client.addPath(request, 'author');

    var url:=client.getRequestUrl(response);

end

```

パラメーター:

request: リクエスト

戻り値:

URL

使用可能な場所:

getResponseUrl

function getResponseUrl(response: http_response): string

レスポンスのURLを取得します。

例:

```

uses core.http as client;

...

begin

    var request := client.createRequest('ebx-1');

    client.addPath(request, 'publisher');

    var response := client.get(request);

end

```

```

    var url:=client.getResponseUrl(response);

end

```

パラメーター:

response: レスポンス

戻り値:

URL

使用可能な場所:

getStatusCode

function getStatusCode(response: http_response): int

レスポンスのステータスコードを返します。

例:

```

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

```

```
end
```

パラメーター:

response: レスポンス

戻り値:

ステータスコード

使用可能な場所:

getStringBody

function getStringBody(response: http_response): string

レスポンスの文字列ボディを返します。レスポンスにボディがない場合、nullを返します。

例:

```
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

end
```

パラメーター:

response: レスポンス

戻り値:

レスポンスの文字列ボディ

使用可能な場所:

hasBody

function hasBody(response: http_response): boolean

レスポンスにボディがある場合、trueを返します。

例:

```
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

end
```

パラメーター:

response: レスポンス

戻り値:

レスポンスボディがある場合はtrue、それ以外の場合はfalse

使用可能な場所:

post

function post(request: http_request, body: string): http_response

指定されたボディを設定してPOSTリクエストを送信し、レスポンスを処理します。I/Oエラーが発生した場合、またはリクエストがタイムアウトした場合、エラーが発生します。返されるレスポンスには、ステータス、ヘッダー、および場合によってはボディが含まれます。

例:

```
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
```

パラメーター:

request: リクエスト

body: ボディ

戻り値:

受信したレスポンス

使用可能な場所:

put

function put(request: http_request, body: string): http_response

指定されたボディを設定してPUTリクエストを送信し、レスポンスを処理します。I/Oエラーが発生した場合、またはリクエストがタイムアウトした場合、エラーが発生します。返されるレスポンスには、ステータス、ヘッダー、および場合によってはボディが含まれます。

例:

```
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
```

パラメーター:

request: リクエスト

body: ボディ

戻り値:

受信したレスポンス

使用可能な場所:

setHeader

procedure setHeader(request: http_request, headerField: string, value: string)

リクエストのヘッダーフィールドに値を設定します。

例:

```
uses core.http as client;

...

begin

    var request := client.createRequest('ebx-1');

    client.setHeader(request, 'Content-Type', 'application/json');

    ...

end
```

パラメーター:

request: リクエスト

headerField: ヘッダーフィールド

value: 値

使用可能な場所:

setTimeout

procedure setTimeout(request: http_request, timeout: int)

リクエストにミリ秒単位でタイムアウトを設定します。ebxプロパティファイルで定義されているデフォルト値よりも小さい必要があります。そうでない場合、無視されます。タイムアウトのデフォルト値は、コンテキスト（トリガーまたはワークフロー）によって異なります。

例:

```
uses core.http as client;

...

begin

    var request := client.createRequest('ebx-1');

    client.setTimeout(request, 900);

    ...

end
```

パラメーター:

request: リクエスト

timeout: タイムアウト

使用可能な場所:

core.json ユニット

JSON データを管理するメソッドを提供するスクリプトユニットです。

このユニットは、文字列、10 進数、ブール値、null、オブジェクト、配列など、あらゆる種類の JSON を作成および解析するメソッドを提供します。配列の項目はすべて同じ型である必要はありません。

json_value 型の変数は、あらゆる種類の JSON 値を表すことができます。以下の読み取り専用ブールプロパティは決して null にならず、JSON 値の型をチェックするために使用できます。

- isNull: boolean - 値が null の場合に true
- isString: boolean - 値が文字列の場合
- isDecimal: boolean - 値が 10 進数の場合
- isBoolean: boolean - 値がブール値の場合
- isObject: boolean - 値がオブジェクトの場合
- isArray: boolean - 値が配列の場合

以下の読み取り専用プロパティを使用して JSON 値を取得できます。値が指定された型でない場合、または null の場合は null を返します。

- asDecimal: string - 値が文字列の場合
- asBoolean: boolean - 値がブール値の場合
- asObject: json_object - 値がオブジェクトの場合
- asArray: list<json_value> - 値が配列の場合

日時、日付、時刻の型は JSON でネイティブにサポートされていません。これらは、ユニット core.datetime、core.date、および core.time によって提供される関数 toIsoString() および fromIsoString() を使用して、文字列との間で変換できます。

サンプル 以下は JSON のサンプルです。

```
{  
  
  "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]

]

}

```

以下のスクリプトは、この JSON 文字列を解析する方法を示しています。

```

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

```

以下のスクリプトは、上記のサンプルと同一の JSON 文字列を作成する方法を示しています。

```

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 of 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

end


// Generate the JSON string.

return json.toJson(json.ofObject(user));

end

```

メソッド

`function arrayToJoson(list: list): string` JSON 値のリストを文字列にシリアル化します。

メソッド

`function copyOfBooleans(value: list): json\_value` ブール値のリストを配列型の JSON 値に変換します。

`function copyOfDecimals(value: list): json\_value` 10 進数のリストを配列型の JSON 値に変換します。

`function copyOfObjects(value: list): json\_value` JSON オブジェクトのリストを配列型の JSON 値に変換します。

`function copyOfStrings(value: list): json\_value` 文字列のリストを配列型の JSON 値に変換します。

`function exists(json: json\_object, key: string): boolean` JSON オブジェクトにキーが存在するかどうかをチェックします。

`function get(json: json\_object, key: string): json\_value` JSON オブジェクト内のキーに関連付けられた JSON 値を返します。

`function getArray(json: json\_object, key: string): list` このメソッドは core のショートカットです。

`function getBoolean(json: json\_object, key: string): decimal` このメソッドは core のショートカットです。

`function getDecimal(json: json\_object, key: string): decimal` このメソッドは core のショートカットです。

`function getObject(json: json\_object, key: string): object` このメソッドは core のショートカットです。

`function getString(json: json\_object, key: string): string` このメソッドは core のショートカットです。

`function isNotNull(json: json\_object, key: string): boolean` このメソッドは not core のショートカットです。

`function isNull(json: json\_object, key: string): boolean` キーに関連付けられた JSON 値が存在し、JSON null 値である場合に true を返します。

`function keys(object: json\_object): list` JSON オブジェクトのキーを文字列のリストとして返します。

メソッド

`function objectToJson(object: json\_object): string` JSON オブジェクトを文字列にシリアル化します。

`function ofArray(list: list): json\_value` 配列 JSON 値を返します。

`function ofBoolean(value: boolean): json\_value` ブール値 JSON 値を返します。

`function ofDecimal(value: decimal): json\_value` 10 進数 JSON 値を返します。

`function ofNull(): json\_value` null JSON 値を返します。

`function ofObject(value: json\_object): json\_value` オブジェクト JSON 値を返します。

`function ofString(value: string): json\_value` 文字列 JSON 値を返します。

`function parse(json: string): json\_value` JSON 文字列を解析します。

`function parseWithReport(json: string): json\_report` JSON 文字列を解析します。

`procedure put(json: json\_object, key: string, value: json\_value)` JSON オブジェクト内のキーに関連付けられた JSON 値を設定します。

`procedure putArray(json: json\_object, key: string, value: list)` このメソッドは core のショートカットです。

`procedure putBoolean(json: json\_object, key: string, value: boolean)` このメソッドは core のショートカットです。

`procedure putDecimal(json: json\_object, key: string, value: decimal)` このメソッドは core のショートカットです。

`procedure putNull(json: json\_object, key: string)` このメソッドは core のショートカットです。

`procedure putObject(json: json\_object, key: string, value: json\_object)` このメソッドは core のショートカットです。

`procedure putString(json: json\_object, key: string, value: string)` このメソッドは core のショートカットです。

`procedure remove(builder: json\_object, key: string)` キーによって識別される項目を削除します。

`function toJson(jsonValue: json\_value): string` JSON 値を文字列にシリアル化します。

- `arrayToJson`
- `copyOfBooleans`
- `copyOfDecimals`
- `copyOfObjects`
- `copyOfStrings`
- `exists`
- `get`
- `getArray`
- `getBoolean`
- `getDecimal`
- `getObject`
- `getString`
- `isNotNull`
- `isNull`
- `keys`
- `objectToJson`
- `ofArray`
- `ofBoolean`
- `ofDecimal`
- `ofNull`
- `ofObject`
- `ofString`
- `parse`
- `parseWithReport`
- `put`
- `putArray`
- `putBoolean`
- `putDecimal`
- `putNull`
- `putObject`

- putString
- remove
- toJson

arrayToJson

function arrayToJson(list: list): string

JSON 値のリストを文字列にシリアル化します。

戻り値:

JSON 文字列。入力リストが null の場合は null

使用可能箇所:

copyOfBooleans

function copyOfBooleans(value: list): json_value

ブール値のリストを配列型の JSON 値に変換します。入力リストを変更しても、返される JSON 値は変更されません。

パラメーター:

list: 入力リスト

戻り値:

JSON 値。入力リストが null の場合は ofNull() と同じ。決して null ではない

使用可能箇所:

copyOfDecimals

function copyOfDecimals(value: list): json_value

10 進数のリストを配列型の JSON 値に変換します。入力リストを変更しても、返される JSON 値は変更されません。

パラメーター:

list: 入力リスト

戻り値:

JSON 値。入力リストが null の場合は ofNull() と同じ。決して null ではない

使用可能箇所:

copyOfObjects

function copyOfObjects(value: list): json_value

JSON オブジェクトのリストを配列型の JSON 値に変換します。入力リストを変更しても、返される JSON 値は変更されません。コピーはディープではないため、リスト内のオブジェクトのプロパティを変更すると、返される JSON 値配列内の対応する JSON 値が変更されます。

パラメーター:

list: 入力リスト

戻り値:

JSON 値。入力リストが null の場合は ofNull() と同じ。決して null ではない

使用可能箇所:

copyOfStrings

function copyOfStrings(value: list): json_value

文字列のリストを配列型の JSON 値に変換します。入力リストを変更しても、返される JSON 値は変更されません。

パラメーター:

list: 入力リスト

戻り値:

JSON 値。入力リストが null の場合は ofNull() と同じ。決して null ではない

使用可能箇所:

exists

function exists(json: json_object, key: string): boolean

JSON オブジェクトにキーが存在するかどうかをチェックします。

使用可能箇所:

get**function get(json: json_object, key: string): json_value**

JSON オブジェクト内のキーに関連付けられた JSON 値を返します。

使用可能箇所:

getArray**function getArray(json: json_object, key: string): list**

このメソッドは `core.json.get(json, key).asArray` のショートカットです。

使用可能箇所:

getBoolean**function getBoolean(json: json_object, key: string): decimal**

このメソッドは `core.json.get(json, key).asBoolean` のショートカットです。

使用可能箇所:

getDecimal**function getDecimal(json: json_object, key: string): decimal**

このメソッドは `core.json.get(json, key).asDecimal` のショートカットです。

使用可能箇所:

getObject**function getObject(json: json_object, key: string): object**

このメソッドは `core.json.get(json, key).asObject` のショートカットです。

使用可能箇所:

getString**function getString(json: json_object, key: string): string**

このメソッドは `core.json.get(json, key).asString` のショートカットです。

使用可能箇所:

isNotNull

function isNotNull(json: json_object, key: string): boolean

このメソッドは not core.json.isNull(json, key) のショートカットです。

使用可能箇所:

isNull

function isNull(json: json_object, key: string): boolean

キーに関連付けられた JSON 値が存在し、JSON null 値である場合に true を返します。

使用可能箇所:

keys

function keys(object: json_object): list

JSON オブジェクトのキーを文字列のリストとして返します。キーの順序は保証されません。

パラメーター:

object: JSON オブジェクト

戻り値:

キーのリスト。オブジェクトが null の場合は null

使用可能箇所:

objectToJson

function objectToJson(object: json_object): string

JSON オブジェクトを文字列にシリアル化します。

戻り値:

JSON 文字列。入力 JSON オブジェクトが null の場合は null

使用可能箇所:

ofArray

function ofArray(list: list): json_value

配列 JSON 値を返します。入力リストを変更すると、返される JSON 配列が変更されます。

パラメーター:

value: 入力値

戻り値:

JSON 値。入力リストが null の場合は ofNull() と同じ。決して null ではない

使用可能箇所:

ofBoolean

function ofBoolean(value: boolean): json_value

ブール値 JSON 値を返します。

パラメーター:

value: 入力値

戻り値:

JSON 値。入力値が null の場合は ofNull() と同じ。決して null ではない

使用可能箇所:

ofDecimal

function ofDecimal(value: decimal): json_value

10 進数 JSON 値を返します。

パラメーター:

value: 入力値

戻り値:

JSON 値。入力値が null の場合は ofNull() と同じ。決して null ではない

使用可能箇所:

ofNull

function ofNull(): json_value

null JSON 値を返します。

戻り値:

null JSON 値を表す値。決して null ではない

使用可能箇所:

ofObject

function ofObject(value: json_object): json_value

オブジェクト JSON 値を返します。入力オブジェクトのプロパティを変更すると、返されるオブジェクト JSON 値のプロパティが変更されます。

パラメーター:

value: 入力値

戻り値:

JSON 値。入力値が null の場合は ofNull() と同じ。決して null ではない

使用可能箇所:

ofString

function ofString(value: string): json_value

文字列 JSON 値を返します。

パラメーター:

value: 入力値

戻り値:

JSON 値。入力値が null の場合は ofNull() と同じ。決して null ではない

使用可能箇所:

parse

function parse(json: string): json_value

JSON 文字列を解析します。JSON は任意の型にできます。

解析に失敗した場合、詳細はなく null を返します。問題の診断など、詳細なエラーメッセージが必要な場合は、代わりに `parseWithReport()` を使用してください。

パラメーター:

json: JSON 文字列

戻り値:

JSON 値。入力文字列が有効な JSON でない場合、または null の場合は null を返します。

使用可能箇所:

parseWithReport

function parseWithReport(json: string): json_report

JSON 文字列を解析します。JSON は任意の型にできます。解析に失敗した場合、問題の診断に役立つ可能性のあるエラーメッセージを返します。

返されるレポートには、以下の読み取り専用プロパティがあります。value: json_value - 解析結果。isSuccess が true または isError が false の場合は決して null ではない。isSuccess: boolean - 解析が成功した場合にのみ true。決して null ではない。isError: boolean - 解析が失敗した場合にのみ false。決して null ではない。message: string - エラーメッセージ。isSuccess が true または isError が false の場合は null の場合がある。

パラメーター:

json: JSON 文字列

戻り値:

JSON レポート。決して null ではない

使用可能箇所:

put

procedure put(json: json_object, key: string, value: json_value)

JSON オブジェクト内のキーに関連付けられた JSON 値を設定します。キーに関連付けられた以前の値は上書きされます。JSON オブジェクトまたはキーが null の場合はエラーが発生します。

使用可能箇所:

putArray

procedure putBoolean(json: json_object, key: string, value: list)

このメソッドは `core.json.put(json, key, core.json.ofArray(value))` のショートカットです。

使用可能箇所:

putBoolean

procedure putBoolean(json: json_object, key: string, value: boolean)

このメソッドは `core.json.put(json, key, core.json.ofBoolean(value))` のショートカットです。

使用可能箇所:

putDecimal

procedure putDecimal(json: json_object, key: string, value: decimal)

このメソッドは `core.json.put(json, key, core.json.ofDecimal(value))` のショートカットです。

使用可能箇所:

putNull

procedure putBoolean(json: json_object, key: string)

このメソッドは `core.json.put(json, key, core.json.ofNull())` のショートカットです。

使用可能箇所:

putObject

procedure putBoolean(json: json_object, key: string, value: json_object)

このメソッドは `core.json.put(json, key, core.json.ofObject(value))` のショートカットです。

使用可能箇所:

putString

procedure putBoolean(json: json_object, key: string, value: string)

このメソッドは `core.json.put(json, key, core.json.ofString(value))` のショートカットです。

使用可能箇所:

remove

procedure remove(builder: json_object, key: string)

キーによって識別される項目を削除します。存在しない項目を削除しても効果はありません。

使用可能箇所:

toJson

function toJson(jsonValue: json_value): string

JSON 値を文字列にシリアル化します。

パラメーター:

jsonValue: JSON 値

戻り値:

JSON 文字列。入力 JSON 値が null の場合は null。JSON null 値は、有効な JSON 文字列 'null' に正しくシリアル化されます。

使用可能箇所:

ユニット core.list

リストを作成および更新するためのメソッドを提供するスクリプトユニットです。

メソッド
procedure add(values: list<ItemType>, element: ItemType) 値リストにエレメントを追加します。
procedure addAll(values: list<ItemType>, elements: list<ItemType>): boolean すべてのエレメントを値リストに追加します。
procedure addFirst(list: list<ItemType>, element: ItemType) 指定されたエレメントをリストの先頭に追加します。
procedure clear(list: list) リストからすべてのエレメントを削除します。
function clone(list: list): list リストのシャロークローンを返します。
function contains(values: list<ItemType>, element: ItemType): boolean 値リストにエレメントが含まれている場合は true を返します。

メソッド

`function containsAll(values: list<ItemType>, element: ItemType): boolean` 値リストにすべてのエレメントが含まれている場合は true を返します。

`function indexOf(values: list<ItemType>, element: ItemType)` 値リスト内のエレメントの最初の出現のインデックスを返します。

`function isEmpty(value: list<ItemType>): boolean` 値が null またはサイズがゼロのリストの場合は true を返します。

`function lastIndexOf(values: list<ItemType>, element: ItemType)` 値リスト内のエレメントの最後の出現のインデックスを返します。

`function of<ItemType>(value: ItemType [, value: ItemType]...): list<ItemType>` パラメーターとして渡されたエレメントを使用してリストを作成します。

`function remove(values: list<ItemType>, element: ItemType): boolean` 値リストからエレメントの最初の出現を削除します。

`function removeAll(values: list<ItemType>, elements: list<ItemType>): boolean` 値リストからすべてのエレメントを削除します。

`function retainAll(values: list<ItemType>, elements: list<ItemType>): boolean` 値リスト内のエレメントのうち、エレメントに含まれるエレメントのみを保持します。

`procedure reverse(values: list<ItemType>)` 値内のエレメントの順序を逆にします。

`procedure reverseSort(values: list<ItemType>)` 指定された値のリストを、そのエレメントの自然な順序に従って降順に並べ替えます。

`procedure sort(values: list<ItemType>)` 指定された値のリストを、そのエレメントの自然な順序に従って昇順に並べ替えます。

`function subList<ItemType>(values: list<ItemType>, startIndex: decimal, endIndex: decimal): list<ItemType>` 値リストのエレメントのサブシーケンスを含む新しいリストを返します。

- `add`
- `addAll`
- `addFirst`
- `clear`
- `clone`

- contains
- containsAll
- indexOf
- isEmpty
- lastIndexOf
- of
- remove
- removeAll
- retainAll
- reverse
- reverseSort
- sort
- subList

add

procedure add(values: list<ItemType>, element: ItemType)

値リストにエレメントを追加します。

パラメーター

values：入力リスト

element：追加する入力エレメント。そのタイプは、リストのタイプと互換性がある必要があります。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

addAll

procedure addAll(values: list<ItemType>, elements: list<ItemType>): boolean

すべてのエレメントを値リストに追加します。

パラメーター

values：入力リスト

elements：追加するエレメント。エレメントタイプは、リストのタイプと互換性がある必要があります。

戻り値

この呼び出しによって値リストの内容が変更された場合は true、そうでない場合は false

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

addFirst

procedure addFirst(list: list<ItemType>, element: ItemType)

指定されたエレメントをリストの先頭に追加します。

例：

```
uses core.list as l;

procedure doSomething()

begin

    var list := ["apple", "banana", "orange"];

    l.addFirst(list, "pear"); // ["pear", "apple", "banana", "orange"]

    ...

end
```

パラメーター

list：更新するリスト

element：リストの先頭に追加するエレメント

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

clear

procedure clear(list: list)

リストからすべてのエレメントを削除します。

パラメーター

list：クリアするリスト

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

clone

function clone(list: list): list

リストのシャロークローンを返します。

パラメーター

list：クローンするリスト

戻り値

入力リストと同じエレメントを持つ新しいリスト

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

contains

function contains(values: list<ItemType>, element: ItemType): boolean

値リストにエレメントが含まれている場合は true を返します。

パラメーター

values：入力リスト。内容はエレメントと同じ種類にする必要があります

element：検索する入力エレメント。値の内容と同じ種類にする必要があります

戻り値

値にエレメントが含まれている場合は true、含まれていない場合は false

値が null の場合、null を返します

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

containsAll

function containsAll(values: list<ItemType>, element: ItemType): boolean

値リストにすべてのエレメントが含まれている場合は true を返します。

パラメーター

values：入力リスト。内容はエレメントと同じ種類にする必要があります

elements：検索する入力エレメント。値と同じ種類にする必要があります

戻り値

値にすべてのエレメントが含まれている場合は true、それ以外の場合は false

いずれかのパラメーターが null の場合、null を返します

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

indexOf

function indexOf(values: list<ItemType>, element: ItemType)

値リスト内のエレメントの最初の出現のインデックスを返します。

パラメーター

values：入力リスト

element：検索する入力エレメント。そのタイプは、リストのタイプと互換性がある必要があります

戻り値

値リスト内のエレメントの最初のカレンスのインデックス。見つからない場合は null

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

isEmpty

function isEmpty(value: list<ItemType>): boolean

値が null またはサイズがゼロのリストの場合は true を返します。

パラメーター

values：入力リスト

戻り値

値が null またはサイズがゼロのリストの場合は true、それ以外の場合は false。null を返すことはありません

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

lastIndexOf

function lastIndexOf(values: list<ItemType>, element: ItemType)

値リスト内の要素の最後の出現のインデックスを返します。

パラメーター

values：入力リスト

element：検索する入力要素

戻り値

値リスト内の要素の最後のオカレンスのインデックス。見つからない場合は null

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

of

function of<ItemType>(value: ItemType ...): list<ItemType>

パラメーターとして渡された要素を使用してリストを作成します。すべての要素は同じタイプにする必要があります。

例：

```
uses core.list as list;

function getCities(): list<string>
begin
    return list.of('Paris', 'Bruxelles', 'Berlin');
end
```

関数の型

ItemType：返されたリストのアイテムのタイプ。少なくとも 1 つの値が指定されている場合はオプションです

パラメーター

value: リストに追加するアイテム。すべてのアイテムは同じタイプにする必要があります

戻り値

新しいリスト。更新可能です

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

remove

function remove(values: list<ItemType>, element: ItemType): boolean

値リストからエレメントの最初の出現を削除します。

パラメーター

values：入力リスト

element：追加する入力エレメント。そのタイプは、リストのタイプと互換性がある必要があります

戻り値

値リストの内容がこの呼び出しによって変更された場合は true、そうでない場合は false

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

removeAll

function removeAll(values: list<ItemType>, elements: list<ItemType>): boolean

値リストからすべてのエレメントを削除します。

パラメーター

values：入力リスト

elements：削除するエレメント。エレメントタイプは、リストのタイプと互換性がある必要があります

戻り値

値リストの内容がこの呼び出しによって変更された場合は true、そうでない場合は false

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

retainAll

function retainAll(values: list<ItemType>, elements: list<ItemType>): boolean

値リスト内のエレメントのうち、エレメントに含まれるエレメントのみを保持します。つまり、値リストから、エレメントに含まれないすべてのエレメントを削除します。

パラメーター

values：入力リスト

elements：保持するエレメント。エレメントタイプは、リストのタイプと互換性がある必要があります

戻り値

値リストの内容がこの呼び出しによって変更された場合は true、そうでない場合は false

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

reverse

procedure reverse(values: list<ItemType>)

値内のエレメントの順序を逆にします。値が null またはサイズがゼロのリストの場合は何も実行されません。

パラメーター

values：反転する入力リスト

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

reverseSort

procedure reverseSort(values: list<ItemType>)

指定された値のリストを、そのエレメントの自然な順序に従って降順に並べ替えます。値が null またはサイズがゼロのリストの場合は何も実行されません。

パラメーター

values：並べ替える入力リスト

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

sort

procedure sort(values: list<ItemType>)

指定された値のリストを、そのエレメントの自然な順序に従って昇順に並べ替えます。値が null またはサイズがゼロのリストの場合は何も実行されません。

パラメーター

values：並べ替える入力リスト

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

subList

function subList<ItemType>(values: list<ItemType>, startIndex: decimal, endIndex: decimal): list<ItemType>

値リストのエレメントのサブシーケンスを含む新しいリストを返します。

subListは、指定された startIndex から始まり、入力リストの endIndex まで拡張されます。

関数の型

ItemType：返されたリストのアイテムのタイプ。オプションです

パラメーター

values：入力リスト

startIndex: 入力開始インデックス。正の整数またはゼロにする必要があります

endIndex：入力終了インデックス。正の整数またはゼロにする必要があります

戻り値

新しいリスト。パラメーターが null の場合は null

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

ユニット core.locale

ロケールを管理するための機能を提供するスクリプトユニット。

メソッド

function fromString(value: string): locale 文字列からロケールを作成します。

メソッド

`function of(language: string, country: string): locale` ロケールを作成します。

- `fromString`
- `of`

fromString

`function fromString(value: string): locale`

文字列からロケールを作成します。

パラメーター

value: ロケール (例: 'en-US')。

戻り値

ロケール。

使用可能な場所:

of

`function of(language: string, country: string): locale`

ロケールを作成します。

パラメーター

language: ISO 639 alpha-2 または alpha-3 言語コード。

country: ISO 3166 alpha-2 国コードまたは UN M.49 numeric-3 市外局番。

戻り値

ロケール。

使用可能な場所:

ユニット core.log

メッセージをログに記録するメソッドを提供するスクリプトユニット。このメッセージは、EBX が提供する管理ユーザーインターフェイスを使用して表示できます。

メソッド

`procedure error(message: string)` エラーメッセージをログに記録します。

`procedure info(message: string)` 情報メッセージをログに記録します。

`procedure warn(message: string)` 警告メッセージをログに記録します。

- `error`
- `info`
- `warn`

error**`procedure error(message: string)`**

エラーメッセージをログに記録します。

パラメーター

value: message: メッセージ。

使用可能な場所：

info**`procedure info(message: string)`**

情報メッセージをログに記録します。

パラメーター

value: message: メッセージ。

使用可能な場所：

warn**`procedure warn(message: string)`**

警告メッセージをログに記録します。

パラメーター

value: message: メッセージ。

使用可能な場所：

ユニット core.math

数学関数と定数を提供するスクリプトユニットです。

メソッド
<code>function abs(value: decimal): decimal</code> 小数値の絶対値を返します。
<code>function acos(value: decimal): decimal</code> 値のアーコサインを返します。
<code>function asin(value: decimal): decimal</code> 値のアークサインを返します。
<code>function atan(value: decimal): decimal</code> 値のアークタングェントを返します。
<code>function cbrt(value: decimal): decimal</code> 小数値の立方根を返します。
<code>function ceil(value: decimal): decimal</code> 入力値の整数 ceil 値を返します。
<code>function cos(value: decimal): decimal</code> 角度の三角関数の余弦を返します。
<code>function e(): decimal</code> e 値 2 を返します。
<code>function exp(value: decimal): decimal</code> オイラー数 e を小数値で累乗した値を返します。
<code>function floor(value: decimal): decimal</code> 入力値の整数フロア値を返します。
<code>function log(value: decimal): decimal</code> 入力値の自然対数 (基数 e) を返します。
<code>function max(a: decimal, b: decimal): decimal</code> 2つの小数値のうち大きい方を返します。
<code>function min(a: decimal, b: decimal): decimal</code> 2つの小数値のうち小さい方を返します。
<code>function pi(): decimal</code> 円周率の値 3 を返します。
<code>function pow(a: decimal, b: decimal): decimal</code> 最初の引数を2番目の引数の累乗で返します。
<code>function random(): decimal</code> 0以上1未満の正の符号を持つランダムな小数値を返します。
<code>function randomInt(maxValue: decimal): decimal</code> 0 (含む) から指定された <code>bound</code> (含まない) までのランダムな整数を返します。
<code>function round(value: decimal): decimal</code> 入力値の整数の丸め値を返します。

メソッド

`function roundDown(value: decimal): decimal` 入力値の整数の切り捨て値を返します。

`function roundHalfDown(value: decimal): decimal` 入力値の整数の切り捨て半分の値を返します。

`function roundHalfEven(value: decimal): decimal` 入力値の整数の四捨五入値を返します。

`function roundUp(value: decimal): decimal` 入力値の整数の切り上げ値を返します。

`function scaleByPowerOfTen(value: decimal, power: decimal): decimal` 値のスケールを 10 の累乗で返します。

`function sin(value: decimal): decimal` 角度の三角関数の正弦を返します。

`function sqrt(value: decimal): decimal` 値の正の平方根を返します。

`function tan(value: decimal): decimal` 角度の三角法の接線を返します。

`function toDegrees(value: decimal): decimal` 度数で測定された角度を、ラジアンで測定されたほぼ同等の角度に変換します。

`function toRadians(value: decimal): decimal` ラジアンで測定された角度を、度数で測定されたほぼ同等の角度に変換します。

- `abs`
- `acos`
- `asin`
- `atan`
- `cbrt`
- `ceil`
- `cos`
- `e`
- `exp`
- `floor`
- `log`
- `max`
- `min`

- [pi](#)
- [pow](#)
- [random](#)
- [randomInt](#)
- [round](#)
- [roundDown](#)
- [roundHalfDown](#)
- [roundHalfEven](#)
- [roundUp](#)
- [scaleByPowerOfTen](#)
- [sin](#)
- [sqrt](#)
- [tan](#)
- [toDegrees](#)
- [toRadians](#)

abs

function abs(value: decimal): decimal

小数値の絶対値を返します。

パラメーター

value: 入力値

戻り値

入力値の絶対値。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

acos

function acos(value: decimal): decimal

値のアークコサインを返します。

返される角度は、0.0 から π の範囲です。

計算は倍精度変換後に実行され、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

パラメーター

value: 入力値。絶対値を 1 より大きくすることはできません。

戻り値

入力値のアーコサイン。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

asin

function asin(value: decimal): decimal

値のアーコサインを返します。

返される角度は、 $-pi/2$ から $pi/2$ の範囲です。

特殊なケース

入力値がゼロの場合、結果はゼロになります。

計算は倍精度変換後に実行され、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

パラメーター

value: 入力値。絶対値を 1 より大きくすることはできません。

戻り値

入力値のアーコサイン。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

atan

function atan(value: decimal): decimal

値のアーコタンジェントを返します。

返される角度は、 $-pi/2$ から $pi/2$ の範囲です。

特殊なケース

入力値がゼロの場合、結果はゼロになります。

計算は倍精度変換後に実行され、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

パラメーター

value: 入力値

戻り値

入力値のアークタンジェント。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

cbirt

function cbirt(value: decimal): decimal

小数值の立方根を返します。計算には倍精度 64 ビット IEEE 754 浮動小数点数が使用されます。

パラメーター

value: 入力値

戻り値

入力値の立方根。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

ceil

function ceil(value: decimal): decimal

入力値の整数 ceil 値を返します。

パラメーター

value: 入力値

戻り値

入力値の ceil 値、または入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

cos

function cos(value: decimal): decimal

角度の三角関数の余弦を返します。

計算は倍精度変換後に実行され、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

パラメーター

value: ラジアン単位の入力値。

戻り値

入力値のコサイン。入力値 null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

e

function e(): decimal

e 値 2.718281828459045235360287471352662 を返します。

戻り値

e 値。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

exp

function exp(value: decimal): decimal

オイラー数 e を小数値で累乗した値を返します。

計算は倍精度変換後に実行され、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

パラメーター

value: 入力値

戻り値

値 e^{value} 。ここで、 e は自然対数の底です。

入力値が null の場合、null を返します。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

floor**function floor(value: decimal): decimal**

入力値の整数フロア値を返します。

パラメーター

value: 入力値

戻り値

入力値のフロア値。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

log**function log(value: decimal): decimal**

入力値の自然対数 (基数 e) を返します。

計算では、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

パラメーター

value: 入力値。厳密にゼロより大きくする必要があります。

戻り値

入力値の自然対数 a または入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

max**function max(a: decimal, b: decimal): decimal**

2つの小数值のうち大きい方を返します。

例:

```
uses core.math as m;

procedure doSomething()

begin
```

```

    var result := m.max(10, 5); // 10

    ...

end

```

パラメーター

a: 最初の入力値

b: 2番目の入力値

戻り値

2つの入力値のうち大きい方

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

min

function min(a: decimal, b: decimal): decimal

2つの小数值のうち小さい方を返します。

例:

```

uses core.math as m;

procedure doSomething()

begin

    var result := m.min(10, 5); // 5

    ...

end

```

パラメーター

a: 最初の入力値

b: 2番目の入力値

戻り値

2つの入力値のうち小さい方

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

pi

function pi(): decimal

円周率の値 3.141592653589793238462643383279503 を返します。

戻り値

円周率の値。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

pow

function pow(a: decimal, b: decimal): decimal

最初の引数を2番目の引数の累乗で返します。パラメーター n は、-999999999 から 999999999 (両端を含む) の範囲である必要があります。pow(0, 0) は 1 を返します。

例:

```
uses core.math as m;

procedure doSomething()

begin

    var result := m.pow(2, 3); // 8

    ...

end
```

パラメーター

a: 基となる入力値

b: 指数となる入力値

戻り値

a を b の累乗で計算した結果

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

random

function random(): decimal

0以上1未満の正の符号を持つランダムな小数値を返します。注意、このメソッドは純粋ではありません。

例:

```

uses core.math as m;

procedure doSomething()

begin

    var result := m.random(); // 0.8574303492

    ...

end

```

戻り値

0 (含む) から 1 (含まない) までの小数値

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

randomInt

function randomInt(maxValue: decimal): decimal

0 (含む) から指定された **bound** (含まない) までのランダムな整数を返します。注意、このメソッドは純粋ではありません。

例:


```

uses core.math as m;

procedure doSomething()

begin

    var result := m.randomInt(10); // a random integer between 0 and 9

    ...

end

```

パラメーター

bound: 上限 (含まない)、正の値である必要があります

戻り値

0 (含む) から `bound` (含まない) までのランダムな整数

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

round

function round(value: decimal): decimal

入力値の整数の丸め値を返します。このメソッドは、標準の切り上げハーフアップ丸めモードを使用します。

パラメーター

value: 入力値

戻り値

入力値の切り上げ値。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

roundDown

function roundDown(value: decimal): decimal

入力値の整数の切り捨て値を返します。

パラメーター

value: 入力値

戻り値

入力値の切り捨て値。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

roundHalfDown

function roundHalfDown(value: decimal): decimal

入力値の整数の切り捨て半分の値を返します。

パラメーター

value: 入力値

戻り値

入力値の切り捨て半分の値。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

roundHalfEven

function roundHalfEven(value: decimal): decimal

入力値の整数の四捨五入値を返します。

パラメーター

value: 入力値

戻り値

入力値の半偶数値、または入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

roundUp

function roundUp(value: decimal): decimal

入力値の整数の切り上げ値を返します。

パラメーター

value: 入力値

戻り値

入力値の切り上げ値。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

scaleByPowerOfTen

function scaleByPowerOfTen(value: decimal, power: decimal): decimal

値のスケールを 10 の累乗で返します。

パラメーター

value: 入力値

power: 値がスケールされる 10 の累乗。整数にする必要があります。

戻り値

入力値を 10 の累乗でスケーリングしたもの、または入力値または累乗が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

sin

function sin(value: decimal): decimal

角度の三角関数の正弦を返します。

計算は倍精度変換後に実行され、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

特殊なケース

入力値がゼロの場合、結果はゼロになります。

パラメーター

value: ラジアン単位の入力値。

戻り値

入力値の正弦。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

sqrt

function sqrt(value: decimal): decimal

値の正の平方根を返します。

計算では、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

パラメーター

value: 入力値。正または null にする必要があります。

戻り値

入力値の正の平方根。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

tan

function tan(value: decimal): decimal

角度の三角法の接線を返します。

特殊なケース

入力値がゼロの場合、結果はゼロになります。

計算は倍精度変換後に実行され、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

パラメーター

value: ラジアン単位の入力値。

戻り値

入力値のタンジェント。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

toDegrees

function toDegrees(value: decimal): decimal

度数で測定された角度を、ラジアンで測定されたほぼ同等の角度に変換します。

度数からラジアンへの変換は、一般的に不正確です。

計算は倍精度変換後に実行され、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

パラメーター

value: ラジアン単位の入力値。

戻り値

度数単位の入力値の測定値。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

toRadians

function toRadians(value: decimal): decimal

ラジアンで測定された角度を、度数で測定されたほぼ同等の角度に変換します。

ラジアンから度数への変換は一般的に不正確です。

ユーザーは、`cos(toRadians(90.0))` が正確に 0.0 に等しいと期待すべきではありません。

計算は倍精度変換後に実行され、倍精度 64 ビット IEEE 754 浮動小数点数を使用します。

パラメーター

value: 度単位の入力値

戻り値

ラジアン単位の入力値の測定値。入力値が null の場合は null。

使用可能な場所：スクリプトタスク、テーブルトリガー、関数フィールド

ユニット core.message

ユーザーメッセージを作成および管理するメソッドを提供するスクリプトユニットです。

メソッド

`procedure addError(message: string)` 指定されたエラーメッセージを現在の検証コンテキストに追加します。

メソッド

`procedure addInfo(message: string)` 指定された情報メッセージを現在の検証コンテキストに追加します。

`function addLocale(userMessage: user\_message, message: string, locale: locale): user\_message` ユーザーメッセージにロケールを追加します。

`procedure addMessage(message: user\_message)` 指定されたユーザーメッセージを現在の検証コンテキストに追加します。

`procedure addRecordMessage(record: record, message: user\_message)` 指定されたレコードの現在の検証コンテキストに情報メッセージを追加します。

`procedure addWarning(message: string)` 指定された警告メッセージを現在の検証コンテキストに追加します。

`procedure addXPathMessage(xpath: string, message: user\_message)` 指定されたXPath述語に一致するレコードの現在の検証コンテキストにユーザーメッセージを追加します。

`function of(message: string, locale: locale): user\_message` 単一のロケールに対してユーザーメッセージのインスタンスを作成します。

`function ofBundleError(path: string, key: string, parameters: list<object>): user\_message` メッセージバンドルからエラーユーザーメッセージのインスタンスを作成します。

`function ofBundleInfo(path: string, key: string, parameters: list<object>): user\_message` メッセージバンドルから情報ユーザーメッセージのインスタンスを作成します。

`function ofBundleWarning(path: string, key: string, parameters: list<object>): user\_message` メッセージバンドルから警告ユーザーメッセージのインスタンスを作成します。

`function toString(userMessage: user\_message, locale: locale): string` 指定されたロケールでメッセージをフォーマットします。

- `addError`
- `addInfo`
- `addLocale`
- `addMessage`
- `addRecordMessage`
- `addWarning`

- [addXPathMessage](#)
- [of](#)
- [ofBundleError](#)
- [ofBundleInfo](#)
- [ofBundleWarning](#)
- [toString](#)

addError

procedure addError(message: string)

指定されたエラーメッセージを現在の検証コンテキストに追加します。

例:

```
uses core.message as msg;

...

export procedure checkRecord(record : .Project)

begin

    msg.addError('This is an error message');

end

...
```

パラメーター:

message: 文字列メッセージ。

使用可能箇所:

addInfo

procedure addInfo(message: string)

指定された情報メッセージを現在の検証コンテキストに追加します。

例:

```

uses core.message as msg;

...

export procedure checkRecord(record : .Project)

begin

    msg.addInfo('This is an info message');

end

...

```

パラメーター:

message: 文字列メッセージ。

使用可能箇所:**addLocale**

**function addLocale(userMessage: user_message, message: string, locale: locale):
user_message**

ユーザーメッセージにロケールを追加します。

例:

```

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
```

パラメーター:

userMessage: ユーザーメッセージ。

message: ロケールの言語での文字列メッセージ。null の場合、空の文字列が使用されます。

locale: ロケール。null の場合、デフォルトのロケールが使用されます。

戻り値:

追加されたロケールを持つ新しいメッセージ。入力 userMessage が null の場合、指定されたロケールのみを持つ新しいユーザーメッセージを作成します。

使用可能箇所:

addMessage

procedure addMessage(message: user_message)

指定されたユーザーメッセージを現在の検証コンテキストに追加します。

例:

```
uses core.message as msg;

...

export procedure checkRecord(record : .Project)

begin

    var userMessage:=msg.ofBundleError('Key.invalid.value', 'Bundle/path');

    msg.addMessage(userMessage);

end

...
```

パラメーター:

message: ユーザーメッセージ。

使用可能箇所:

addRecordMessage

procedure addRecordMessage(record: record, message: user_message)

指定されたレコードの現在の検証コンテキストに情報メッセージを追加します。

例:

```
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

...
```

パラメーター:

record: レコード。

message: ユーザーメッセージ。

使用可能箇所:

addWarning

procedure addWarning(message: string)

指定された警告メッセージを現在の検証コンテキストに追加します。

例:

```
uses core.message as msg;

...

export procedure checkRecord(record : .Project)

begin

    msg.addWarning('This is a warning message');

end

...
```

パラメーター:

message: 文字列メッセージ。

使用可能箇所:

addXPathMessage

procedure addXPathMessage(xpath: string, message: user_message)

指定されたXPath述語に一致するレコードの現在の検証コンテキストにユーザーメッセージを追加します。

例:

```
uses core.message as msg;

...

export procedure checkTable()

begin

    var userMessage:=msg.ofBundleError('key.key1', 'Bundle/path');

    msg.addXPathMessage(' ./number<100 ',userMessage);

end
```

```
end
```

```
...
```

パラメーター:

xPath: メッセージが適用されるレコードのXPath述語。

message: ユーザーメッセージ。

使用可能箇所:**of**

function of(message: string, locale: locale): user_message

単一のロケールに対してユーザーメッセージのインスタンスを作成します。

例:

```
uses core.message as msg;

export procedure executeScriptTask()

begin

    var en:=locale.of('en', 'EN');

    var message:=msg.of('A new message', en);

end
```

パラメーター:

message: ロケールの言語での文字列メッセージ。null の場合、空の文字列が使用されます。

locale: ロケール。null の場合、デフォルトのロケールが使用されます。

戻り値:

新しいユーザーメッセージ。

使用可能箇所:

ofBundleError

function ofBundleError(path: string, key: string, parameters: list<object>): user_message

メッセージバンドルからエラーユーザーメッセージのインスタンスを作成します。

バンドルからのユーザーメッセージは、EBX IDE でパスとキーを指定することで作成できます。

例:

```
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

...
```

パラメーター:

key: バンドル内のメッセージのキー。

parameters: メッセージのパラメーター。

戻り値:

新しいユーザーメッセージ。

使用可能箇所:**ofBundleInfo**

function ofBundleInfo(path: string, key: string, parameters: list<object>): user_message

メッセージバンドルから情報ユーザーメッセージのインスタンスを作成します。

バンドルからのユーザーメッセージは、EBX IDE でパスとキーを指定することで作成できます。

例:

```
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
```

パラメーター:

key: バンドル内のメッセージのキー。

path: バンドルのパス。

parameters: メッセージのパラメーター。

戻り値:

新しいユーザーメッセージ。

使用可能箇所:

ofBundleWarning

function ofBundleWarning(path: string, key: string, parameters: list<object>):
user_message

メッセージバンドルから警告ユーザーメッセージのインスタンスを作成します。

バンドルからのユーザーメッセージは、EBX IDE でパスとキーを指定することで作成できます。

例:

```
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

....
```

パラメーター:

key: バンドル内のメッセージのキー。

path: バンドルのパス。

parameters: メッセージのパラメーター。

戻り値:

新しいユーザーメッセージ。

使用可能箇所:

toString

function toString(userMessage: user_message, locale: locale): string

指定されたロケールでメッセージをフォーマットします。

例:

```

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

```

パラメーター:

userMessage: ユーザーメッセージ。

locale: ロケール。null の場合、デフォルトのロケールが使用されます。

戻り値:

指定されたロケールでフォーマットされた文字列メッセージ。入力 userMessage が null の場合、null。

使用可能箇所:

ユニット core.profile

プロファイルの管理に役立つメソッドを提供するスクリプトユニット。

メソッド
function forSpecificRole(roleName: string): profile 指定された名前の特定のロールを返します。
function ofAdministrator(): profile 組み込みの「administrator」ロールを返します。
function ofEveryone(): profile 組み込みの「everyone」ロールを返します。
function ofOwner(): profile 組み込みの「owner」ロールを返します。
function ofReadOnly(): profile 組み込みの「read-only」ロールを返します。

- `forSpecificRole`
- `ofAdministrator`
- `ofEveryone`
- `ofOwner`
- `ofReadOnly`

forSpecificRole

function forSpecificRole(roleName: string): profile

指定された名前の特定のロールを返します。

使用可能場所:

ofAdministrator

function ofAdministrator(): profile

組み込みの「administrator」ロールを返します。

使用可能場所:

ofEveryone

function ofEveryone(): profile

組み込みの「everyone」ロールを返します。

使用可能場所:

ofOwner

function ofOwner(): profile

組み込みの「owner」ロールを返します。

使用可能場所:

ofReadOnly

function ofReadOnly(): profile

組み込みの「read-only」ロールを返します。

使用可能場所:

ユニット core.resource

このスクリプトユニットは、スキーマ タイプ ods:resource と互換性のあるリソース参照を生成するメソッドです。

例

```
uses core.resource as resource;

export function getValue(): typeof record.functions.ResourceValue

begin

    return resource.toImage('ebx-test', 'on_anim_wait.gif');

end
```

メソッド

`function toHtml(moduleName: string, filePath: string): string` 画像リソースへの参照を返します。

`function toIcon(moduleName: string, filePath: string): string` アイコンリソースへの参照を返します。

`function toImage(moduleName: string, filePath: string): string` 画像リソースへの参照を返します。

`function toJavaScript(moduleName: string, filePath: string): string` JavaScript ファイルリソースへの参照を返します。

`function toStyleSheet(moduleName: string, filePath: string): string` スタイルシートファイルリソースへの参照を返します。

- `toHtml`
- `toIcon`
- `toImage`
- `toJavaScript`
- `toStyleSheet`

toHtml**function toHtml(moduleName: string, filePath: string): string**

画像リソースへの参照を返します。

パラメーター

moduleName: モジュール名。

filePath: リソースのローカルファイルパス。

戻り値

スキーマタイプ ods:resource と互換性のあるリソース文字列。

使用可能な場所：

tolcon**function tolcon(moduleName: string, filePath: string): string**

アイコンリソースへの参照を返します。

パラメーター

moduleName: モジュール名。

filePath: リソースのローカルファイルパス。

戻り値

スキーマタイプ ods:resource と互換性のあるリソース文字列。

使用可能な場所：

tolImage**function tolImage(moduleName: string, filePath: string): string**

画像リソースへの参照を返します。

パラメーター

moduleName: モジュール名。

filePath: リソースのローカルファイルパス。

戻り値

スキーマタイプ ods:resource と互換性のあるリソース文字列。

使用可能な場所：

toJavaScript

function toJavaScript(moduleName: string, filePath: string): string

JavaScript ファイルリソースへの参照を返します。

パラメーター

moduleName: モジュール名。

filePath: リソースのローカルファイルパス。

戻り値

スキーマタイプ ods:resource と互換性のあるリソース文字列。

使用可能な場所：

toStyleSheet

function toStyleSheet(moduleName: string, filePath: string): string

スタイルシートファイルリソースへの参照を返します。

パラメーター

moduleName: モジュール名。

filePath: リソースのローカルファイルパス。

戻り値

スキーマタイプ ods:resource と互換性のあるリソース文字列。

使用可能な場所：

ユニット core.set

セットの作成と操作のためのメソッドを提供するスクリプトユニット。

メソッド

`function add(set: set<ItemType>, element: ItemType): boolean` 指定された要素がまだ存在しない場合、セットに追加します。

`function addAll(set: set<ItemType>, elements: collection<ItemType>): boolean` 指定されたコレクションのすべての要素をセットに追加します。

`procedure clear(set: set<ItemType>)` セットからすべての要素を削除します。

`function clone(set: set<ItemType>): set<ItemType>` セットのシャロークローンを返します。

`function contains(set: set<ItemType>, element: ItemType): boolean` セットに指定された要素が含まれている場合、trueを返します。

`function containsAll(set: set<ItemType>, elements: collection<ItemType>): boolean` セットに指定されたコレクションのすべての要素が含まれている場合、trueを返します。

`function difference(set1: set<ItemType>, set2: set<ItemType>): set<ItemType>` 最初のセットには存在するが2番目のセットには存在しない要素を含む新しいセットを返します (差集合)。

`function equals(set1: set<ItemType>, set2: set<ItemType>): boolean` 2つのセットが等しいかどうかを比較します。

`function fromList(list: list<ItemType>): set<ItemType>` リストを、同じ要素を含むセットに変換し、最初の出現順序を保持します。

`function intersection(set1: set<ItemType>, set2: set<ItemType>): set<ItemType>` 両方のセットに存在する要素のみを含む新しいセットを返します (積集合)。

`function isEmpty(value: set): boolean` 値がnullまたは空のセットである場合、trueを返します。

`function of(value: ItemType [, value: ItemType]...): set` パラメーターとして渡された要素を持つセットを作成します。

`function remove(set: set<ItemType>, element: ItemType): boolean` 指定された要素が存在する場合、セットから削除します。

`function removeAll(set: set<ItemType>, elements: collection<ItemType>): boolean` セットから、指定されたコレクションに含まれるすべての要素を削除します。

`function retainAll(set: set<ItemType>, elements: collection<ItemType>): boolean` セット内で、指定されたコレクションに含まれる要素のみを保持します。

メソッド

`function size(set: set<ItemType>): integer` セット内の要素数を返します。

`function toList(set: set<ItemType>): list<ItemType>` セットを、同じ要素を含むリストに変換し、順序を保持します。

`function union(set1: set<ItemType>, set2: set<ItemType>): set<ItemType>` 両方のセットのすべての要素を含む新しいセットを返します (和集合)。

- `add`
- `addAll`
- `clear`
- `clone`
- `contains`
- `containsAll`
- `difference`
- `equals`
- `fromList`
- `intersection`
- `isEmpty`
- `of`
- `remove`
- `removeAll`
- `retainAll`
- `size`
- `toList`
- `union`

add

`function add(set: set<ItemType>, element: ItemType): boolean`

指定された要素がまだ存在しない場合、セットに追加します。

パラメーター:

set: 要素を追加するセット。

element: セットに追加する要素。

戻り値:

セットに指定された要素がまだ含まれていなかった場合、true。

使用可能場所:

addAll

function addAll(set: set<ItemType>, elements: collection<ItemType>): boolean

指定されたコレクションのすべての要素をセットに追加します。

パラメーター:

set: 要素を追加するセット。

elements: セットに追加する要素を含むコレクション。

戻り値:

呼び出しの結果、セットが変更された場合、true。

使用可能場所:

clear

procedure clear(set: set<ItemType>)

セットからすべての要素を削除します。

パラメーター:

set: クリアするセット。

使用可能場所:

clone

function clone(set: set<ItemType>): set<ItemType>

セットのシャロークローンを返します。

パラメーター:

set: クローンするセット。

戻り値:

入力セットと同じ要素を持つ新しいセット。

使用可能場所:

contains

function contains(set: set<ItemType>, element: ItemType): boolean

セットに指定された要素が含まれている場合、trueを返します。

パラメーター:

set: 入力セット。

element: セット内に存在するかどうかをテストする要素。

戻り値:

セットに指定された要素が含まれている場合はtrue、それ以外の場合はfalse。セットがnullの場合はnullを返します。

使用可能場所:

containsAll

function containsAll(set: set<ItemType>, elements: collection<ItemType>): boolean

セットに指定されたコレクションのすべての要素が含まれている場合、trueを返します。

パラメーター:

set: 入力セット。

elements: セットに含まれているか確認するコレクション。

戻り値:

セットに指定されたコレクションのすべての要素が含まれている場合はtrue、それ以外の場合はfalse。いずれかのパラメーターがnullの場合はnullを返します。

使用可能場所:

difference

function difference(set1: set<ItemType>, set2: set<ItemType>): set<ItemType>

最初のセットには存在するが2番目のセットには存在しない要素を含む新しいセットを返します (差集合)。

パラメーター:

set1: 要素を削除するセット。

set2: set1から削除する要素を含むセット。

戻り値:

set1には存在するがset2には存在しない要素を含む新しいセット。set1がnullの場合は空のセットを返します。

使用可能場所:**equals**

function equals(set1: set<ItemType>, set2: set<ItemType>): boolean

2つのセットが等しいかどうかを比較します。

パラメーター:

set1: 最初のセット。

set2: 2番目のセット。

戻り値:

両方のセットが同じ要素を含んでいる場合はtrue、それ以外の場合はfalse。

使用可能場所:**fromList**

function fromList(list: list<ItemType>): set<ItemType>

リストを、同じ要素を含むセットに変換し、最初の出現順序を保持します。

パラメーター:

list: 入力リスト。

戻り値:

リストの一意の要素を、最初の出現順序で含むセット。

使用可能場所:**intersection**

function intersection(set1: set<ItemType>, set2: set<ItemType>): set<ItemType>

両方のセットに存在する要素のみを含む新しいセットを返します (積集合)。

パラメーター:

set1: 最初のセット。

set2: 2番目のセット。

戻り値:

両方のセットに存在する要素のみを含む新しいセット。いずれかのパラメーターがnullの場合は空のセットを返します。

使用可能場所:**isEmpty**

function isEmpty(value: set): boolean

値がnullまたは空のセットである場合、trueを返します。

パラメーター:

value: 入力セット。

戻り値:

値がnullまたは空のセットである場合はtrue、それ以外の場合はfalse。

使用可能場所:**of**

function of(value: ItemType ...): set

パラメーターとして渡された要素を持つセットを作成します。すべての要素は同じ型である必要があります。

パラメーター:

value: セットに追加する項目。すべての項目は同じ型である必要があります。

戻り値:

新しいセット。更新可能です。

使用可能場所:**remove**

function remove(set: set<ItemType>, element: ItemType): boolean

指定された要素が存在する場合、セットから削除します。

パラメーター:

set: 要素を削除するセット。

element: 存在する場合、セットから削除する要素。

戻り値:

セットに指定された要素が含まれていた場合、true。

使用可能場所:**removeAll**

function removeAll(set: set<ItemType>, elements: collection<ItemType>): boolean

セットから、指定されたコレクションに含まれるすべての要素を削除します。

パラメーター:

set: 要素を削除するセット。

elements: セットから削除する要素を含むコレクション。

戻り値:

呼び出しの結果、セットが変更された場合、true。

使用可能場所:

retainAll

function retainAll(set: set<ItemType>, elements: collection<ItemType>): boolean

セット内で、指定されたコレクションに含まれる要素のみを保持します。

パラメーター:

set: 変更するセット。

elements: セットに保持する要素を含むコレクション。

戻り値:

呼び出しの結果、セットが変更された場合、true。

使用可能場所:**size**

function size(set: set<ItemType>): integer

セット内の要素数を返します。

パラメーター:

set: 入力セット。

戻り値:

セット内の要素数。セットがnullの場合は0。

使用可能場所:**toList**

function toList(set: set<ItemType>): list<ItemType>

セットを、同じ要素を含むリストに変換し、順序を保持します。

パラメーター:

set: 入力セット。

戻り値:

セットの要素を反復順序で含むリスト。

使用可能場所:

union

function union(set1: set<ItemType>, set2: set<ItemType>): set<ItemType>

両方のセットのすべての要素を含む新しいセットを返します (和集合)。

パラメーター:

set1: 最初のセット。

set2: 2番目のセット。

戻り値:

両方のセットのすべての要素を含む新しいセット。両方のパラメーターがnullの場合は空のセットを返します。

使用可能場所:

ユニット core.string

文字列操作機能を提供するスクリプトユニット。

メソッド
function capitalize(str: string): string 文字列の最初の文字を Character.toUpperCase(Number) に従ってタイトルケースに変更して大文字にします。
function contains(value: string, searchString: string): boolean 値文字列に searchString が含まれている場合は true を返します。
function endsWith(value: string, suffix: string): boolean 値文字列がサフィックスで終わる場合は true を返します。
function format(format: string, args: string[]): string 指定された書式文字列と引数を使用して、書式設定された文字列を返します。
function formatWithLocale(format: string, args: string[]): string 指定されたロケールで、指定された書式文字列と引数を使用して、書式設定された文字列を返します。
function indexOf(str: string, searchStr: string): number null を処理して、文字列内の最初のインデックスを検索します。

メソッド

`function indexOfFromIndex(str: string, searchStr: string, fromIndex: number): number` null を処理して、指定されたインデックスから始まる文字列内の最初のインデックスを検索します。

`function isAlphanumeric(str: string): boolean` 文字列が英数字であるかどうかを確認します。

`function isBlank(str: string): boolean` 文字列が空 ("")、null、または空白のみであるかどうかを確認します。

`function isEmpty(value: string): boolean` 値が null または長さがゼロの文字列の場合は true を返します。

`function isNumeric(str: string): boolean` 文字列に Unicode 数字のみが含まれているかどうかを確認します。

`function join(delimiter: string [, substring: string]...): string` 区切り文字列として区切られたすべての入力サブ文字列で構成される新しい文字列を返します。

`function joinExtended(delimiter: string, prefix: string, suffix: string [, substring: string]...): string` 区切り文字で区切られ、プレフィックスとサフィックスが付いたすべての入力サブ文字列で構成される新しい文字列を返します。

`function joinList(delimiter: string, prefix: string, suffix: string, list: list): string` 区切り文字で区切られ、プレフィックスとサフィックスが付いたすべてのリスト項目で構成される新しい文字列を返します。

`function lastIndexOf(str: string, searchStr: string): number` 指定されたサブ文字列が文字列内で最も右に現れるインデックスを返します。

`function lastIndexOfFromIndex(str: string, searchStr: string, fromIndex: number): number` 指定されたインデックスから逆方向に検索して、この文字列内で指定されたサブ文字列が最後に現れるインデックスを返します。

`function replaceAll(value: string, pattern: string, replacement: string): string` 入力値文字列内のすべてのパターン出現を置換文字列に置き換えた新しい文字列を返します。

`function replaceFirst(value: string, pattern: string, replacement: string): string` 入力値文字列内の最初のパターン出現を replacement に置き換えた新しい文字列を返します。

`function reverse(str: string): string` 文字列を反転します。

`function right(value: string, index: decimal): string` 値文字列から正しい文字を含む新しい文字列を返します。

メソッド

`function split(str: string): string[]` 指定されたテキストを、空白を区切り文字として配列に分割します。

`function split(str: string, separator: string): string[]` 指定されたテキストを、指定された区切り文字で配列に分割します。

`function startsWith(value: string, prefix: string): boolean` 値文字列がプレフィックスで始まる場合は true を返します。

`function substring(value: string, startIndex: decimal, endIndex: decimal): string` 値文字列の文字の部分シーケンスを含む新しい文字列を返します。

`function toLowerCase(value: string): string` 入力値文字列のすべての文字を小文字にした新しい文字列を返します。

`function toUpperCase(value: string): string` 入力値文字列のすべての文字を大文字にした新しい文字列を返します。

`function trim(str: string): string` この文字列の両端から制御文字 (char <= 32) を削除し、null の場合は null を返します。

`function valueOf(obj: Object): string` Object 引数の文字列表現を返します。

- capitalize
- contains
- endsWith
- format
- formatWithLocale
- indexOf
- indexOfFromIndex
- isAlphanumeric
- isBlank
- isEmpty
- isNumeric
- join
- joinExtended

- `joinList`
- `lastIndexOf`
- `lastIndexOfFromIndex`
- `replaceAll`
- `replaceFirst`
- `reverse`
- `right`
- `split`
- `splitWith`
- `startsWith`
- `substring`
- `toLowerCase`
- `toUpperCase`
- `trim`
- `valueOf`

capitalize

function capitalize(str: string): string

文字列の最初の文字を `Character.titleCase(Number)` に従ってタイトルケースに変更して大文字にします。

例:

```
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

```

パラメーター

str: 入力文字列

戻り値

大文字にされた文字列

使用可能な場所：

contains

function contains(value: string, searchString: string): boolean

値文字列に searchString が含まれている場合は true を返します。

パラメーター

value: 入力文字列

searchString: 検索する入力 searchString

戻り値

値に searchString が含まれている場合は true、含まれていない場合は false。

いずれかのパラメーターが null の場合、null を返します。

使用可能な場所：

endsWith

function endsWith(value: string, suffix: string): boolean

値文字列がサフィックスで終わる場合は true を返します。

パラメーター

value: 入力文字列

prefix: 検索する入力サフィックス

戻り値

値がサフィックスで終わる場合は true、そうでない場合は false。

いずれかのパラメーターが null の場合、null を返します。

使用可能な場所：

format

function format(format: string, args: string): string

指定された書式文字列と引数を使用して、書式設定された文字列を返します。これは Locale.US を使用します。

例:

```
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

パラメーター

format: 書式文字列

args: 書式文字列内の書式指定子によって参照される引数

戻り値

書式設定された文字列

使用可能な場所：

formatWithLocale

function formatWithLocale(format: string, args: string): string

指定されたロケールで、指定された書式文字列と引数を使用して、書式設定された文字列を返します。

例:

```
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
```

パラメーター

local: 書式設定に使用されるロケール

format: 書式文字列

args: 書式文字列内の書式指定子によって参照される引数

戻り値

書式設定された文字列

使用可能な場所：

indexOf

function indexOf(str: string, searchStr: string): number

null を処理して、文字列内の最初のインデックスを検索します。

例:

```
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
```

パラメーター

str: 入力文字列

searchStr: 検索するサブ文字列

戻り値

文字列内でサブ文字列が最初に現れるインデックス。見つからない場合は -1

使用可能な場所：

indexOfFromIndex

function indexOfFromIndex(str: string, searchStr: string, fromIndex: number): number

null を処理して、指定されたインデックスから始まる文字列内の最初のインデックスを検索します。

例:

```
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
```

パラメーター

str: 入力文字列

searchStr: 検索するサブ文字列

fromIndex: 検索を開始するインデックス

戻り値

文字列内でサブ文字列が最初に現れるインデックス。見つからない場合は -1

使用可能な場所：

isAlphanumeric

function isAlphanumeric(str: string): boolean

文字列が英数字であるかどうかを確認します。

例:

```

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

```

パラメーター

str: 入力文字列

戻り値

文字列が英数字の場合は true、それ以外の場合は false

使用可能な場所：

isBlank

function isBlank(str: string): boolean

文字列が空 (""), null、または空白のみであるかどうかを確認します。

例:

```
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
```

パラメーター

str: 入力文字列

戻り値

文字列が空、null、または空白のみの場合は true、それ以外の場合は false

使用可能な場所：

isEmpty

function isEmpty(value: string): boolean

値が null または長さがゼロの文字列の場合は true を返します。

パラメーター

value: 文字列値。

戻り値

値が null または長さがゼロの文字列の場合は true、それ以外の場合は false。null を返すことはありません。

使用可能な場所：

isNumeric

function isNumeric(str: string): boolean

文字列に Unicode 数字のみが含まれているかどうかを確認します。このメソッドは、正または負の先行符号を許可しないことに注意してください。

例:

```
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
```

パラメーター

str: 入力文字列

戻り値

文字列に Unicode 数字のみが含まれている場合は true、それ以外の場合は false

使用可能な場所：

join

function join(delimiter: string ...): string

区切り文字列として区切られたすべての入力サブ文字列で構成される新しい文字列を返します。

パラメーター

delimiter: 入力区切り文字列

substring: 結果文字列に追加するサブ文字列。すべてのエレメントは文字列にする必要があります。

戻り値

区切り文字で区切られたすべての入力サブ文字列で構成される新しい文字列。

区切り文字が null の場合は null を返します。null のサブ文字列はすべて文字列 'null' に置き換えられます。

使用可能な場所：

joinExtended

function joinExtended(delimiter: string, prefix: string, suffix: string ...): string

区切り文字で区切られ、プレフィックスとサフィックスが付いたすべての入力サブ文字列で構成される新しい文字列を返します。

パラメーター

delimiter: 入力区切り文字列。

prefix: 入力プレフィックス文字列。

suffix: 入力サフィックス文字列。

substring: 結果文字列に追加するサブ文字列。すべてのエレメントは文字列にする必要があります。

戻り値

区切り文字で区切られたすべての入力サブ文字列で構成される新しい文字列。

区切り文字、プレフィックス、またはサフィックスが null の場合は null を返します。null のサブ文字列はすべて文字列 'null' に置き換えられます。

使用可能な場所：

joinList

function joinList(delimiter: string, prefix: string, suffix: string, list: list): string

区切り文字で区切られ、プレフィックスとサフィックスが付いたすべてのリスト項目で構成される新しい文字列を返します。

パラメーター

delimiter: 入力区切り文字列。

prefix: 入力プレフィックス文字列。

suffix: 入力サフィックス文字列。

list: 結果文字列に追加する文字列のリスト。

戻り値

入力区切り文字で区切られたすべての入力サブ文字列で構成される新しい文字列。

いずれかのパラメーターが null の場合は null を返します。リストの null 項目はすべて文字列 'null' に置き換えられます。

使用可能な場所：

lastIndexOf

function lastIndexOf(str: string, searchStr: string): number

指定されたサブ文字列が文字列内で最も右に現れるインデックスを返します。最も右の空文字列 "" は、インデックス値 str.length() で現れると見なされます。

例:

```
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

```

パラメーター

str: 入力文字列

searchStr: 検索するサブ文字列

戻り値

文字列内でサブ文字列が最後に現れるインデックス。見つからない場合は -1

使用可能な場所：

lastIndexOfFromIndex

function lastIndexOfFromIndex(str: string, searchStr: string, fromIndex: number): number

指定されたインデックスから逆方向に検索して、この文字列内で指定されたサブ文字列が最後に現れるインデックスを返します。

例:

```

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

```

パラメーター

str: 入力文字列

searchStr: 検索するサブ文字列

fromIndex: 検索を開始するインデックス

戻り値

文字列内でサブ文字列が最後に現れるインデックス。見つからない場合は -1

使用可能な場所：

replaceAll

function replaceAll(value: string, pattern: string, replacement: string): string

入力値文字列内のすべてのパターン出現を置換文字列に置き換えた新しい文字列を返します。

パラメーター

value: 入力文字列

pattern: 置き換える文字列パターン (正規表現)

replacement: 置換文字列

戻り値

新しい文字列。パラメーターが null の場合は null。

使用可能な場所：

replaceFirst

function replaceFirst(value: string, pattern: string, replacement: string): string

入力値文字列内の最初のパターン出現を replacement に置き換えた新しい文字列を返します。

パラメーター

value: 入力文字列

pattern: 置き換える文字列パターン (正規表現)

replacement: 置換文字列

戻り値

新しい文字列。いずれかのパラメーターが null の場合は null。

使用可能な場所：

reverse

function reverse(str: string): string

文字列を反転します。

例:

```
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
```

パラメーター

str: 入力文字列

戻り値

反転された文字列

使用可能な場所：

right

function right(value: string, index: decimal): string

値文字列から正しい文字を含む新しい文字列を返します。

指定されたインデックスから始まり、文字列の終わりまで続きます。

パラメーター

value: 入力文字列

index: 入力開始インデックス (含む)。0以上、文字列の長さ以下の整数にする必要があります。

戻り値

新しい文字列。いずれかのパラメーターが null の場合は null。

使用可能な場所：

split

function split(str: string): string

指定されたテキストを、空白を区切り文字として配列に分割します。

例:

```
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
```

パラメーター

str: 入力文字列

戻り値

サブ文字列の配列

使用可能な場所：

splitWith

function split(str: string, separator: string): string

指定されたテキストを、指定された区切り文字で配列に分割します。

例:

```
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
```

パラメーター

str: 入力文字列

separator: 区切り文字列

戻り値

サブ文字列の配列

使用可能な場所：

startsWith

function startsWith(value: string, prefix: string): boolean

値文字列がプレフィックスで始まる場合は true を返します。

パラメーター

value: 入力文字列

prefix: 検索する入力プレフィックス

戻り値

値がプレフィックスで始まる場合は true、そうでない場合は false。

いずれかのパラメーターが null の場合、null を返します。

使用可能な場所：

substring

function substring(value: string, startIndex: decimal, endIndex: decimal): string

値文字列の文字の部分シーケンスを含む新しい文字列を返します。

サブ文字列は、指定された startIndex から始まり、入力文字列の endIndex まで拡張されます。

パラメーター

value: 入力文字列

startIndex: 入力開始インデックス (含む)。0 以上、文字列の長さ以下の整数にする必要があります。

endIndex: 入力終了インデックス (含まない)。startIndex 以上、文字列の長さ以下の整数にする必要があります。

戻り値

新しい文字列。いずれかのパラメーターが null の場合は null。

使用可能な場所：

toLowerCase

function toLowerCase(value: string): string

入力値文字列のすべての文字を小文字にした新しい文字列を返します。

パラメーター

value: 入力文字列

戻り値

小文字の文字列。パラメーターが null の場合は null。

使用可能な場所：

toUpperCase

function toUpperCase(value: string): string

入力値文字列のすべての文字を大文字にした新しい文字列を返します。

パラメーター

value: 入力文字列

戻り値

大文字の文字列。パラメーターが null の場合は null。

使用可能な場所：

trim

function trim(str: string): string

この文字列の両端から制御文字 (char <= 32) を削除し、null の場合は null を返します。

例:

```
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;
```

```
end
```

パラメーター

str: 入力文字列

戻り値

トリムされた文字列

使用可能な場所：

valueOf

function valueOf(obj: Object): string

Object 引数の文字列表現を返します。

例:

```
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
```

パラメーター

obj: 文字列に変換するオブジェクト

戻り値

オブジェクトの文字列表現

使用可能な場所：

ユニット core.time

時間関数を提供するスクリプトユニットです。

メソッド
<code>function addHours(value: time, hours: decimal): time</code> 時刻に時間を追加します。
<code>function addMinutes(value: time, minutes: decimal): time</code> 時刻に分を追加します。
<code>function addSeconds(value: time, seconds: decimal): time</code> 時刻に秒を追加します。
<code>function fromDatetime(value: datetime): datetime</code> 日付時刻を時刻に変換します。
<code>function fromIsoString(value: string): date</code> ISO 形式 (yyyy-MM-dd'T'HH:mm:ss.) の文字列から時刻を解析します。
<code>function now(): time</code> 現在の時刻を返します。
<code>function of(hours: decimal, minutes: decimal, day: decimal): time</code> 時間、分、秒で指定された時刻を作成します。
<code>function seconds(startValue: time, endValue: time): decimal</code> 2 つの時刻の間の秒数を返します。
<code>function toIsoString(value: date): string</code> 時刻を ISO 形式 (HH:mm:ss.) の文字列に変換します。

- `addHours`
- `addMinutes`
- `addSeconds`
- `fromDatetime`
- `fromIsoString`
- `now`
- `of`
- `seconds`
- `toIsoString`

`addHours`

`function addHours(value: time, hours: decimal): time`

時刻に時間を追加します。計算は深夜に終了します。

パラメーター:

value: 入力時間

hours: 追加される時間数。整数にする必要があります。

戻り値:

新しい時刻。パラメーターが null の場合は null。

使用可能な場所:

addMinutes

function addMinutes(value: time, minutes: decimal): time

時刻に分を追加します。計算は深夜に終了します。

パラメーター:

value: 入力時間

minutes: 追加する分数。整数にする必要があります。

戻り値:

新しい時刻。パラメーターが null の場合は null。

使用可能な場所:

addSeconds

function addSeconds(value: time, seconds: decimal): time

時刻に秒を追加します。秒の小数部はミリ秒の精度でサポートされています。計算は深夜に終了します。

パラメーター:

value: 入力時間

seconds: 追加される秒数。小数部は小数点以下 3 桁に丸められます。

戻り値:

新しい時刻。パラメーターが null の場合は null。

使用可能な場所:

fromDatetime

function fromDatetime(value: datetime): datetime

日付と時刻を時刻に変換します。日付部分は無視されます。

パラメーター:

value: 入力日時

戻り値:

新しい時刻。パラメーターが null の場合は null。

使用可能な場所:

fromIsoString

function fromIsoString(value: string): date

ISO 形式 (yyyy-MM-dd'T'HH:mm:ss.SSS) の文字列から時刻を解析します。ミリ秒部分はオプションです。

パラメーター:

value: 入力文字列

戻り値:

日付。値が null または形式が無効な場合は null。

使用可能な場所:

now

function now(): time

現在の時刻を返します。

戻り値:

現在の時刻。

使用可能な場所:

of**function of(hours: decimal, minutes: decimal, day: decimal): time**

時間、分、秒で指定された時刻を作成します。秒の小数部はミリ秒の精度でサポートされています。

パラメーター:

hour: 入力時間

minute: 入力分

second: 入力秒。小数部は小数点以下 3 桁に丸められます。

戻り値:

新しい時刻。パラメーターが null の場合は null。

使用可能な場所:**seconds****function seconds(startValue: time, endValue: time): decimal**

2 つの時刻の間の秒数を返します。

パラメーター:

startValue: 入力開始日時 (含む)

endValue: 入力終了日時 (含まない)

戻り値:

秒数。パラメーターが null の場合は null。戻り値には小数部分が含まれる場合があります。

使用可能な場所:**toIsoString****function toIsoString(value: date): string**

時刻を ISO 形式 (HH:mm:ss.SSS) の文字列に変換します。

パラメーター:

value: 入力時刻

戻り値:

文字列。値が null の場合は null。

使用可能な場所:**ユニット core.uri**

URI (Uniform Resource Identifier) 参照を管理するための関数を提供するスクリプトユニットです。

メソッド

`function of(value: string): uri` 文字列から URI 参照を作成します。

of**function of(value: string): uri**

文字列から URI 参照を作成します。

パラメーター

value: 入力値。たとえば「https://www.tibco.com」。

戻り値

URI。

使用可能な場所**ユニット core.validation**

バリデーション制約のコンテキストで依存関係を設定するためのメソッドを提供するスクリプトユニット。

メソッド

`procedure addDependencyToInsertAndDelete(path: string)` 現在の制約のバリデーション結果が、指定されたテーブルパスの挿入と削除に依存することを指定する依存関係パスを追加します。

`procedure addDependencyToInsertAndDeleteAndModifyInOtherDataset(path: string, dataset: string, dataspace: string)` 現在の制約のバリデーション結果が、別のデータセット内のパスで指定されたテーブルまたはテーブルフィールドの挿入、変更、および削除に依存することを指定する依存関係パスを追加します。

メソッド

`procedure addDependencyToInsertAndDeleteInOtherDataset(path: string, dataset: string, dataspace: string)` 現在の制約のバリデーション結果が、特定のデータスペース内の指定されたデータセットの指定されたテーブルの挿入と削除に依存することを指定する依存関係パスを追加します。

`procedure addDependencyToInsertDeleteAndModify(path: string)` 現在の制約のバリデーション結果が、パスで指定されたテーブルまたはテーブルフィールドの挿入、変更、および削除に依存することを明示的に宣言する依存関係パスを追加します。

`procedure addDependencyToModify(path: string)` 現在の制約のバリデーション結果が、パスで指定されたフィールドの変更に依存することを明示的に宣言する依存関係パスを追加します。

`procedure disableAjaxPrevalidation()` ユーザーインターフェースで、関連フィールドがユーザー入力で更新されたときに、制約が動的に事前バリデーションされるのを防ぎます。

`procedure setBlocksCommitToNever()` バリデーションエラーを追加する場合でも、現在の制約が操作をブロックしてはならないことを指定します。

`procedure setDependencyToLocalNode()` 現在の制約のバリデーション結果が、ノードのローカル値のみに依存することを指定します。

- `addDependencyToInsertAndDelete`
- `addDependencyToInsertAndDeleteAndModifyInOthe...`
- `addDependencyToInsertAndDeleteInOtherDataset`
- `addDependencyToInsertDeleteAndModify`
- `addDependencyToModify`
- `disableAjaxPrevalidation`
- `setBlocksCommitToNever`
- `setDependencyToLocalNode`

addDependencyToInsertAndDelete

procedure addDependencyToInsertAndDelete(path: string)

現在の制約のバリデーション結果が、指定されたテーブルパスの挿入と削除に依存することを指定する依存関係パスを追加します。スクリプト公開時に指定されたパスでテーブルパスが見つからない場合、エラーが発生します。

このメソッドは、制約のプロシージャセットアップから呼び出す必要があります。そうでない場合、無視されます。

例:

```
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

    end

export procedure setup()

begin

    v.addDependencyToInsertDelete('/root/ProductCatalog');

end
```

パラメーター:

path: テーブルパス。

使用可能な場所:

addDependencyToInsertAndDeleteAndModifyInOthe...

procedure addDependencyToInsertAndDeleteAndModifyInOtherDataset(path: string, dataset: string, dataspace: string)

現在の制約のバリデーション結果が、別のデータセット内のパスで指定されたテーブルまたはテーブルフィールドの挿入、変更、および削除に依存することを指定する依存関係パスを追加します。スクリプト公開時に指定されたデータセットでテーブルパスまたはテーブルフィールドパスが見つからない場合、エラーが発生します。

このメソッドは、制約のプロシージャセットアップから呼び出す必要があります。そうでない場合、無視されます。

例:

```
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
```

パラメーター:

path: フィールドパス。

dataset: 他のデータセット。

dataspace: 他のデータスペース。

使用可能な場所:

addDependencyToInsertAndDeleteInOtherDataset

procedure addDependencyToInsertAndDeleteInOtherDataset(path: string, dataset: string, dataspace: string)

現在の制約のバリデーション結果が、特定のデータスペース内の指定されたデータセットの指定されたテーブルの挿入と削除に依存することを指定する依存関係パスを追加します。スクリプト公開時に、パスがテーブルパスでない場合、または指定されたデータセットとデータスペースで見つからない場合、エラーが発生します。

このメソッドは、制約のプロシージャセットアップから呼び出す必要があります。そうでない場合、無視されます。

例:

```
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

```

パラメーター:

path: テーブルパス。

dataset: 他のデータセット名。

dataspace: 他のデータスペース名。

使用可能な場所:

addDependencyToInsertDeleteAndModify

procedure addDependencyToInsertDeleteAndModify(path: string)

現在の制約のバリデーション結果が、パスで指定されたテーブルまたはテーブルフィールドの挿入、変更、および削除に依存することを明示的に宣言する依存関係パスを追加します。スクリプト公開時に指定されたパスでテーブルパスまたはテーブルフィールドパスが見つからない場合、エラーが発生します。

このメソッドは、制約のプロシージャセットアップから呼び出す必要があります。そうでない場合、無視されます。

例:

```

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

end

export procedure setup()

begin

    v.addDependencyToInsertDeleteAndModify('/root/Employee/number');

end

```

パラメーター:

path: テーブルパスまたはテーブルフィールドパス。

使用可能な場所:**addDependencyToModify****procedure addDependencyToModify(path: string)**

現在の制約のバリデーション結果が、パスで指定されたフィールドの変更に依存することを明示的に宣言する依存関係パスを追加します。スクリプト公開時に、フィールドにローカル値がない場合、または指定されたパスで見つからない場合、エラーが発生します。

このメソッドは、制約のプロシージャセットアップから呼び出す必要があります。そうでない場合、無視されます。

例:

```
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

end

export procedure setup()

begin

    v.addDependencyToModify('/root/code');

end
```

パラメーター:

path: フィールドパス。

使用可能な場所:

disableAjaxPrevalidation

procedure disableAjaxPrevalidation()

ユーザーインターフェースで、関連フィールドがユーザー入力で更新されたときに、制約が動的に事前バリデーションされるのを防ぎます。これは、パフォーマンスを向上させたり、フォーム更新中の不要なバリデーションエラーを回避したりするのに役立ちます。ただし、フォームが送信されたとき、またはビューでレコードがロードされたときには、制約は引き続きバリデーションされます。

例:

```
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
```

使用可能な場所:

setBlocksCommitToNever

procedure setBlocksCommitToNever()

バリデーションエラーを追加する場合でも、現在の制約が操作をブロックしてはならないことを指定します。

例:

```
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
```

使用可能な場所:

setDependencyToLocalNode

procedure setDependencyToLocalNode()

現在の制約のバリデーション結果が、ノードのローカル値のみに依存することを指定します。

例:

```
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
```

使用可能な場所:

ユニット **core.rest**

RESTサービス作成時に役立つメソッドを提供するスクリプトユニット。

メソッド
<code>function emptyResponse(status: int): rest_response</code> ボディなしのレスポンスを作成します。
<code>function responseOf(status: int, content: any): rest_response</code> オプションのHTTPステータスを持つJSONレスポンスを作成します。

- `emptyResponse`
- `responseOf`

emptyResponse

function emptyResponse(status: int): rest\response

ボディなしのレスポンスを作成します。

パラメーター:

status: ステータスコード。nullの場合、204が想定されます。

戻り値:

レスポンス。

使用可能場所:

responseOf

function responseOf(status: int, content: any): rest\response

オプションのHTTPステータスを持つJSONレスポンスを作成します。

パラメーター:

status: ステータスコード。nullの場合、200が想定されます。

content: シリアル化するコンテンツ。JSONスキーマからの型付きオブジェクト、汎用 json_value、およびこれらの型のリスト（多次元を含む）をサポートします。nullの場合、ボディはJSON値のnullです。

戻り値:

JSONレスポンス。

使用可能場所:

EBX® Script IDE

- [EBX® Script IDE の概要](#)
- [基本操作ガイド](#)
- [チュートリアル](#)
 - [関数の作成](#)
 - [トリガーの作成](#)

- ワークフロースクリプトの作成
- スクリプト化された REST サービスの作成チュートリアル
- ユーザーインターフェイスリファレンス

EBX® Script IDE の概要

ビルトインの EBX® パースペクティブとしてアクセスできる Script IDE を使用すると、ブラウザで直接 EBX® スクリプト言語 (ESL) を使用して関数、トリガー、ワークフロースクリプト、およびスクリプト化された REST サービスを作成できます。IDE は、ユーザーのアクセシビリティ、利便性、効率性において大きなメリットをもたらします。また、前述のタスクに Java を使用する必要性も軽減します。同じパフォーマンスを維持しながら、カスタムモジュールのコンパイルと展開が不要なため、オーバーヘッドが少なくなります。さらに、ESL の設計により、Java プログラミングの経験がないユーザーでもアクセスできます。

IDE のドキュメントは次のように構成されています。

- 『[基本操作ガイド](#)』：『[クイックスタートガイド](#)』と同様に、このガイドでは一般的なタスクを実行するための高度な手順を説明します。このセクションは、プログラミング経験がある方、または ESL にすでに慣れている方を対象としています。
- チュートリアル：チュートリアルは基本的に、プログラミング経験があまりないユーザー向けのレッスンであり、次のプロセスをガイドします。
 - 関数
 - トリガー
 - ワークフロースクリプト
 - スクリプト化された REST サービス
- [ユーザーインターフェイスリファレンス](#)：このセクションでは、IDE の UI コンポーネントについて説明します。

基本操作ガイド

- 概要
- 関数の作成
- ワークフロースクリプトタスクの作成と管理
- トリガーの作成
- スクリプト化された REST サービスの作成

- 履歴を使用した変更のロールバック
- スクリプトログの確認

概要

このセクションは、より経験豊富な TIBCO EBX® ユーザーを対象としており、EBX® Script IDE に関連する一般的なタスクに関する概要情報を提供します。追加のドキュメントリソースについては、次のとおりです。

- ドキュメントの *管理ガイド* > *EBX® スクリプト* セクションでは、EBX® スクリプト言語 (ESL) の詳細について説明します。
- *管理ガイド* > *EBX® スクリプト IDE* > *チュートリアル* セクションでは、経験の浅いユーザー向けに設計されたいくつかの基本的な使用例に関する詳細情報について説明します。

関数の作成

EBX® Script IDE で関数を記述する前に、まず関数タイプを使用してデータモデルフィールドを作成する必要があります。フィールドの作成後、スクリプトはスクリプト IDE で編集できるようになります。

データモデルに関数型フィールドを追加するには、次の手順を実行します。

1. 新しいデータモデルフィールドを作成し、を に設定します。
2. フィールドを使用して、必要なデータ型を指定します。選択したデータ型は、IDE のスクリプトに自動的に反映されます。
3. データモデルを保存し、必要に応じて公開します。フィールドを保存すると、スクリプトアウトラインが自動的に作成され、IDE で使用できるようになります。ただし、データモデルの公開が最新になるまでスクリプトを公開することはできません。そのため、この時点でデータモデルを公開することをお勧めします。
4. EBX® Script IDE パースペクティブが開いていない場合:
 - a. EBX® ツールバーの右上にある  アイコンをクリックし、を選択します。
 - b. 左側のドックの で、を選択します。
5. 関数が追加されたデータモデルを展開し、スクリプトを記述して公開します。

関連項目:

- 関数の作成チュートリアル
- 関数フィールド

ワークフロースクリプトタスクの作成と管理

ワークフロースクリプトタスクの作成は、スクリプト IDE で直接行うことができます。ワークフロースクリプトが公開されると、ワークフローモデルで使用できるようになります。

ワークフロースクリプトを作成するには、次の手順を実行します。

1. EBX® Script IDE パースペクティブが開いていない場合:
 - a. EBX® ツールバーの右上にある  アイコンをクリックし、を選択します。
 - b. 左側のドックの で、を選択します。
2. 新しいスクリプトを作成します。
3. 必要な情報とパラメーターを追加します。
 - a. スクリプト名、ラベル、およびオプションの説明を入力します。
 - b. データスペース、データセット、テーブル、レコードなどの EBX® コンポーネントにアクセスするための入力/出力として使用されるパラメーターを追加します。
4. スクリプトを記述したら、タブの下部にある ボタンを選択します。

次の操作を実行すると、ワークフローモデルタスクでスクリプトを選択できるようになります。

- a. メニューから を選択し、をクリックします。
- b. の場合は、を に設定し、メニューを使用してワークフロースクリプトを選択します。

関連項目:

- ワークフロースクリプトの作成チュートリアル
- スクリプトタスク

トリガーの作成

EBX® Script IDE でトリガーのロジックを記述する前に、まずデータモデルにトリガーを追加する必要があります。データモデルでトリガーを作成すると、EBX® Script IDE で自動的に使用できるようになります。

データモデル内のテーブルにトリガーを追加するには、次の手順を実行します。

1. データモデルを開き、目的のデータモデルエレメントの タブを選択します。
2. **[EBX スクリプト]** コンポーネントを使用するトリガーを追加して構成します。
3. データモデルを保存し、必要に応じて公開します。トリガーを追加した後にテーブルを保存すると、トリガーのアウトラインが自動的に作成され、使用できるようになります。ただし、データモデルの公開が最新になるまでスクリプトを公開することはできません。そのため、この時点でデータモデルを公開することをお勧めします。
4. EBX® Script IDE パースペクティブが開いていない場合:
 - a. EBX® ツールバーの右上にある  アイコンをクリックし、 を選択します。
 - b. 左側のドックの で、 を選択します。
5. トリガーが追加されたデータモデルを展開し、スクリプトを記述して公開します。

関連項目:

- トリガーの作成チュートリアル
- テーブルトリガー
- データセットトリガー

スクリプト化された REST サービスの作成

スクリプト化された REST サービスは、REST エンドポイントと、そのエンドポイントにアクセスしたときに実行されるスクリプト化されたロジックの組み合わせです。

Script IDE でスクリプト化された REST サービスを作成するには、次の手順を実行します。

1. EBX® Script IDE パースペクティブが開いていない場合:
 - a. EBX® ツールバーの右上にある  アイコンをクリックし、 を選択します。
 - b. 左側のドックの で、 を選択します。

2. REST エンドポイントを作成するには、次のいずれかの方法を使用します。
 - a. タブと タブを使用して、エンドポイントを手動で定義します。
 - b. 最初の REST サービスを作成するときに、 を選択して、OpenAPI 仕様に準拠する YAML または JSON ファイルで REST API 定義をインポートします。
3. タブを使用して、サービスのプログラムロジックを記述します。

注意:

関数を作成するときは、その名前が関数を呼び出すエンドポイントの と一致する必要があります。

4. 公開後、 オプションを使用して IDE の組み込みテスターを使用します。

関連項目:

- [スクリプト化された REST サービスの作成チュートリアル](#)
- [スクリプト化された REST サービス](#)

履歴を使用した変更のロールバック

IDE 内の各スクリプトと API には、すべてのリビジョンの履歴が含まれています。各リビジョンは、スクリプトが保存されるときに作成されます。

スクリプトの変更を表示するには、次の手順を実行します。

1. EBX® Script IDE パースペクティブが開いていない場合は、EBX® ツールバーの右上にある  アイコンをクリックし、 を選択します。
2. 左側のドックでスクリプトタイプを選択します。
3. ワークスペースの上部にある タブまたは タブを選択します。
4. 列を使用して変更を参照し、現在の 列にコピー/貼り付けまたは編集を行います。
5. タブの下部にあるボタンを使用して、スクリプトを保存、コンパイル、公開します。

スクリプトログの確認

スクリプトにログ記録を組み込むと、>>> からログ出力にアクセスできます。

チュートリアル

- 関数の作成
- トリガーの作成
- ワークフロースクリプトの作成
- スクリプト化された REST サービスの作成チュートリアル

関数の作成

- 関数の概要
 - インデックス付き関数とインデックスなし関数
 - このチュートリアルの構成
- 関数の前提条件：データモデルへの関数の追加
- 関数の作成
- インデックスなし関数の作成

関数の概要

関数は、データに対してアクションを実行するように設計された命令セットを受け取ります。これは、データを入力として受け取り、一連のルールに従って処理し、目的の出力を生成する小さなアプリケーションのようなものだと考えてください。たとえば、関数は2つのフィールドの値を結合したり、一連の数値の平均値を計算したり、特定の条件が true であるかどうかを確認したりします。

関数を使用すると、基礎となる技術的な複雑さを深く理解しなくても、より複雑なデータ操作を効率的に処理することもできます。つまり、データ操作の初心者でも、関数を使用するとデータを効果的に操作および分析できるため、広範な技術的知識を必要とするタスクが軽減されます。

インデックス付き関数とインデックスなし関数

関数がインデックス化されると、関数が操作するフィールドが検索クエリ用に最適化されます。ただし、インデックス化された関数は、マシンのローカル時間や、最適化された検索と互換性のないその他の方法にアクセスすることはできません。このチュートリアルには、各タイプの関数の例が含まれています。

このチュートリアルの構成

このチュートリアルでは、EBX® スクリプト IDE で関数を作成して公開する方法を説明します。複雑な使用例やすべてのユーザーインターフェイスオプションについては説明しません。チュートリアルの前提条件セクションで作成した、以下に示す基本データモデルを作成し、赤で強調表示されたフィールドを省略して進めてください。

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

このチュートリアルは、次のパートで構成されています。

1. 関数の前提条件
2. 関数の作成
3. インデックスなし関数の作成

関数の前提条件：データモデルへの関数の追加

EBX® Script IDE で関数を記述する前に、まずデータモデルフィールドを作成し、それが **関数** であることを指定する必要があります。このフィールドには、関数からの出力が表示されます。したがって、このフィールドのデータ型は、関数の予想される戻り値の型 (日付、文字列、整数など) に対応している必要があります。

データモデルに関数型フィールドを追加するには、次の手順を実行します。

1. DMA でデータモデルを開き、次のフィールドを作成します。
 - フィールド 1：このフィールド用に記述された関数は、名フィールドと姓フィールドを連結し、文字列を返します。
 - 名前： `full_name`

- ラベルと説明: `Full name`
 - エLEMENTの種類: `[フィールド]`、チェックボックスをオンにします。
 - データ型: `文字列`
- フィールド 2: このフィールド用に記述された関数は、ユーザーの最終ログインを使用して、そのユーザーがアクティブユーザーであるかどうかを判断します。戻り値/データ型は `ブール値` になります。

- 名前: `status`
- ラベルと説明: `Active user`
- エLEMENTの種類: `[フィールド]`、チェックボックスをオンにします。
- データ型: `ブール値`

2. データモデルを保存し、必要に応じて公開します。フィールドを保存すると、スクリプトのアウトラインが自動的に生成され、使用できるようになります。ただし、データモデルのパブリケーションが最新になるまで、スクリプトを公開することはできません。そのため、この時点でデータモデルを公開することをお勧めします。

次のステップ [インデックス付き](#) 関数または [インデックスなし](#) 関数を作成します。

関数の作成

デフォルトでは、関数が作成されると、インデックスが作成されるように設定されます。そのため、特別な考慮は必要ありません。ただし、タブを選択し、プロパティのチェックがオンに設定されていることを確認することで、関数がインデックスされているかどうかを確認できます。

****スクリプトのヒント:****ビルトイン ESL 関数と変数について理解しておいてください。詳細については、『API リファレンス』を参照してください。

インデックス付け可能な関数を作成するには、次の手順を実行します。

1. EBX® スクリプト IDE パースペクティブが開いていない場合は、EBX® ツールバーの右上にある  アイコンをクリックし、を選択します。
2. 画面左側のアクティビティバーで、を見つけて選択します。
3. 中央のナビゲーションパネルで、上記の前提条件セクションで作成した `Full name` 関数を見つけて選択します。  アイコンは関数を示します。

以下に示すように、関数はフィールドのデータ型 (文字列) に基づいて自動的に入力され、文字列値を返します。

```
export function getValue(): string

begin

    return 'A string value';

end
```

4. この例の関数では、値は次のように計算されます。

- `_ebx.record.first_name` 変数と `_ebx.record.last_name` 変数を使用して、それぞれ Person の名前 (ファーストネーム) と 姓 (ラストネーム) を取得します。これらの変数では、`_ebx.record` は現在のレコードにアクセスするビルトイン関数であり、`.<name>` はデータモデルで取得するフィールドの名前です。追加のビルトイン関数の使用方法の例については、[インデックスなし関数の作成](#) を参照してください。
- スペース文字で区切られた 2 つの値を含む、名前と姓を組み合わせた単一の値を返します。このロジックはすべて、セミコロンで終了する単一の `return` ステートメントに記述されます (パイプ文字は文字列にエレメントを追加します)。

```
export function getValue(): string

begin

    return _ebx.record.first_name | ' ' | _ebx.record.last_name;

end
```

5. スクリプトを保存します。IDE は自動的にコンパイルエラーをチェックし、スクリプトの下でセクションにエラーを表示します。
6. エラーが見つからない場合は、を選択してスクリプトを公開します。

次の画像は、Full name フィールドに値が入力されるスクリプトの結果を示しています。

Person					
+ Actions ▾					
≡	Id	^	First name	Last name	Full name
☐	1		bob	smith	bob smith
☐	2		jim	reynolds	jim reynolds
☐	3		mary	sandall	mary sandall
☐	4		lisa	hempstead	lisa hempstead

インデックスなし関数の作成

この関数の例では、インデックスなしの関数を作成する方法を示します。また、インポートを使用してスクリプトにビルトイン関数を追加する方法も示します。ESL では、関連する関数とプロシージャはユニットにまとめられています。ユニットの関数を使用するには、`uses` `core.<unit_name> as <alias>;` 構文を使用してスクリプトでユニットを宣言します。

****スクリプトのヒント：****コーディングを始める前に、より複雑な関数の擬似コードを書き出してください。この例の擬似コードは次のとおりです：最後のログインから今日までの日数を取得します。この数が 30 より大きい場合は `false` を返し、そうでない場合は (`else`)、アクティブユーザーであることを示す `true` を返します。

インデックスなし関数を作成するには、次の手順を実行します。

1. EBX® スクリプト IDE パースペクティブが開いていない場合は、EBX® ツールバーの右上にある  アイコンをクリックし、を選択します。
2. 画面左側のアクティビティバーで、を見つけて選択します。
3. 中央のナビゲーションパネルで、上記の前提条件セクションで作成した `Active` 関数を見つけて選択します。関数は、この  アイコンで示されます。

以下に示すように、関数はフィールドのブールデータ型に基づいて自動的に設定され、`true` または `false` を返します。

```
export function getValue(): boolean

begin

    return false;
```

```
end
```

4. インデックス作成を無効にします。このスクリプトはマシンのローカル時間にアクセスするため、インデックスを作成できません。
 - a. スクリプトの上部にある タブを選択します。
 - b. チェックボックスをオフにします。
 - c. タブに戻ります。
5. 関数が何を達成するかを決定するロジックを記述します。この例では、関数はユーザーの最終ログイン日と今日の日付を取得し、その差を計算することで、ユーザーがアクティブであるかどうかを判断します。差が 30 日未満の場合、ユーザーはアクティブです。
 - `core.date` ユニットには 2 つの日付間の日数を計算する `days()` 関数が含まれているため、スクリプトの先頭に次の行を追加します。

```
uses core.date as date;
```

- `if then else` ステートメントを使用して、条件が true かどうかを確認します。 `if` は `days()` 関数を使用して、ユーザーの最終ログインからの日数を取得し、 `>` 演算子は日数が 30 より大きいかどうかを確認します。

```
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. スクリプトを保存します。IDE は自動的にコンパイルエラーをチェックし、スクリプトの下でセクションにエラーを表示します。
7. エラーが見つからない場合は、 を選択してスクリプトを公開します。

次の画像は、ユーザーがアクティブであるかどうかを示すためにアクティブユーザーフィールドに値が入力されているスクリプトの結果を示しています。

Person						
+ Actions ▾		Q				
≡	Id	First name	Last name	Full name	Last login	Active user
☐	1	bob	smith	bob smith	12/22/2023	Yes
☐	2	jim	reynolds	jim reynolds	08/08/2023	No
☐	3	mary	sandall	mary sandall	12/08/2023	Yes
☐	4	lisa	hempstead	lisa hempstead	10/04/2023	No

トリガーの作成

- [トリガーの概要](#)
 - [このチュートリアル](#)の構成
- [トリガーの前提条件：データモデルテーブルへのトリガーの追加](#)
- [トリガープロシージャの作成](#)

トリガーの概要

EBX® スクリプト言語 (ESL) では、トリガーは、レコードの変更 (作成、更新、削除) などの特定のイベントに応じて自動的に実行されるスクリプト化された手順です。トリガーを使用すると、ビジネスルールを適用したり、データを検証したり、これらのイベントに合わせて自動化されたタスクを実行したりできます。イベントが発生するたびに条件が満たされるか、アクションが実行されることを確認することで、データの整合性と一貫性が向上します。

このチュートリアル

このチュートリアルでは、EBX® Script IDE でトリガーを作成して公開する方法を説明します。複雑なユースケースやすべてのユーザーインターフェイスオプションについては説明しません。このチュートリアルに沿って、以下に示す基本的なデータモデルを作成できます (このモデルは、[関数チュートリアル](#) で使用したモデルの続きです)。

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

このチュートリアルは、次のパートで構成されています。

- 前提条件の完了
- トリガープロセスの作成

トリガーの前提条件：データモデルテーブルへのトリガーの追加

EBX® Script IDE でトリガーのロジックを記述する前に、まずデータモデルエレメントにトリガーを追加する必要があります。このチュートリアルでは、テーブルにトリガーを追加する方法を説明します。データモデルでトリガーを作成すると、EBX® Script IDE で自動的に使用できるようになります。

データモデルテーブルにトリガーを追加するには、次の手順を実行します。

1. `docPerson` データモデルを開き、`person` テーブルを選択します。
2. タブを選択し、の横にあるアイコンをクリックします。
3. トリガーを設定します。

- トリガーのとして、`[EBX スクリプト]`を選択します。
- トリガーのパラメーター
 - **スクリプト名**：トリガーの名前を入力します。この名前は、IDE がこのトリガーを識別するために使用されます。この例では、「`lastModificationDate`」と入力します。
 - **トリガーのタイプ**：メニューを使用して、目的のトリガータイプを選択します。このチュートリアルでは、を使用します。

4. データモデルを保存し、必要に応じて公開します。トリガーを追加した後にテーブルを保存すると、トリガーのスクリプトアウトラインが自動的に作成され、IDE で使用できるようになります。ただし、データモデルの公開が最新になるまでスクリプトを公開することはできません。そのため、この時点でデータモデルを公開することをお勧めします。

次のステップトリガーされたプロシージャの [スクリプトを作成して公開](#) します。

トリガープロシージャの作成

ESL のプロシージャは値を返さないため、通常はイベントに基づいて一連の指示を実行するトリガーに適しています。この例でスクリプトプロシージャの実行を開始するイベントは、レコードの変更です。目標は、レコードが変更されるたびに、Person テーブルの フィールドに現在の日付を入力することです。

スクリプトのヒント：変数を使用して、時間の経過とともに変化する値、または入力や他の変数に基づいて計算される値を保持します。この例では、変数を使用して、データモデル、テーブル、および PK を指定する値を保持します。これらの値は、レコードを更新するために必要です。構文：`var <variable_name> := <assignment>` を使用して、変数を差k製します。ここで、`var` は `<variable_name>` を使用して変数を宣言し、`:=` は値を割り当てます。値は、直接の値、式、またはメソッド呼び出しの結果にすることができます。

1. EBX® スクリプト IDE パースペクティブが開いていない場合は、EBX® ツールバーの右上にある  アイコンをクリックし、を選択します。
2. 画面左側の **アクティビティバー** で を見つけて選択します。

中央の **ナビゲーションパネル** で **@trigger** フォルダーを見つけ、[前提条件の手順](#) で作成したトリガーを選択します。

以下に示すように、トリガーは前提条件の手順で選択されたトリガーのタイプに基づいており、変更後に実行するように設定されます。

```
export procedure onAfterModify()

begin

end
```

3. プロシージャの指示を記述します。

- a. ユニット `core.data`、`core.date`、および `core.complex` から機能をインポートします。さらに、Person テーブル (この場合は `docPerson`) を含むデータモデルへの参照を追加します。

```
uses core.data as data;

uses core.complex as complex;

uses core.date as date;

references model docPerson;
```

- b. テーブル、テーブルの主キー、およびレコードを格納する変数を使用して、`begin` ステートメントと `end` ステートメントの間に手順の指示を記述します。

- `table` 変数は、`core.data` ユニットの `findTable()` 関数を使用してテーブルを検索します。 `<dataModel.tableName>(_ebx.dataset)` の入力が必要です。この例では、`docPerson` モデルと `person` テーブルを使用します。定義済み変数 `_ebx.dataset` は、イベントがトリガーされるデータセットに関する必要な情報を提供します。

```
var table := data.findTable<docPerson.person>(_ebx.dataset);
```

- `core.complex` ユニットは、主キーなどの複雑なデータ型の操作を容易にします。以下に示すように、テーブルの主キーフィールドにビルトインの `primaryKeyOf` 関数 (`core.complex` から) を使用します。パラメーターとして、`dataModel.tableName` (この場合は `docPerson.person`) を渡します。これにより、値が割り当てられていない PK が作成されます。次のコード行は、PK の id フィールドに、選択したレコードの値を割り当てます。

```
var pk := complex.primaryKeyOf<primarykeyof docPerson.person>();

pk.id := _ebx.record.id;
```

- `core.data` ユニットの `lookupRecordForUpdate` 関数は、変更されたレコードを取得します。ビルトイン変数 `_ebx.transaction` を使用して、現在の EBX® トランザクションをパラメーターとして渡します。次に、上記で定義した `table` 変数と `pk` 変数を渡します。

```
var record := data.lookupRecordForUpdate(_ebx.transaction, table,
pk);
```

c. レコードに変更日を追加して保存します。

- `record.fieldName` を使用して、更新するフィールドを指定します。サンプルモデルのフィールド名は `last\_modification\_date` です。

```
record.last_modification_date := date.now();
```

- 更新されたレコードを保存します。ビルトインの `saveRecord()` 関数を使用し、変更されたレコードを単一のパラメーターとして渡します。

```
data.saveRecord(record);
```

- 完成したプロシージャは次のとおりです。

```
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

```

4. スクリプトを保存します。IDE は自動的にコンパイルエラーをチェックし、スクリプトの下
のセクションにエラーを表示します。
5. エラーが見つからない場合は、 を選択してスクリプトを公開します。

次の画像は、レコードが変更されるたびに最終変更日フィールドが更新されるスクリプトの結果を示しています。

+

Actions

1 - 4 of 4

View

<<

<

>

>>

</

ワークフロースクリプトの作成

- 概要
 - 始める前に
- ステップ 1: スクリプトの記述と公開
- ステップ 2: ワークフローモデルの作成
- ステップ 3: ワークフローランチャーの作成とアクティベーション

概要

EBX® スクリプト IDE から直接ワークフロースクリプトタスクを作成して公開します。スクリプトを公開すると、任意の EBX® ワークフローモデルで使用できます。ワークフローの詳細については、『TIBCO EBX® ユーザーガイド』のセクション **ワークフローモデルの概要** および **データワークフローの概要** を参照してください。

このチュートリアルでは、次の方法を説明します。

- **ステップ 1:** 入力パラメーターと出力パラメーターを含むワークフロースクリプトタスクを作成して公開します。これらのパラメーターにより、スクリプトとの間でデータの受け渡しが可能になります。
- **ステップ 2:** ワークフローモデルを作成し、スクリプトタスクをモデルのステップに追加します。このタスク中に、コンテキスト依存のデータ値をキャプチャするデータ コンテキストがモデル内に作成されます。ワークフローが起動されると、値が設定されます。データコンテキスト内の値は、スクリプトの I/O パラメーターにマップされます。
- **ステップ 3:** ワークフローランチャーを作成してアクティブ化します。EBX® には、ワークフローを開始するさまざまな方法が含まれています。このチュートリアルには、ワークフローランチャーを作成してテーブルのメニューに追加するための簡単な例が含まれています。

始める前に

このチュートリアルでは、[関数の作成](#) および [トリガーの作成](#) チュートリアルと同じデータモデルとスクリプトを使用します。このチュートリアルに従うには、他のチュートリアルを完了して同じ状態のデータモデルを作成するか、モデルのコピーを作成して従ってください。次の画像は、完全なデータモデルを示しています。

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

ステップ 1: スクリプトの記述と公開

前提条件なしで、Script IDE で直接ワークフロースクリプトを作成して記述できます。この例では、ワークフローのデータコンテキストにマップする入力パラメーターをスクリプトに追加する

方法を示します。これにより、データがワークフロースクリプトに渡され、処理されるようになります。

****チュートリアルの使用例：****ビジネスルールでは、Person テーブルからアクティブなユーザーが削除されないようにする必要があります。このテーブルには、ユーザーがアクティブかどうかを示すフィールドが含まれています。ユーザーは、Person レコードを削除するときにワークフローを起動します。スクリプトはユーザーのステータスをチェックします。ユーザーが非アクティブな場合は、レコードが削除されます。ユーザーがアクティブな場合は、レコードを削除できないことを示すメッセージがログに書き込まれます。

****スクリプトのヒント：****定期的にスクリプトを保存してコンパイルすることを習慣にしてください。これにより、コンパイル時のエラーを検出し、保存ポイントを作成して、スクリプトをより安定した状態に戻すことができます。

EBX® スクリプト IDE を使用してワークフロースクリプト タスクを記述するには、次の手順を実行します。

1. EBX® スクリプト IDE パースペクティブが開いていない場合は、EBX® ツールバーの右上にある  アイコンをクリックし、 を選択します。
2. 左側のアクティビティバーで、 を見つけて選択します。
3. 中央のナビゲーションパネルで を選択して、新しいスクリプトを作成します。
4. 必要な情報とパラメーターを追加します。
 - a. スクリプト名 `statusBeforeDelete` を入力します。
 - b. ラベルとオプションの説明 `Check status before delete` を入力します。
 - c. データスペース、データセット、テーブル、レコードなどのデータコンテキストコンポーネントにアクセスするためのパラメーターを追加します。これらの値はスクリプトへの入力を提供し、スクリプトのロジックによって操作されます。

次の各パラメーターを追加し、すべてのオプションをデフォルトのままにします。

- WF_Script_dataspace
- WF_Script_dataset
- WF_Script_table
- WF_Script_requestXPath

5. ワークスペースの下部にある をクリックします。
6. タブでワークフロースクリプトタスクのロジックを記述します。

- a. 次のように、インポートを追加します。

このスクリプトは、EBX® スクリプト言語 (ESL) ユニットに含まれる事前構築済み関数 (`core.data`、`core.complex`、および `core.log`) を活用します。ユニットを宣言する構文は、`uses core.<unit_name>;` です。データモデルへの参照も、このワークフロー例のデータコンテキストの必須部分です。モデルを参照する構文は、`references model <data_model_name>;` です。次の例は、インポートを示しています。

```
uses core.data as data;

uses core.complex as complex;

uses core.log as log;

references model docPerson;
```

- b. ユーザーの選択に基づいて値を保持する定数を作成します。

ユーザーがレコードを選択し、テーブルの **アクション** メニューからワークフローを開始すると、選択されたレコードに関する情報がワークフロー スクリプトの入力パラメーターに渡されます。定数には、データスペース、データセット、テーブル、および選択されたレコードが含まれている必要があります。親子関係として考えてください。データスペースから始めて親を宣言し、子を宣言すると、親が入力として渡されます。次のコードは、`executeScriptTask()` プロシージャの `begin` ステートメントの後に追加されます。

```
// 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);
```

- c. トランザクションを使用して、選択したデータに対してアクションを実行します。

コンテキスト依存データを操作する場合は、トランザクションを使用する必要があります。トランザクションは、影響を受けるデータを分離し、トランザクションが完了した場合にのみ変更を行うことで、データの一貫性を保証します。ESL では、トランザクションはデータスペースで実行され、構文: `execute transaction <transaction_name> on <dataspace>` を使用します。ここで、`<transaction_name>` はトランザクションに割り当てられた変数名、`<dataspace>` はトランザクションが発生するデータスペース名です。トランザクションを開始した後、サンプルコードは次のようになります。

- XPath を使用して現在のレコードを取得します。
- 選択したレコードの主キーを格納するための複合オブジェクトを作成します。
- 現在選択されているレコードの ID を PK の `id` 属性に割り当てます。

```
// 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
```

- d. ユーザーのステータスをチェックし、レコードを削除するか、ログにメッセージを書き込むロジックを記述します。

この例では、`if then else` ステートメントが、選択したレコードの `status` フィールドを渡します。ステータス `false` は、ユーザーが非アクティブであり、

`deleteRecordByPrimaryKey()` 関数を呼び出して削除できることを意味します。

`else` は、ユーザーがアクティブで、情報タイプのログ メッセージを送信する場合にのみ実行されます。EBX® スクリプト ログ メッセージを表示するには、[管理 > スクリプト管理 > スクリプト ログ](#)に移動します。

```
// 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.');
```

完全なワークフロースクリプトタスクの例は、次のとおりです。

```
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. タブ の下部にある ボタンを使用してスクリプトを公開し、ワークフローモデルで使えるようにします。

次のステップ：上記の手順で作成したスクリプトを使用する **ワークフローモデル** を作成します。

ステップ 2: ワークフローモデルの作成

ワークフローモデル定義には、ステップとその実行順序が含まれます。これらのモデルは汎用性が高く、単純な単一ステップのプロセスから複雑な多層シーケンスまで多岐にわたります。このチュートリアルでは、IDE で記述されたスクリプトとワークフローモデル間の相互作用に焦点を当てます。これを実現するために、単純なワークフローモデルの例を使用します。

ワークフローモデルを作成するには、次の手順を実行します。

1. メインの EBX® ツールバーで、  を選択し、  をクリックします。
2. スペースや特殊文字を含まない一意の名前を指定します。この例に従うには、 `deletePerson` を使用します。
3. データコンテキストを作成します。

データコンテキストは、選択したレコードの ID など、コンテキストに依存するデータ値をキャプチャするために使用されます。

- a.  に移動します。
- b. アイコンをクリックして次の変数のリストを追加し、各変数の  を  に設定します。

- WF_Variable_XPath
- WF_Variable_dataspace
- WF_Variable_dataset
- WF_Variable_table

各変数は、ワークフロースクリプトのパラメーターにマップされます。名前はスクリプトのパラメーター名に似たものにする必要がありますが、追加の説明情報を含めることをお勧めします。これにより、ワークフローモデル変数とスクリプトパラメーターを区別しやすくなります。

4. ワークフロースクリプトタスクを実行するステップをワークフローモデルに追加します。

- a. > に移動し、ワークフローダイアグラムに移動します。
- b. と の間の線の上にマウスポインタを移動し、アイコンをクリックします。
- c. タスク名として `deletePersonScript` と入力します。
- d. メニューから を選択し、をクリックします。
- e. スクリプトを設定します。

- i. と はデフォルト値のままにします。

グループの場合は、を に設定します。

- ii. メニューを使用して、`statusBeforeDelete` スクリプトを選択します。

スクリプトの入力パラメーターのリストが表示されます。

- f. 杖アイコンを使用して、各入力パラメーター (ワークフロースクリプトから取得) を、手順 2 で作成したデータコンテキストの対応する変数にマップします。
- g. をクリックします。
- h. ステップを保存して閉じます。
- i. を選択して公開プロセスを開始し、表示される情報が適切であれば、ページの下部にある を選択します。

ステップ 3: ワークフローランチャーの作成とアクティベーション

ワークフローは、ワークフローランチャーによって開始される必要があります。ワークフローランチャーは、いくつかの方法で構成できます。簡潔にするために、この例では、ユーザーがテーブルのメニューから直接起動できるワークフローを作成する方法を示します。これには、ワークフローランチャーの作成とワークフローのアクティブ化が必要です。

ワークフローランチャーとアクティベーションを作成するには、次の手順を実行します。

1. >>> に移動します。
2. 新しいレコードを作成します。
 - a. ランチャー名として、`DeleteInactiveUser` と入力します。

- b. メニューを使用して、`deletePerson` ワークフローモデルパブリケーションを選択します。

グループが下に表示されます。

- c. の場合、テーブルメニューに表示される名前 `Delete an inactive user` を追加します。
- d. データコンテキスト変数を EBX® ビルトインデータコンテキストタイプにマップします。これらの値は、ユーザーが削除するデータを選択してワークフローを起動すると、アクティブなコンテキストに基づいて自動的に入力されます。
- e. 保存しますが、レコードを閉じないでください。

セクションが表示されます。

3. ワークフローランチャーのアクティベーションの作成

ワークフローランチャーのアクティブ化により、ワークフローランチャーのアクセシビリティが決まります。ランチャーは、指定されたテーブルを表示しているときに、選択されたプロファイルで使用できます。

- a. をクリックして、新しいランチャーのアクティベーションを作成します。
- b. `docPerson` データモデルを選択します。
- c. `Person` テーブルを選択します。
- d. の場合は、プロファイルが追加されていることを確認します。
- e. 保存して閉じます。

4. レコードを保存して閉じます。

作成したスクリプトを使用するワークフローは、`Person` テーブルのメニューから起動できるようになりました。

スクリプト化されたRESTサービスの作成チュートリアル

- スクリプト化されたRESTサービスの概要
 - このチュートリアルの構成
- 前提条件
 - スクリプト化されたRESTサービスの構造
 - チュートリアルのデータモデル

- スクリプト化されたRESTサービスの作成
- CRUD操作のスクリプト化されたRESTサービスの例
 - 例の前提条件
 - ユーザーの作成
 - IDによるユーザーの取得
 - ユーザーの更新
 - IDによるユーザーの削除

スクリプト化されたRESTサービスの概要

スクリプト化されたRESTサービスは、RESTエンドポイント定義とそれに関連するスクリプトの組み合わせです。スクリプトはRESTサービスのプログラムによるアクションを処理し、EBX® Script言語で記述されます。EBX® Script IDEで作成されたRESTサービス定義は、OpenAPI Specification (OAS) のサブセットに準拠しています。これにより、OAS v3標準に準拠した定義をインポートおよびエクスポートできます。

実装時には、UIのオプションを使用して以下を定義します。

- パス
- オペレーション
- パスパラメーター
- クエリパラメーター
- ボディとレスポンス
- データ型

各RESTサービススクリプトには、複数のエンドポイントとRESTメソッドのロジックを処理するための複数の関数を含めることができます。スクリプト化されたRESTサービスを実装して公開すると、外部コンシューマーからアクセスできるようになります。JSONリクエストとレスポンスのみがサポートされることに注意してください。

このチュートリアル構成

このチュートリアルでは、EBX® Script IDEでスクリプト化されたRESTサービスを作成および公開する方法を示します。複雑なユースケースや、すべてのユーザーインターフェースオプションについては説明していません。

このチュートリアルを構成するパートは次のとおりです。

- 前提条件の完了
- スクリプト化されたRESTサービスの作成
- スクリプト化されたRESTサービスの例

前提条件

このチュートリアルを開始する前に、以下を確認してください。

- スクリプト化されたRESTサービスの構造
- チュートリアル用のデータモデル

スクリプト化されたRESTサービスの構造

スクリプト化されたRESTサービスには、主に2つのコンポーネントがあります。

- パス、HTTPメソッド (GET、POST、PUT、DELETE)、クエリパラメーター、アクセス、レスポンスなどのコンポーネントを含むエンドポイントのREST API定義
- データのクエリまたは更新、ビジネスルールまたは制約の検証を行うプログラムロジックが記述されたスクリプト。RESTサービスによって呼び出される関数タイプは、EBX® Scriptのエクスポート関数である必要があります。

注意:

REST APIとスクリプト間の橋渡しは、**オペレーションID**です。REST APIを定義する際に、スクリプト内の関数名と**オペレーションID**プロパティに同じ値を使用する必要があります。このIDにより、エンドポイントにアクセスされたときにEBX®がどの関数を呼び出すべきかを認識できます。

次のステップ: チュートリアル用のデータモデルを作成して公開します。

チュートリアル用のデータモデル

このチュートリアルを進めるには、以下に示す基本的なデータモデルを作成できます。

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

次のステップ: スクリプト化されたRESTサービスを作成してテストします。

スクリプト化されたRESTサービスの作成

このスクリプト化されたRESTサービスの目標は、Usersテーブルからユーザーレコードのリストを返すエンドポイントとそれに関連するスクリプトを作成することです。

1. EBX® Script IDEパースペクティブが開いていない場合は、EBX®ツールバーの右上にある  アイコンをクリックし、**Script IDE**を選択します。
2. 画面左側のドックで** REST service**を見つけて選択します。
3. +アイコンをクリックして新しいスクリプトを作成し、名前を指定した後、スクリプトを保存します。
4. RESTパスとオペレーションを作成します。
 - a. **API**タブを選択し、新しいパスを作成します。
 - b. パステンプレート名に `/users` と入力し、クリックして確定します。

注:

ここには示されていませんが、パラメーターを追加するには、スラッシュで区切られた中括弧を使用します。例: `/somePath/{param1}/{param2}`。

- c. **GET**タブの下に新しいオペレーションを作成します。
- d. **オペレーションID**を入力するか、オプションで**生成**を選択してIDを自動的に作成します。このチュートリアルでは、値 `getUsers` を使用します。

注意:

オペレーションIDは、このエンドポイントとそれに対応するスクリプト関数との間のマッピングとして機能します。

- e. APIを保存します。
5. オペレーションレスポンスのデータ型を構築します。
 - a. **DATA TYPES**タブを選択します。
 - b. **Person** という名前で新しいデータ型を追加します。
 - c. スクリプト内のデータモデルフィールドにマッピングされる以下のプロパティを追加します。
 - 名前: **id**、型: decimal
 - 名前: **firstName**、型: string
 - 名前: **lastName**、型: string
 - d. データ型を保存します。
6. データ型をRESTレスポンスに追加します。
 - a. **API DEFINITION**タブを開きます。
 - b. **/users** パスを展開します。
 - c. **Responses**グループで、**型**メニューから **Person** を選択し、**配列**ボックスにチェックを入れます。
 - d. APIを保存します。
7. スクリプトを作成します。
 - a. **オペレーションID**フィールドの右側にある**** SCRIPT****ボタンをクリックして、オペレーションIDと同じ名前の関数テンプレートを自動的に生成します。

生成された関数は以下のとおりです。

```
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

```

- b. 関数のロジックを記述する前に、このスクリプトを実行したいデータスペースとデータセットを取得するためのヘルパーメソッドを作成します。以下の例では、`Reference` と `userDirectory` と異なる場合は、データスペースとデータセットの名前を指定する必要があります。

```

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

```

- c. サービスのビジネスロジックを処理するスクリプトを記述します。

以下のサンプルは、関数の実装を説明するために注釈が付けられています。

```

//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. スクリプトを保存します。IDEは自動的にコンパイルエラーをチェックし、スクリプトの下
** Messages**セクションに表示します。
9. エラーが見つからない場合は、**PUBLISH**を選択します。
10. Script IDEの組み込みテスト機能を使用してサービスをテストします。または、Postmanなどの
ツールを使用してテストすることもできます。
 - a. 右上にある** TEST API**をクリックします。
 - b. テストするAPIを選択します。この場合は `users` です。
 - c. **Test Request**をクリックし、表示される次の画面で** Send**をクリックします。

以下に示すように、レスポンスボディにはユーザーテーブルのすべてのレコードが含まれているはずです。



```

Body
Download
application/json

1  [
2    {
3      "id": 1,
4      "firstName": "Bob",
5      "lastName": "Smith"
6    },
7    {
8      "id": 2,
9      "firstName": "Sherrie",
10     "lastName": "Price"
11   }
12 ]

```

次のステップ: チュートリアルで学んだことを基に、追加のスキプト化されたCRUD操作を作成します。

CRUD操作のスキプト化されたRESTサービスの例

このセクションには、さまざまなCRUD操作のスキプト化されたRESTサービスの例が含まれています。各例には、使用されるパス、RESTメソッド、オペレーションID、およびデータ型が含まれています。サンプルは上記のチュートリアルに基づいており、いくつかの一般的なユースケースを実装する方法を示すことを目的としています。これらの例をテストする前に、[例の前提条件](#)を必ず確認してください。

例の前提条件

これらの例を既存のデータで使いたい場合は、サンプルコードを適切なデータモデル、データスペース、およびデータセット情報で更新する必要があります。サンプルコードをそのまま使用するには、環境を次のように設定します。

- [データモデル](#)
- [データ型](#)
- [共有コードテンプレート](#)

データモデル

サンプルコードは、チュートリアルで使用されているのと同じ[データモデル](#)に基づいています。これは、コードサンプルで参照されているデータモデルコンポーネントの命名に反映されています。

データ型

DATA TYPESタブの下に以下の型が作成されていることを確認してください。

データ型	プロパティ
Person	• decimal 型の <code>id</code> • string 型の <code>firstName</code> • string 型の <code>lastName</code>
Error	string 型の <code>message</code>

共有コードテンプレート

以下のコードは、以下のすべての例で必要です。これには、必要なコアユニット、データモデル参照、および2つのヘルパー関数が含まれています。後の例ではこのテンプレートを参照し、各オペレーションに固有のコードのみを示します。

```
//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

```

ユーザーの作成

以下のAPI定義とスクリプトを使用してユーザーを作成します。

API定義

パス	/users
パスパラメーター	該当なし
RESTメソッド	POST
オペレーションID:	createUser
クエリパラメーター	該当なし
ボディ	型: Person 必須: はい
レスポンス:	型: Person

スクリプト

以下のスクリプトには、2つの関数と1つのプロシージャが含まれており、次の処理を行います。

- リクエストボディで指定されたユーザーが存在するかどうかをチェック
- ユーザーが存在しない場合、データベースにユーザーを作成
- ユーザーが存在する場合、エラーメッセージを返す。データは更新されない

```
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

```

IDによるユーザーの取得

以下のAPI定義とスクリプトを使用して、IDを使用してユーザーを返します。

API定義

パス	/users/{id}
パスパラメーター	名前: id 型: integer
RESTメソッド	GET
オペレーションID:	getUser

レスポンス:

型: Person

配列: チェック済み

スクリプト

以下のスクリプトは、データベースからユーザーをクエリし、そのユーザーが見つからない場合はエラーメッセージを返します。

```
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

```

ユーザーの更新

以下のAPI定義とスクリプトを使用してユーザーを更新します。

API定義

パス	/user
パスパラメーター	該当なし
RESTメソッド	PUT
オペレーションID:	updateUser
クエリパラメーター	該当なし
ボディ	型: Person 必須: はい
レスポンス:	なし。レスポンスボディなしをチェック

スクリプト

以下のスクリプトには2つの関数が含まれており、次の処理を行います。

- リクエストボディで指定されたユーザーが存在するかどうかをチェック
- ユーザーが存在しない場合、エラーが返される。データは更新されない
- ユーザーが存在する場合、そのレコードは提供されたデータで更新される

```

export function updateUser(const user: Person): rest_response
begin
    ref response: rest_response;

    execute transaction tr on getDirectoryDataspaces() 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
```

IDによるユーザーの削除

以下のAPI定義とスクリプトを使用して、IDを使用してユーザーを削除します。

API定義

パス	/users/{id}
パスパラメーター	名前: id 型: integer
RESTメソッド	DELETE
オペレーションID:	deleteUser

レスポンス:

レスポンスボディなし: チェック済み

スクリプト

以下のスクリプトは、データベースからユーザーをクエリし、ユーザーが見つかった場合は削除します。

```
export function deleteUser(const id: int): rest_response
begin
    execute transaction tr on getDirectoryDataspaces() 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

    end

    return rest.emptyResponse(204);
```

end

ユーザーインターフェイスリファレンス

- [概要](#)
- [ドック](#)
- [ナビゲーションパネル](#)
- [ワークスペース](#)

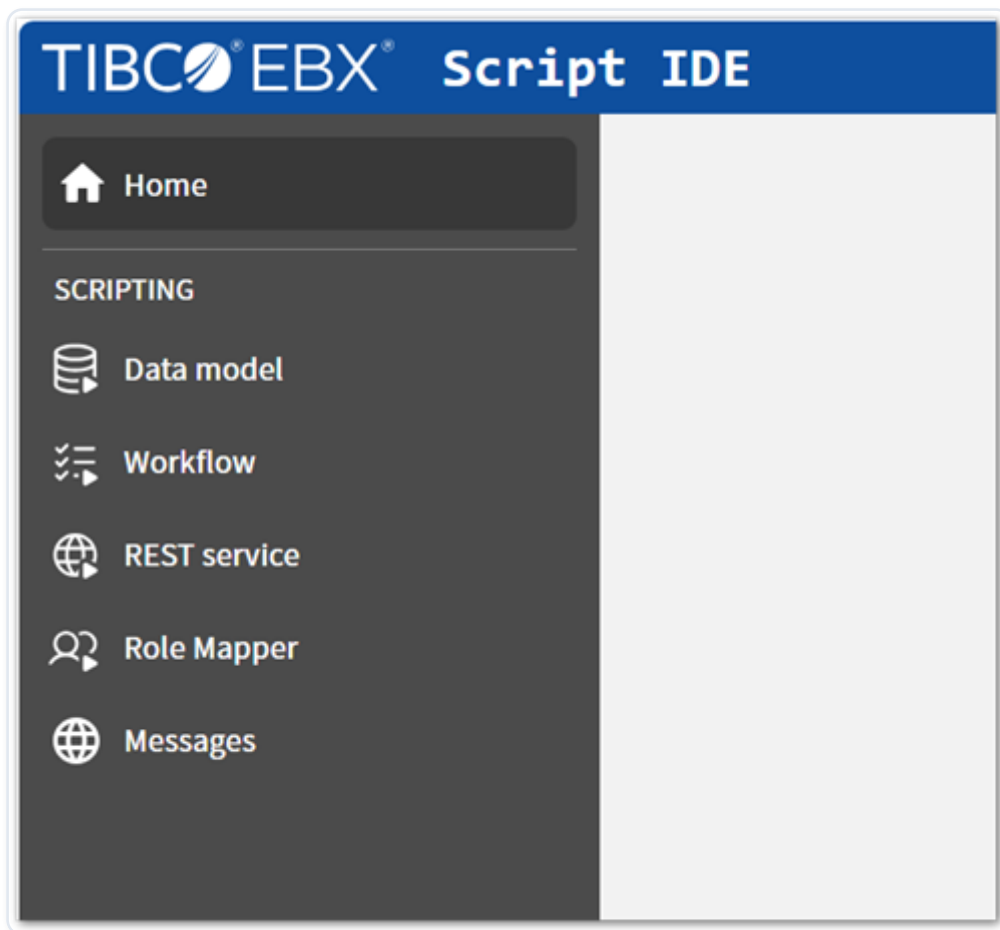
概要

EBX® Script IDE には、次の主要な UI コンポーネントが含まれています。

- **ドック:** さまざまな種類のスクリプトワークスペース (データモデル、ワークフロー、REST サービス、ロールマッパー、およびメッセージ) にアクセスできます。
- **ナビゲーションパネル:** 表示しているスクリプトワークスペースの種類に応じて、参照、選択、作成、インポート、およびエクスポートができます。
- **ワークスペース:** スクリプトの記述、REST エンドポイントの構築などを行う場所です。

ドック

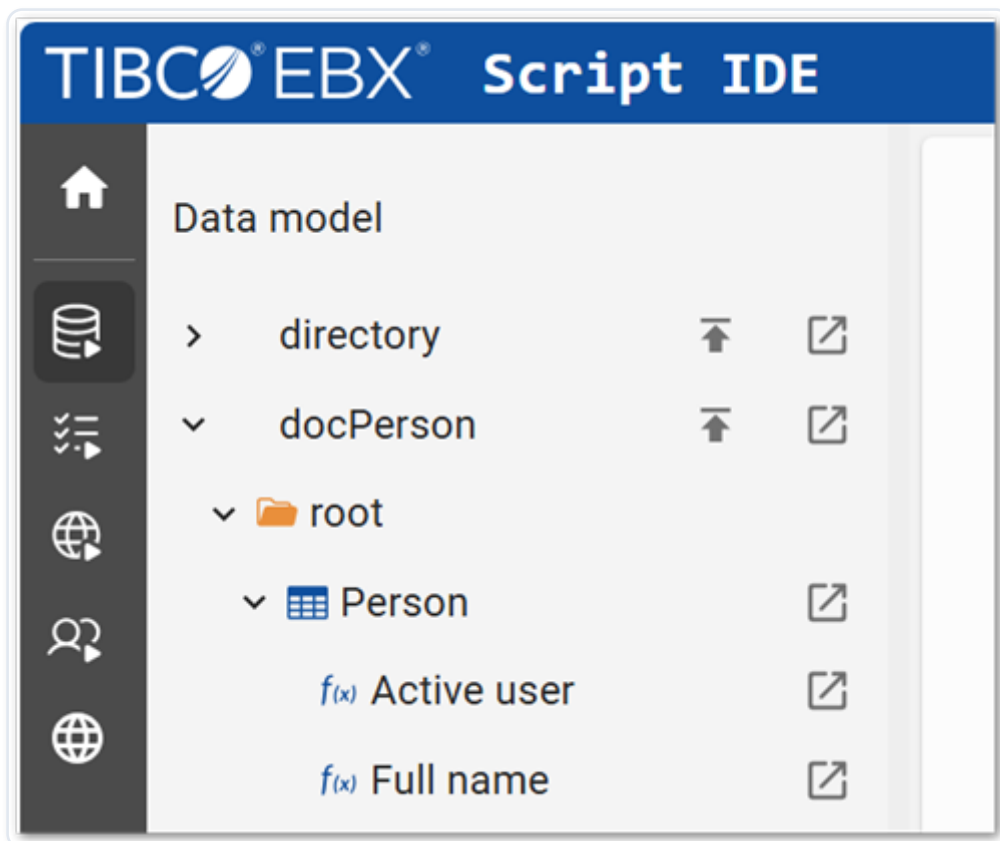
ドックは画面の最も左にあるコンポーネントです。利用可能なスクリプトワークスペースにアクセスできます。各ワークスペースは、1 種類のスクリプト作成に特化しています。ドックはサイズ変更可能で、以下が含まれます。



ナビゲーションパネル

ナビゲーションパネルは中央の UI コンポーネントで、ファイルエクスプローラーのように機能します。ここでスクリプトを参照して選択できます。各スクリプトタイプには、次のクイックアクションアイコンが指定されています。

- 作成、削除、および公開
- データモデルアシスタント (DMA) の特定のコンポーネントに移動
- インポートとエクスポート



次の表は、ナビゲーションパネルで使えるアイコンとアクションを示しています。

☐	[すべて公開] アイコンは、特定のデータモデル内のすべてのスクリプトを公開します。
☐	[DMA で表示] アイコンは、選択した項目にフォーカスを当てたデータモデルアシスタント (DMA) を開きます。
☐	[作成] ボタンは、ワークフロー、REST サービス、ロールマッパー、およびメッセージで利用できます。
☐	[削除] アイコンは、ワークフロー、REST サービス、ロールマッパー、およびメッセージで利用できます。
☐	[インポート] アイコンを使用すると、スクリプトをインポートできます。
☐	[エクスポート] アイコンを使用すると、スクリプトをエクスポートできます。

ワークスペース

ワークスペースは、スクリプトを記述および編集するメイン領域であり、コーディングが行われる場所です。スクリプト化された REST サービスの場合、REST API 定義を作成できる場所でもあります。

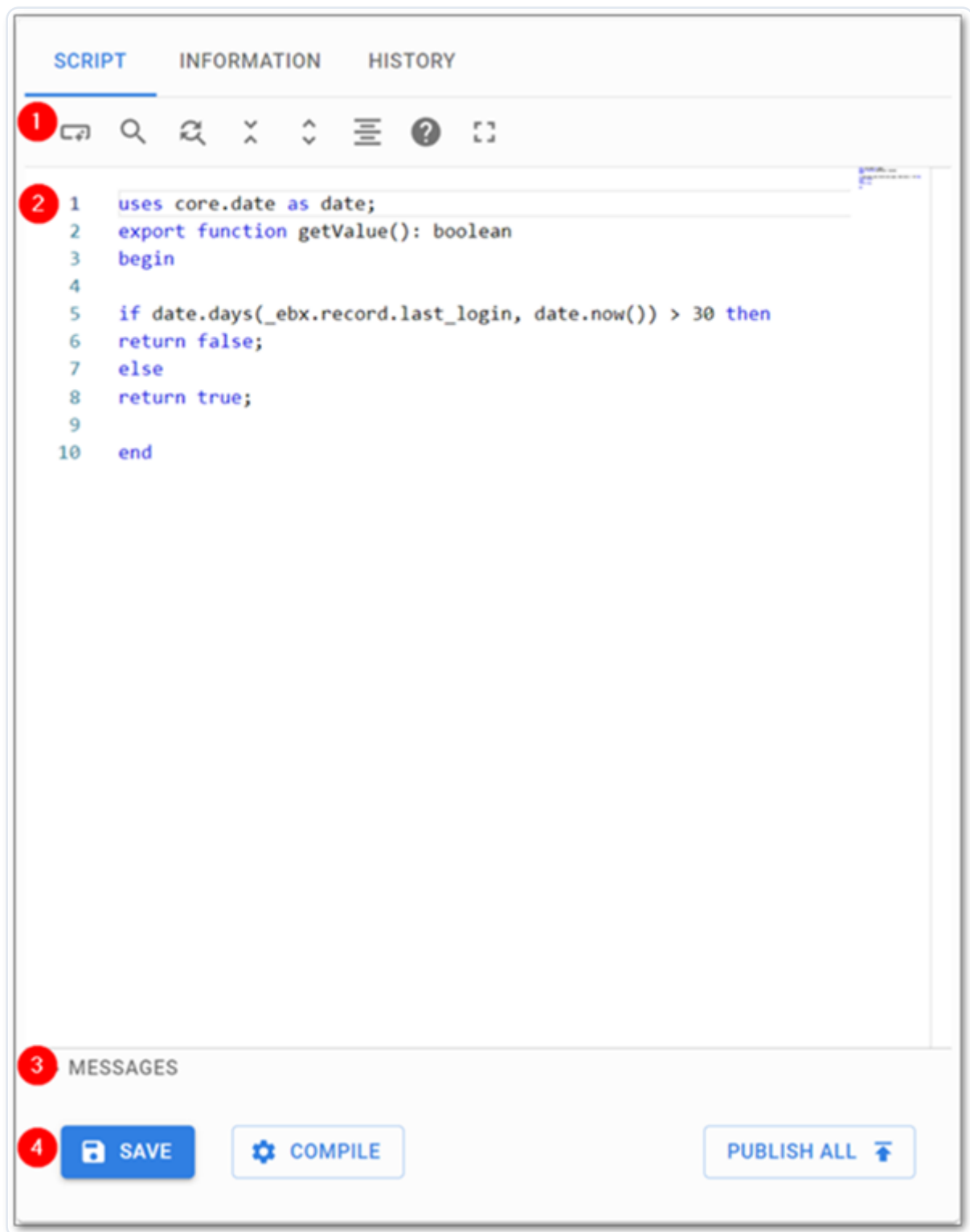
ワークスペース には、次のタブとセクションが含まれています。

- **タブ**：このタブではスクリプトの作成、編集、表示ができます。次の機能が含まれます。
 - コード補完
 - 個々のコンポーネントのツールヒント/ヘルプを表示するにはマウスを移動します。
 - 構文の強調表示
 - コンパイルとエラー検出
 - スクリプトの保存と公開

タブ の上部には、さまざまなアクションを呼び出すアイコンを含むツールバーがあります。各アイコンの上にマウスポインタを置くと、それぞれの簡単な説明が表示されます。

タブ の本体には、実際のスクリプトが記述されます。下部には、スクリプトを保存またはコンパイルした後の情報を表示する展開可能な セクションがあります。メッセージの下には、次のボタンがあります。

- ：スクリプトを自動的にコンパイルして保存します。
- ：スクリプトを保存せずにコンパイルします。
- ：関連するデータモデルに含まれるすべてのスクリプトを公開します。



- **タブ**：このタブには、選択したスクリプトの詳細が表示されます。また、検索最適化のために、有効なスクリプトをインデックス化するかどうかもここで指定します。
- **タブ (ワークフロースクリプトのみ)**：スクリプトに適用されるパラメーターを追加および管理できます。
- **タブ (REST サービススクリプトのみ)**：パス、パスパラメーター、メソッド、クエリパラメーター、本文、および応答を含む REST エンドポイントを定義できます。

- **タブ** (REST サービススクリプトのみ) : API 定義で使用するデータ型を作成できます。データ型は、REST サービスのスクリプトを作成するときにも使用できます。
- **タブ** (REST サービススクリプトのみ) : API 定義のリビジョンリストを表示し、現在のリビジョンとの違いを比較できます。
- **タブ** : このタブでスクリプトのリビジョンを効率的に管理します。スクリプトに保存されたすべての変更を時系列でわかりやすく表示します。履歴の各エントリは特定の変更のスナップショットであるため、時間の経過とともにスクリプトがどのように進化したかをすばやく理解できます。

ロールバック機能 : 矢印を使用して個々の変更を元に戻し、スクリプトを以前の状態にロールバックします。

Java API Reference

- [TIBCO EBX® 6.2.3 Java API](#)

移行ガイド

- 移行の概要
- 環境とハードウェア要件
- サポートされているアドオンの変更
- ステップ 1: カスタムコードの調整
- ステップ 2: リポジトリの準備
- ステップ 3: 展開
- Java API の変更

移行の概要

- 概要
 - 追加のドキュメントリソース
- 移行ロードマップ

概要

EBX® 5.9 から 6 へのリポジトリの移行は、サーバーの起動時に自動的に行われます。ただし、自動移行が確実に成功するためには、起動前にいくつかの技術的および機能的なタスクを実行する必要があります。これらのタスクには、次のようなものがあります。

- 既存の 5.9 環境をバックアップしてクリーンアップします。このガイドは 5.9 から 6 への移行時に使用することを目的としていますが、EBX® 6 の古いバージョンからアップグレードする場合でも、既存のリポジトリをバックアップすることが不可欠です。
- カスタムモジュールのアドレス指定
- 新しいスケーラブルなアーキテクチャとそれがカスタム Java コードのパフォーマンスに与える影響を理解します。
- 新しいハードウェアおよびメモリ要件に合わせて調整します。

注意:

このドキュメントで使用されているバージョン番号の規則は次のとおりです。

- 5.9 は、5.9.0 から 5.9 の最新のサービスパックまでのリリースを指します。
- 6 は、6.0.0 から最新リリースまでのリリースを指します。
- 6.0 は、6.0.0 から 6.0 の最新のサービスパックまでのリリースを指します。
- 6.x は、6.1.0 から最新リリースまでのすべてのリリースを指します。

おそらく、上記のタスクは、組織内のさまざまなロール、さまざまな環境 (開発、プレプロダクション、実稼働など)、およびリポジトリの複数のバージョンを使用して実行されます。一部のタスクは、環境ごとに 1 回ずつ、複数回実行する必要があることに注意してください。開始するには、以下を参照してください。

- **移行ロードマップ** では、移行プロセスのさまざまなフェーズの概要を示し、関連するドキュメントへのリンクを提供します。
- **環境とハードウェアの要件** では、移行が既存のハードウェアに与える影響について説明します。
- **サポートされるアドオンの変更** では、EBX® 6 に移行されたアドオン、再設計されたバージョンのアドオン、および廃止されたアドオンの詳細を示します。

追加のドキュメントリソース

移行プロセスに関連する追加情報については、次のドキュメントを参照してください。

- docs.tibco.com で入手可能な **すべてのリリースノート** セクションで、移行先のバージョンまでのリリースの **下位互換性** セクションを注意深く確認してください。これらのセクションには、移行作業に役立つ重要な情報が含まれています。
- *『TIBCO EBX® 管理ガイド』*の **インストールと構成** セクションを参照してください。このセクションには、パフォーマンスのヒント、ハードウェアのガイドライン、展開情報、および環境固有のインストール手順が記載されています。

移行ロードマップ

移行プロセスは、組織のビジネス要件と EBX® の実装によって異なります。以下は、一般的な要件に対応し、特定のニーズに対応するために構築および適応できる基盤を提供する一般的なガイドラインです。リンクをクリックすると、*『移行ガイド』*の関連セクションに移動できます。

ステップ 1) ローカル環境 (開発者)	このフェーズには次の内容が含まれます。 • EBX® 6 JAR のダウンロード • 新しい EBX® 6アーキテクチャの理解 • パフォーマンスを向上させ、互換性を確保するためにカスタムコードをリファクタリング • 新しい JAR でカスタムコードをコンパイル • テスト移行の実行 • バイナリの生成 詳細な手順については、ステップ 1: カスタムコードの適応 を参照してください。
ステップ 2) プレプロダクション環境 (Dev Sec Ops)	このフェーズには次の内容が含まれます。 • リポジトリの プレプロダクションバージョンを準備します。詳細な手順については、ステップ 2: リポジトリの準備 を参照してください。 • 新しい環境とハードウェアの要件を理解します。詳細については、環境とハードウェアの要件 を参照してください。 • 実稼働前環境での展開とテスト。詳細な手順については、ステップ 3: 展開 を参照してください。
ステップ 3) 実稼働環境 (Dev Sec Ops)	このフェーズには次の内容が含まれます。 • リポジトリの 実稼働バージョンを準備しています。詳細な手順については、ステップ 2: リポジトリの準備 を参照してください。

- 実稼働環境が新しい環境およびハードウェア要件に準拠していることを確認します。詳細については、[環境およびハードウェア要件](#) を参照してください。
- 運用環境での展開とテスト。詳細な手順については、[ステップ 3: 展開](#) を参照してください。

環境とハードウェア要件

- [サポートされる環境](#)
- [ハードウェアリソース](#)

サポートされる環境

EBX® 6 に移行できるのは EBX® 5.9 リポジトリのみです。以前のバージョンを使用している場合は、まずリポジトリをバージョン 5.9 に移行する必要があります。また、6.0 ベータリリースリポジトリから 6 GA バージョンへの移行はサポートされていません。常に最新の EBX® リリースへの移行をお勧めします。

必要なアップグレードの詳細については、以下に示す [サポートされている環境](#) を参照してください。

ブラウザーのサポート	Microsoft Internet Explorer と Microsoft Edge の非 Chromium バージョンはサポートされていないことに注意してください。
JVM サポート	必要な Java ランタイム環境は、JRE 17、21、または 25 LTS のいずれかです。Java 8 および 11 はサポートされなくなりました。
IBM サポート	WebSphere Application Server 9 はサポートされなくなりました。 WebSphere Application Server Liberty (Java EE 8 のみ) は、バージョン 6.0.0 からバージョン 6.2.0 まで (バージョン 6.2.0 は除く) がサポートされます。その後、バージョン

	6.2.0 以降では、Jakarta EE 10+ への移行により、Jakarta EE 10 では Websphere Application Server Liberty 23 以降のみがサポートされます。
Oracle サポート	Oracle 12c および 21c はサポートされなくなりました。現在サポートされている Oracle の最小バージョンは 19c、23c 以降です。 WebLogic Server 15c がサポートされます。
PostgreSQL サポート	PostgreSQL 10 から 13 はサポートされなくなりました。現在サポートされている PostgreSQL の最小バージョンは 14 で、最新のサポートバージョンは 18 です。
Microsoft サポート	SQL Server 2014 はサポートされなくなりました。現在サポートされている SQL Server の最小バージョンは 2016 です。

ハードウェアリソース

EBX® 6 ではデータインデックスを管理するために Apache Lucene が導入されているため、ハードウェアのアップグレードが必要になる場合があります。インデックスはメモリから永続ストレージにオフロードされるようになりました。次の表にある高レベルの推奨事項に加えて、[パフォーマンスとチューニング](#) に含まれる情報を注意深く確認する必要があります。

	ディスク	メモリ
推奨	- Lucene インデックスを永続化するために十分なディスク領域が確保されていることを確認します。 - SSD などの高性能ドライブを使用します。	- Java 仮想マシン (JVM) へのメモリ割り当てを減らします。 - OS へのメモリ割り当てが十分であることを確認します。

	ディスク	メモリ
	ストレージエリアネットワーク (SAN) を使用する場合は、ファイバーチャネル (FC) などの他の高性能テクノロジーと組み合わせます。 ディスク要件の詳細については、ディスクに関するパフォーマンスのセクションを参照してください。	ガベージコレクターを調整してパフォーマンスを向上させます。 メモリ要件の詳細については、メモリに関するパフォーマンスのセクションを参照してください。
非推奨	- EBX® ルートディレクトリを他の EBX® インスタンスと共有すること。これには、ホットスタンバイによるフェールオーバーが構成されている場合のアクティブインスタンスとスタンバイインスタンス間の共有が含まれます。 - Lucene インデックスを含むルートディレクトリをネットワーク接続ボリューム (NAT) に保存すること。 - ルートディレクトリでウイルス対策ソフトウェアを使用すること。 ディスク要件の詳細については、ディスクに関するパフォーマンスのセクションを参照してください。	

サポートされているアドオンの変更

- 概要
- アドオンの手動移行
- Data Model and Visualization Add-on の追加手順
- Digital Asset Manager Add-on の追加手順

概要

一部のアドオンは EBX® 6 に移行され、その他のアドオンは改良バージョンに置き換えられ、いくつかのアドオンは段階的に廃止されました。これらの更新と廃止の包括的な詳細は、以下の表で確認できます。削除または置き換えられたアドオンにデータモデルが依存している場合は、自動移行プロセスを開始する前に対処する必要があります。

アドオン	EBX® ステータスと移行要件
TIBCO EBX® Match and Merge Add-on (以前の Match and Cleanse (DAQA))	このアドオンは廃止され、EBX® 6 で改良されたバージョンである TIBCO EBX® Match and Merge Add-on (MAME) に置き換えられました。 DAQA の構成は移行されません。TIBCO EBX® 5.9 でこのアドオンを使用した場合は、移行を妨げる可能性のあるアーティファクトが残らないように、 廃止されたアドオンのインクルード の指示に従う必要があります。
TIBCO EBX® Match and Merge Add-on (MAME)	このアドオンは、TIBCO EBX® Match and Merge Add-on (旧称 Match and Cleanse DAQA) の改良版であり、これに代わるものです。このアドオンは新しいため、移行プロセス中に特別な考慮は必要ありません。
TIBCO EBX® Data Exchange Add-on (ADIX)	EBX® 6 には、このアドオンの改良版である TIBCO EBX® Data Exchange Add-on (New) (DINT) があります。従来のアドオンの構成情報は自動的に移行されます。それでも、構成のバックアップをエクスポートすることをお勧めします。
TIBCO EBX® Data Exchange Add-on (New) (DINT)	このアドオンは、TIBCO EBX® Data Exchange Add-on (ADIX) の改良版です。このアドオンは新しいため、移行プロセス中に特

アドオン	EBX® ステータスと移行要件
	別な考慮は必要ありません。このアドオンの旧バージョンは、EBX® 6 でも引き続き使用できます。アドオンのバージョンを区別しやすくするために、UI のメニューでは改良版に (New) が追加されています。
TIBCO EBX® Insight Add-on (DQID)	このアドオンは EBX® 6 で使用可能で、その構成情報は自動的に移行されます。このアドオンの改良版である TIBCO EBX® Insight Add-on (New) (DPRA) が使用可能です。アドオンのレガシーバージョンを展開しない場合は、関連する Java 拡張機能を削除する必要があります。詳細については、廃止されたアドオンの Java 拡張機能 を参照してください。
TIBCO EBX® Insight Add-on (New) (DPRA)	このアドオンは、TIBCO EBX® Insight Add-on (DQID) の改良版です。このアドオンは新しいため、移行プロセス中に特別な考慮は必要ありません。このアドオンの旧バージョンは、EBX® 6 でも引き続き使用できます。アドオンのバージョンを区別しやすくするために、UI のメニューでは改良版に (New) が追加されています。
TIBCO EBX® Information Search Add-on (TESE)	このアドオンの機能の一部は現在コア製品に含まれており、その構成は大幅に異なります。アドオンの構成は自動的に移行されません。
TIBCO EBX® Digital Asset Manager (DAMA)	アドオンの構成は自動的に移行されます。以前のバージョンで非推奨となった API は使用できなくなります。そのため、非推奨となった API を新しいバージョンに置き換えるには、コードレビューが必要になる場合があります。
TIBCO EBX® Data Model and Data Visualization (DMDV)	アドオンの構成は自動的に移行されません。構成のアーカイブをエクスポートし、移行プロセスが正常に完了した後に再インポートする必要があります。このタスクを実行する手順については、アドオンの手動移行 を参照してください。リンクの方向を定義する新しい機能があります。これは必須フィールドであり、移行後にすべてのリンクに対して定義する必要

アドオン	EBX® ステータスと移行要件
	があります。一度設定して、他のレコードに変更を適用することができます。詳細については、 こちらの手順 を参照してください。
TIBCO EBX® EBX GO Add-on (MODA)	このアドオンの構成は自動的に移行されません。構成ファイルのアーカイブをエクスポートし、移行が成功した後にインポートする必要があります。このタスクを実行する手順については、 アドオンの手動移行 を参照してください。
TIBCO EBX® Rules Portfolio Add-on (RPFL)	このアドオンは廃止されました。このアドオンを以前に使用していた場合は、関連するJava 拡張機能を削除する必要があります。詳細については、 廃止されたアドオンの Java 拡張機能 を参照してください。
TIBCO EBX® Graph View Add-on (GRAM)	このアドオンは廃止されました。
TIBCO EBX® Information Governance Add-on (IGOV)	このアドオンは廃止されました。このアドオンを以前に使用していた場合は、関連するJava 拡張機能を削除する必要があります。詳細については、 廃止されたアドオンの Java 拡張機能 を参照してください。
TIBCO EBX® Oracle Hyperion EPM Add-on (HMFH)	このアドオンは廃止されました。
TIBCO EBX® Activity Monitoring Add-on (MTRN)	このアドオンは廃止されました。

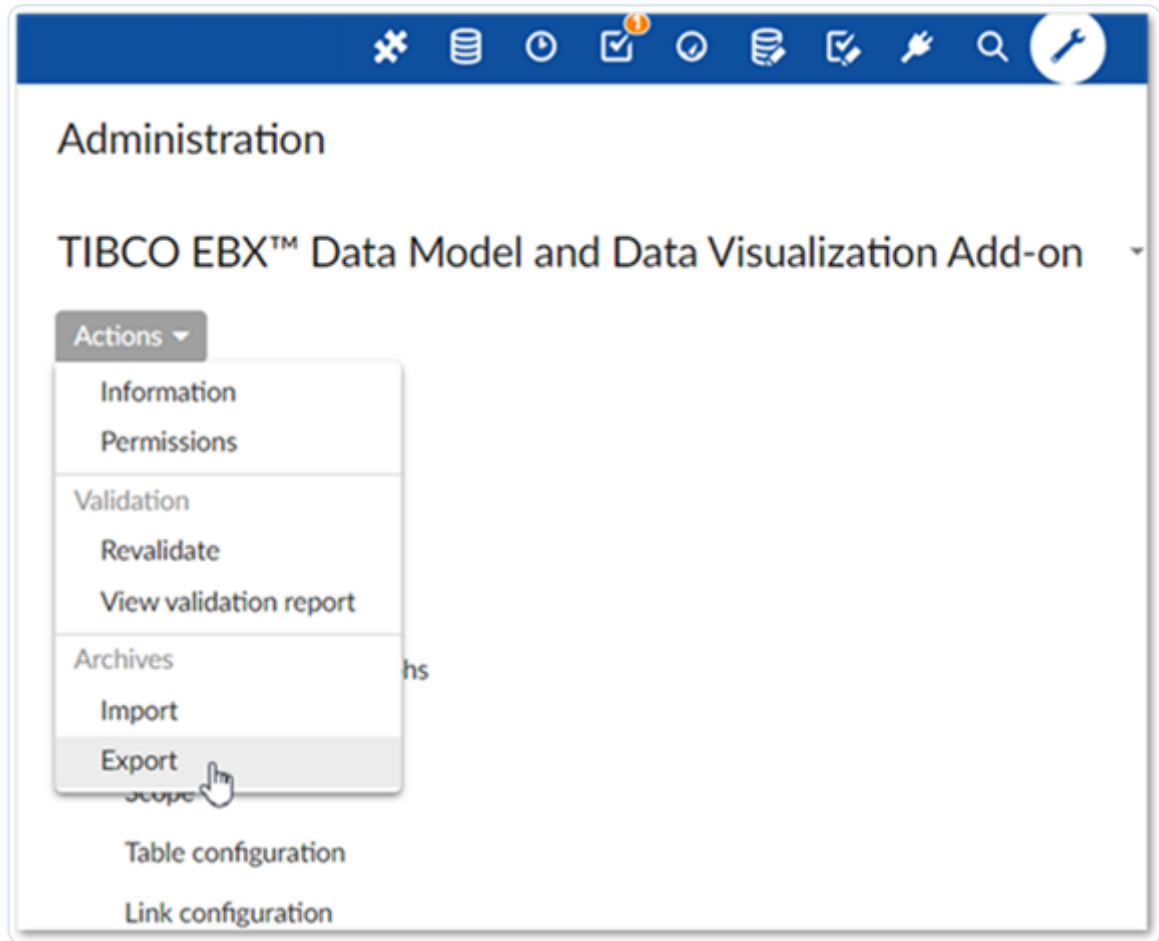
アドオンの手動移行

アドオン構成を手動で移行するには、次の操作を実行します。

1. アドオンの設定のアーカイブをエクスポートします。
 - a. 目的のアドオンに移動します。
 - [管理 > ユーザーインターフェイス > TIBCO EBX® Data Model and Data Visualization Add-on](#)

- 管理 > TIBCO EBX® GO Add-on > 構成

b. 次の画像に示すように、メニューから > を選択します。



c. アーカイブの名前を入力し、デフォルト設定のままにして、を選択します。

エクスポートが成功してもメッセージは表示されません。

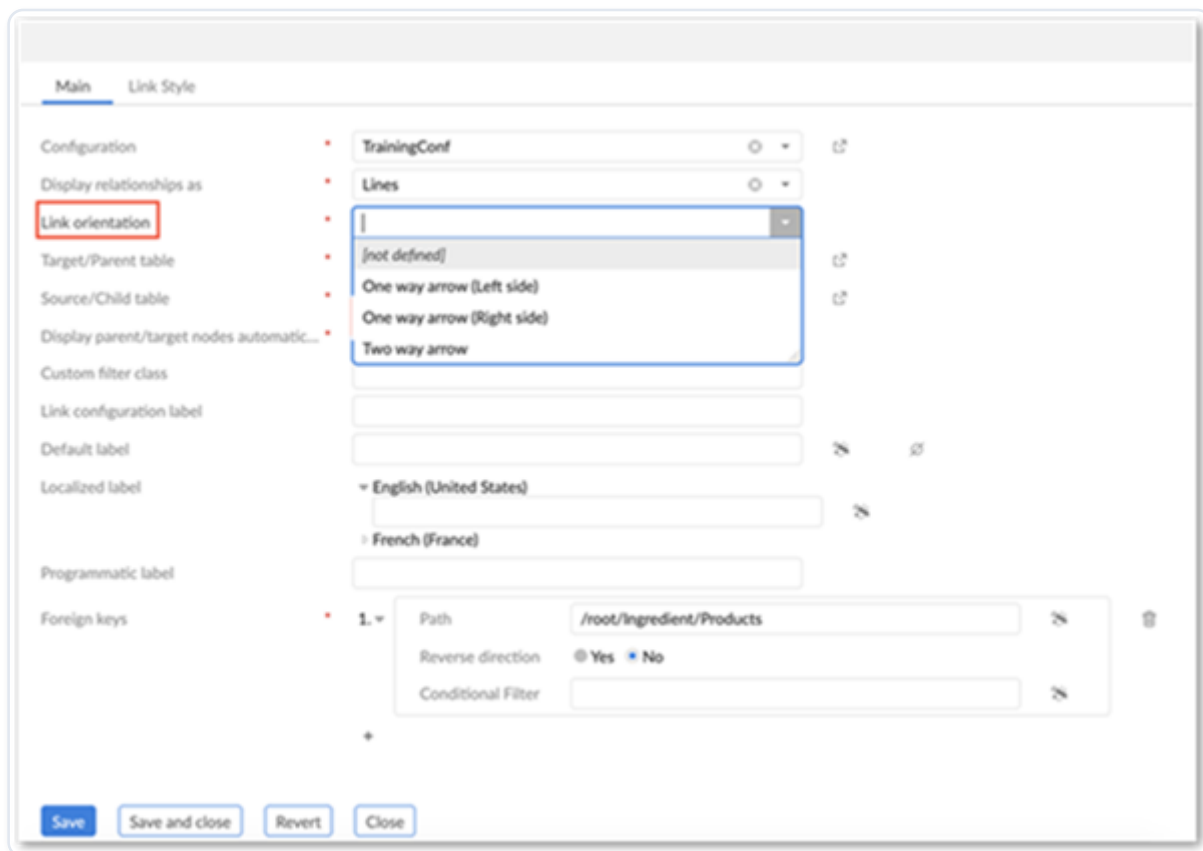
2. 構成アーカイブをインポートします。

- 目的のアドオンに移動します。
- メニューから、> を選択します。
- メニューから、手順 1 でエクスポートしたアーカイブの名前を選択します。
- を選択します。

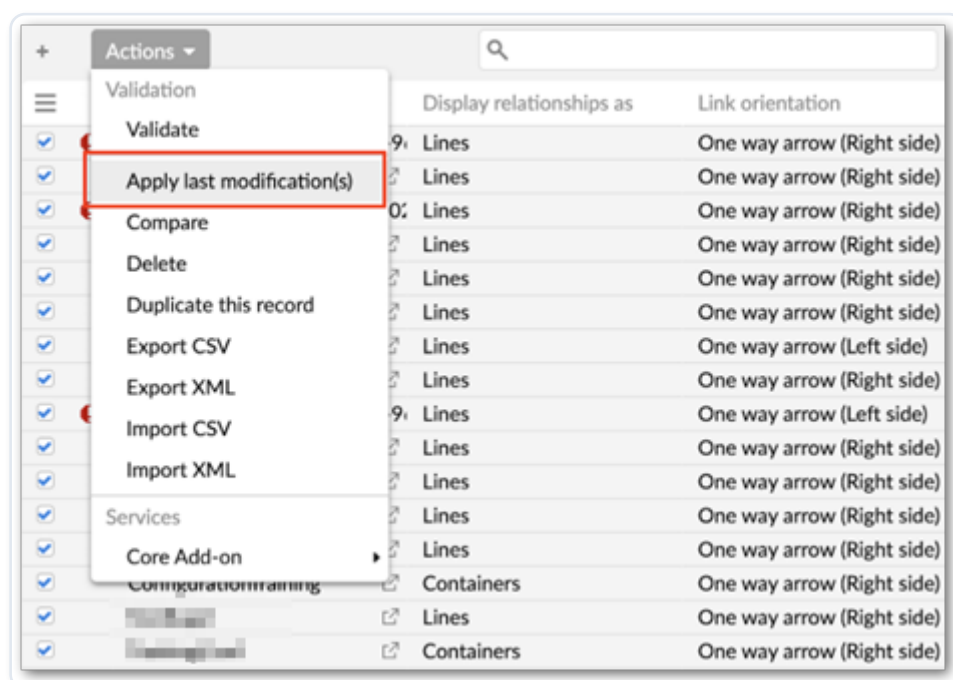
Data Model and Visualization Add-on の追加手順

アドオンには、ユーザーがリンクの方向を定義できる新しい機能が含まれています。これはすべてのリンクの必須フィールドであり、初期化する必要があります。リポジトリを移行し、アドオンの構成アーカイブをインポートした後、次の手順を実行します。

1. >>> に移動します。
2. レコードを開き、プロパティを設定し、レコードを保存して閉じます。



3. オプションで、テーブル内のすべてのレコードに同じ変更を適用します。
 - a. テーブル内のすべてのレコードを選択します。
 - b. メニューを開き、 を選択します。



Digital Asset Manager Add-on の追加手順

EBX® 5.9.X と互換性のあるバージョンから 6 と互換性のあるバージョンにアドオンを移行するには、次の手順に従います。

1. 古いバージョンのアドオンが含まれている環境で、>> {addon.label.full} に移動します。
2. メニューから、> を選択します。
3. 新しいバージョンのアドオンを含む環境で、>> {addon.label.full} に移動します。
4. メニューから、> を選択します。

バージョン 5 から 6 へのすべてのプロパティの直接マッピングは行われなかったため、一部のテーブルにエラーが発生することに注意してください。これらのエラーを解決するには、次の手順を実行します。

5. メニューから、> を選択します。

以下に、バージョン 5 と 6 の間で行われた用語の変更の一部を示します (その他の変更はごくわずかです)。

- ドライブタイプ --> コネクタ
- ファイルシステム --> デフォルトコネクタ
- 外部管理 --> 外部マネージャー
- 表示構成 --> レコード詳細表示
- エディター構成 --> ドライブビュー構成

ステップ 1: カスタムコードの適応

- 概要
- 新しいアーキテクチャがカスタムコードのパフォーマンスに与える影響
- コードの適応と移行のテスト

概要

開発者として、移行をサポートする役割は、EBX® 実装で使用されるカスタムコードが新しい EBX® バイナリに対してコンパイルされ、正常に展開できることを保証することです。このタスクでは、新しい EBX® 6 アーキテクチャの [違いを理解している](#) ことと、それが既存のカスタム Java コードのパフォーマンスにどのように影響するかを理解していなければなりません。また、

TIBCO EBX® リリースノートに記載されている下位互換性に関する情報にも精通している必要があります。

新しいアーキテクチャがカスタムコードに与える影響を確認して、カスタムコードの更新プロセスを開始します。次に、コードの適応と移行のテストの手順に従います。テストフェーズが正常に完了すると、次のようになります。

- コードの調整とテストが完了。
- ローカル環境での移行を正常にテストするための必要なデータモデルとプロジェクトの変更が完了。
- 実稼働前の環境での展開とテストが可能な新しいバイナリの生成が完了。

新しいアーキテクチャがカスタムコードのパフォーマンスに与える影響

新しいスケーラブルなアーキテクチャは、カスタム Java コードのパフォーマンスに影響を与える可能性があります。以前は問題なく動作していたシナリオでも、調整が必要になる場合があります。特に、次の項目の置き換えや最適化に重点を置いてください。

主キーによる繰り返しルックアップ	以前は、カスタム EBX® インメモリインデックスのキャッシュにテーブル全体のデータが含まれる場合、ランダムテーブル検索を使用してレコードにアクセスする方がコストがかかりませんでした。このタイプのアクセスは、新しい SQL 機能を利用するために書き直す必要があります。たとえば、テーブル A の <code>ResultSet</code> が反復され、テーブル B で <code>osd:tableRef</code> が検索されるネストされたループ結合を実行する Java コードを考えてみましょう。SQL 結合を使用して検索を処理する <code>Query</code> を実装する必要があります。
類似リクエストの繰り返し作成	EBX® には、Apache Calcite フレームワークが組み込まれ、送信された <code>Query</code> および <code>Request</code> インスタンスが最適化されるようになりました。クエリの最適化にはコストがかかりますが、最適化されていないク

	エリを実行する場合と比較すると、通常はコストが低く、回収できます。ただし、ループによって1つのパラメーターのみが異なる多数のクエリが生成される場合は、大幅なオーバーヘッドが発生する可能性があります。この場合、パラメーター化されたリクエスト (<code>Request.setXPathParameter</code> を参照) またはパラメーター化されたクエリ (<code>Query.setParameter</code> を参照) を使用するのが最適です。
プログラムによるアクセスルール	プログラムによるアクセスルールは、Java インターフェイス <code>AccessRule</code> を実装します。クエリの実行時にテーブルにルールを設定すると、各レコードに対してメソッド <code>getPermission</code> が実行されます。これにより、インデックスに対する最適化の利点が失われます。可能な場合は、これらのプログラムによるアクセスルールを、新しいスクリプト化されたレコード権限ルールに置き換える必要があります。
プログラムラベル	プログラムによるラベルは、Java インターフェイス <code>TableRefDisplay</code>、<code>UILabelRenderer</code>、および <code>ConstraintEnumeration</code> を実装します。クエリの実行時に、外部キーフィールドにルールを設定すると、各レコードに対してメソッド <code>displayOccurrence</code> が実行されます。これにより、インデックスに対する最適化の利点が無効になります。可能な場合は、代わりにパターン文字列を使用してください。
プログラムフィルター	プログラムフィルターは、Java インターフェイス <code>AdaptationFilter</code> を実装します。クエリの実行時にこのフィルターを使用すると、各レコードに対してメソッド

	<code>accept</code> が実行されます。これにより、インデックスに対する最適化の利点が失われます。可能な場合は、プログラムフィルターを新しい SQL クエリに置き換えてください。
依存関係が不明な制約	大規模なデータセットに対する依存関係が不明な制約は、検証リクエストごとにチェックされるため、避けてください。詳細については、データ検証 を参照してください。
不要なインデックスの更新	トランザクションパフォーマンス (<code>Procedure</code> および <code>TableTrigger</code> の Java 実装) を向上させるには、不要なインデックスの更新を避ける必要があります。詳細については、不要なインデックスの更新 を参照してください。
継承されたフィールド	検索の制限 で説明したように、継承されたフィールドは新しいインデックスの最適化の恩恵を受けることができません。その結果、継承されたフィールドで実行されるすべての操作 (クエリ、検証、データの比較など) は、EBX® 6 より前のバージョンと比較して遅くなります。パフォーマンスの問題を回避するために、継承されたフィールドを使用しないことを強くお勧めします。可能な場合は、ユースケースに応じて、これらのフィールドを リンクされたフィールド に変換するか、たとえばトリガーを使用して継承をシミュレートして継承されたフィールドの動作を置き換えることをお勧めします。
圧縮の最適化	リリース 6.2.3 以降、圧縮の実装では直接/ネイティブメモリを使用します。これにより、メモリ (およびインデックスとデータベースリソース) が迅速に返されるように、<code>RequestResult</code> および <code>QueryResult</code> オ

プロジェクトをできるだけ早く明示的に閉じることの重要性が増します。以前は、

`ReverseRelationshipResult.getResult()`

は同じ `RequestResult` インスタンスを返していました。現在は常に新しい

`RequestResult` を返します。使用後は必ず `RequestResult` を閉じるようにカスタムコードを更新してください。

コードの適応と移行のテスト

移行をテストする前に、EBX® 5 に関連するすべての JAR ファイルと WAR ファイルを環境から削除することが重要です。

1. リポジトリの準備

以下に、開発環境用にローカルリポジトリを準備するための高レベルのタスクを示します。移行用に実稼働前または実稼働リポジトリを準備する場合、または次の手順で説明するリポジトリのクリーンアップタスクの詳細については、[ステップ 2: リポジトリの準備](#) を参照してください。

- a. 既存のリポジトリをバックアップします。
- b. EBX® を起動し、すべてのデータモデルがエラーなしでコンパイルされることを確認します。
- c. 廃止されたアドオンに対するデータモデルの依存関係が存在しないことを確認してください。詳細については、[サポートされるアドオンの変更](#) を参照してください。
- d. データスペースとワークフローをクリーンアップします。この手順はオプションです。ただし、移行中に閉じられたデータスペースは削除されることに注意してください。保持するデータスペースを再度開きます。この手順は、データスペースの数が多いと移行時間が長くなる可能性がある大規模なリポジトリでより効果的です。
- e. URL ポリシーが 6.x の要件に準拠していることを確認します。 >>> ブ>> に移動して、既存の設定を元に戻します。URL コンピューティング構成を更新するには、`ebx.properties` ファイルを編集します。詳細については、[URL コンピューティング](#) を参照してください。
- f. シャットダウンしてから、クリーンなリポジトリをバックアップします。

2. コードの準備

- a. TIBCO eDelivery (eDelivery にアクセスするにはアカウントが必須です。アカウントをリクエストするには、[サポートチーム](#)にお問い合わせください) から EBX® 6 バイナリをダウンロードし、次の JAR をローカル環境 (`<your download directory>/ebx.software/lib` にあります) に展開します。

- EBX® 6.0.x に移行する場合

- EBX® API の場合: `ebx.jar`
- 新しい圧縮アルゴリズムのサポート: `ebx-lz4.jar`

- EBX® 6.x に移行する場合

- バージョン 6.2.3 より前の場合のみ、新しい圧縮アルゴリズムをサポートするため: `ebx-lz4.jar`
- EBX® Java API の場合: `ebx-api.jar`
- サードパーティ Java API: `ebx-resources-api.jar`

- b. カスタムコードが新しい推奨事項と要件に準拠していることを確認するには、次の点を確認してください。

- [新しいアーキテクチャがカスタムコードのパフォーマンスに与える影響](#)
- [リリースノート](#) > [下位互換性](#) セクション。
- [Java API の変更](#)

- c. カスタムライブラリまたは Web アプリケーションを再コンパイルします。

3. ローカルサーバーを起動して自動移行を開始します。プロセスは正常に終了するはずですが、`kernel.log` ファイルでログエントリ「`EBX® Started and Initialized`」を探します。

問題が発生する可能性のある一般的な問題は次のとおりです。

- EBX® 6 を使用してコンパイルされていないライブラリ。この場合、EBX® は例外「`java.lang.IncompatibleClassChangeError`」をスローします。
- データモデルエラー。解決されていないエラーが残っている場合、移行プロセスは自動的に終了します。
- プロジェクトが正しくビルドされていない場合、プロジェクトでコンパイルエラーが発生します。

ローカル環境での移行が正常に完了したら、より大きなリポジトリを使用して実稼働前環境に展開してテストできます。ただし、実稼働前環境でテストする場合は、アプリケーションサーバー

を起動する前に、[ステップ 2: リポジトリの準備](#) で説明されているプロセスに従ってリポジトリを準備する必要があります。

ステップ 2: リポジトリの準備

- はじめに
- タスク 1: バックアップの作成
- タスク 2: リポジトリの準備
 - データスペースのクリーンアップ
 - ワークフローのクリーンアップ
 - データモデルのクリーンアップ
 - URL ポリシー
- タスク 3: クリーンなリポジトリのバックアップ

はじめに

実稼働環境に移行する前に、少なくともテスト環境と稼働前環境ですべての移行タスクを実行することを強くお勧めします。EBX® 6 アーティファクトを含むサーバーを起動すると、リポジトリの移行が自動的に開始されます。このプロセスは元に戻せません。リポジトリ内のデータモデルにエラーが含まれていない場合にのみ、移行が正常に完了します。

注意:

実稼働環境のリポジトリのコピー、または実稼働中のリポジトリによく似たものを使用することをお勧めします。これにより、実稼働環境のリポジトリを準備するために必要な労力を見積もることができ、自動移行プロセスにかかる時間を把握できるため、それに応じてダウンタイムを計画できます。

リポジトリを移行用に準備するには、次のタスクの手順を完了します。

- タスク 1: バックアップの作成
- タスク 2: リポジトリの準備
- タスク 3: クリーンなリポジトリのバックアップ

タスク 1：バックアップの作成

移行プロセスは元に戻すことはできません。この点を考慮して、リポジトリを現在の状態でバックアップすることが重要です。

注意:

リポジトリを 5.9 から移行する場合でも、以前のバージョン 6 からアップグレードする場合でも、リポジトリをバックアップすることが重要です。

リポジトリをバックアップするには、次の手順を実行します。

1. EBX® 5 インスタンスをホストしているアプリケーションサーバーをシャットダウンします。
2. EBX® 5 リポジトリをバックアップします。名前が現在のリポジトリプレフィックスで始まるリレーショナルデータベースオブジェクトをすべて含めます。このプレフィックスは、`ebx.properties` ファイルの `ebx.persistene.table.prefix` の下に指定されています。
3. EBX® ログをバックアップして削除するか、別のフォルダーに移動します。
4. アドオン構成をバックアップします。

TIBCO EBX® データモデルおよびデータ視覚化アドオン、または TIBCO EBX® GO アドオンをお持ちの場合は、構成アーカイブをエクスポートし、移行の完了後に再度インポートする必要があります。手順については、[アドオンの手動移行](#) を参照してください。

タスク 2: リポジトリの準備

移行プロセスを高速化するには、特に大量のデータを含むリポジトリの場合、EBX® 5.9 環境をクリーンアップすることが重要です。このプロセスは EBX® ユーザーインターフェイスで実行する必要があるため、EBX® への管理アクセスが必要です。

バージョン 6.2.3 以降のデータ圧縮の変更により、不要なデータスペースをすべて削除することが特に重要です。移行では、辞書が作成され、すべてのブロックが新しい形式に書き換えられ、コンテンツを共有できるブロックが検索され、辞書を使用してブロックが再圧縮されます。

次のリポジトリエレメントをクリーンアップする必要があります。

- [データスペース](#)
- [ワークフロー](#)
- [データモデル](#)

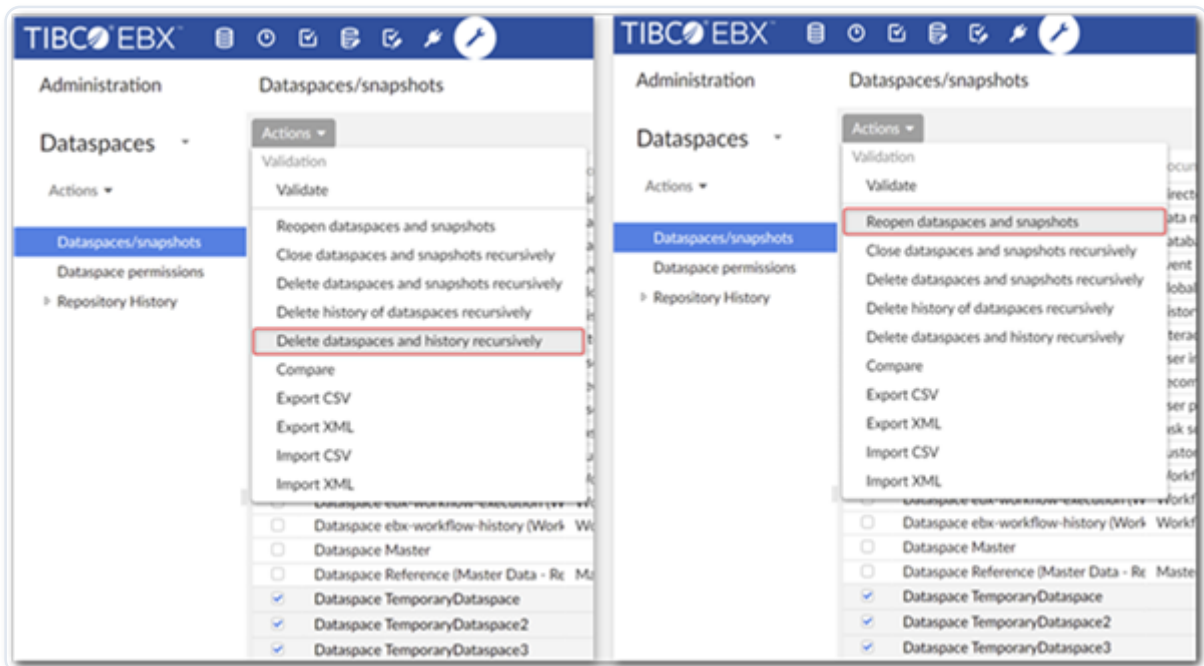
- URL ポリシー

データスペースのクリーンアップ

閉じられたデータスペースは移行中に削除されます。移行前に保持したいデータスペースを再度開く必要があります。移行処理時間を最適化できるように、保持しない閉じられたデータスペースは移行前に削除して消去することをお勧めします。

データスペースを管理するには、次の手順を実行します。

1. >> に移動します。
2. データスペースを再度開くか削除します。
 - 削除する場合は、削除するデータスペースを選択し、テーブルのメニューから **削除** を選択します。
 - 再度開く場合は、削除するデータスペースを選択し、テーブルのメニューから **再度開く** を選択します。



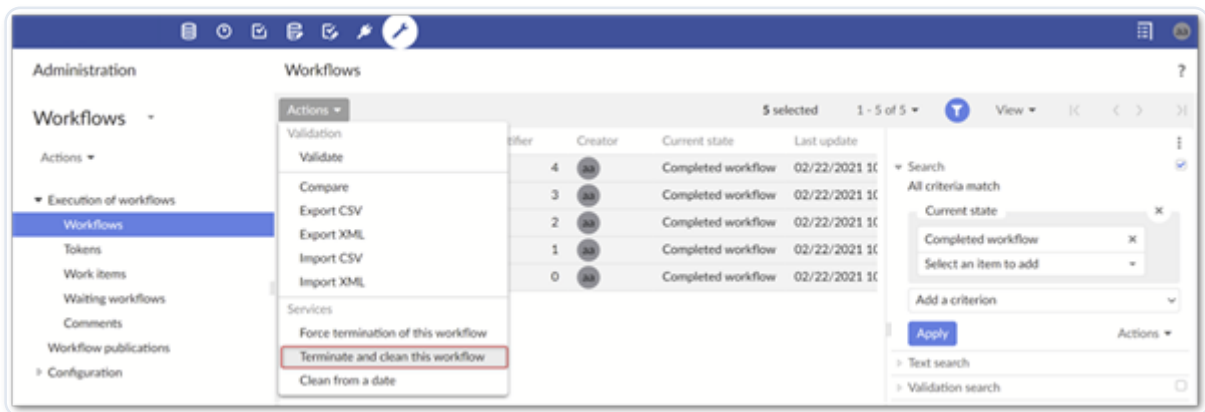
3. パージ後、G_BLK テーブルを確認します。以下のいずれかに該当する場合は、移行時間を短縮するための追加のクリーンアップについてサポートを受けるために、移行前にサポートに連絡してください。
 - G_BLK テーブルの行数が 1,000 万行を超える場合、または
 - ペイロードが 100 GB を超える場合、または
 - G_TRV テーブルの行数が 1 万行を超える場合

ワークフローのクリーンアップ

ワークフローのクリーンアップはオプションの手順です。ただし、すべてのワークフローテーブルは自動的に移行され、これらのテーブルは大きくなる可能性があるため、移行処理時間が長くなる可能性があります。ビジネスユースケースでワークフロー履歴が必要ない場合、または監査証拠の「保存」データベースに外部化できる場合は、テーブルを消去すると移行プロセスを高速化できます。

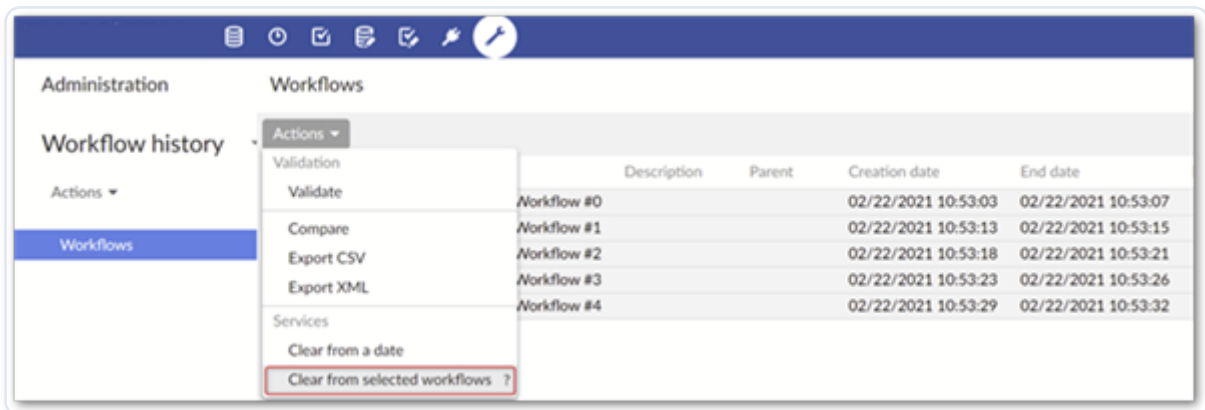
完了したワークフローを削除するには、次の手順を実行します。

1. >>> に移動します。
2. が に設定されているテーブルにフィルターを作成します。
3. すべてのレコードを選択し、テーブルの メニューから を選択します。



ワークフロー履歴を削除するには、次の手順を実行します。

4. >> に移動します。
5. 削除するすべてのレコードを選択し、テーブルの メニューから を選択します。



データモデルのクリーンアップ

移行プロセスを正常に完了するには、リポジトリのデータセットで使用するすべてのデータモデルがエラーなしでコンパイルされる必要があります。この厳格な要件は、データの損失を防ぎ、移行プロセスの終了時にリポジトリの一貫した状態を確保するために設けられています。また、廃止されたアドオンの成果物は、データモデルから削除する必要があります。

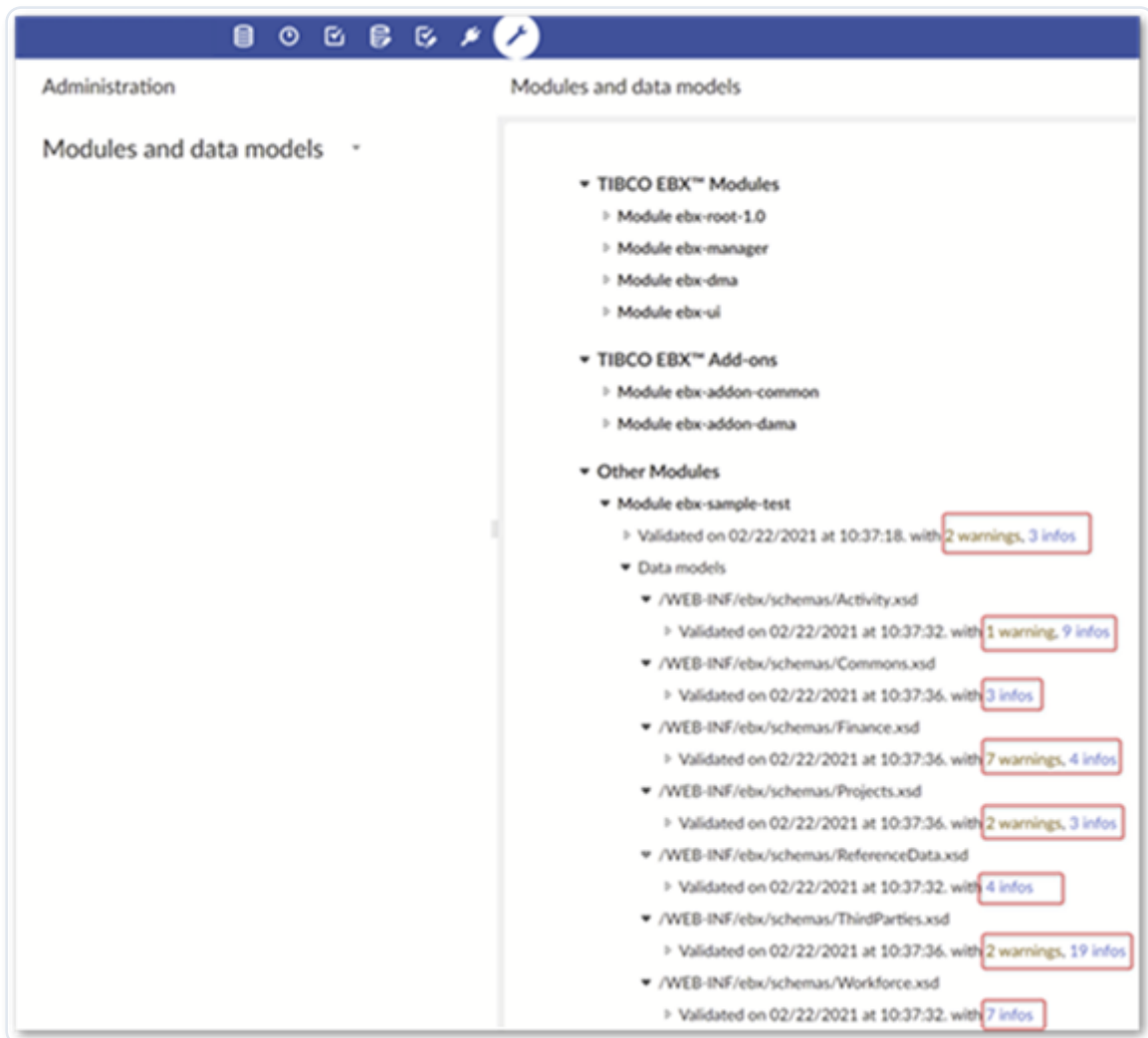
データモデルをクリーンアップするには、次のセクションで説明するトピックを参照してください。

- [データモデルエラー](#)
- [廃止されたアドオンのインクルード](#)
- [Java 拡張機能](#)

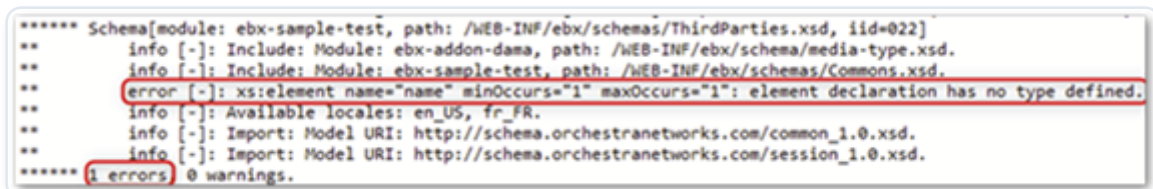
データモデルエラーの解決

データモデルエラーを解決するには、次の手順を実行します。

1. 次の場所を確認して、データモデルエラーを見つけます。
 - >> に移動し、データモデルエラーがないか確認します。



- `kernel.log` ファイルを開き、`error [-]` を検索します。



2. データモデルエラーを修正するために必要なアクションを実行します。または、データモデルが使用されなくなった場合は、削除できます。ただし、これは、子データスペースを含む、データモデルに基づいてデータセットを定義するすべてのデータスペースでも実行する必要があります。
3. アプリケーションサーバーを再起動し、テーブルとログを再確認して、まだエラー状態にあるモデルがないことを確認します。

廃止されたアドオンのインクルード

TIBCO EBX® Match and Merge Add-on (DAQA) がデータモデルに含まれている可能性があります。その場合、データモデルから include 命令とそのデータ型に基づくエレメントを削除する必要があります。

廃止されたアドオンに対処するには、次の手順を実行します。

1. `kernel.log` ファイルを開き、`***** Schema` で始まるデータモデルコンパイルレポートを検索します。レポートに `Include: Module: ebx-addon-daqa` という情報行が含まれているかどうかを確認します。

次の例では、カスタム モジュール `ebx-sample-test` に、Persons モデルへの包含が含まれています。

```
***** Schema[module: ebx-sample-test, path: /WEB-INF/ebx/schemas/Persons.xsd, iid=021]
**
** info [-]: Include: Module: ebx-addon-daqa, path: /WEB-INF/ebx/schema/ebx-addon-daqa-types.xsd.
** 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.
***** successful.
```

2. データモデルの XSD を開き、`/WEB-INF/ebx/schema/ebx-addon-daqa-types.xsd` を削除します。

```
<xs:schema xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0" xmlns:ebxnd="urn:ebx-schemas:binding_1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ebx="urn:ebx-schemas:session_1.0">
  <xs:import namespace="urn:ebx-schemas:common_1.0" schemaLocation="http://schema.orchestranetworks.com/common_1.0.xsd"/>
  <xs:import namespace="urn:ebx-schemas:session_1.0" schemaLocation="http://schema.orchestranetworks.com/session_1.0.xsd"/>
  <xs:include schemaLocation="urn:ebx:module:ebx-addon-daqa:/WEB-INF/ebx/schema/ebx-addon-daqa-types.xsd"/>
  <xs:annotation>
```

3. 定義されたタイプ `DaqaMetaData` を持つエレメントを検索して削除します。

```
<xs:element name="DaqaMetaData" type="DaqaMetaData" minOccurs="0">
  <xs:annotation>
    <xs:appinfo>
      <osd:history disable="true"/>
      <osd:defaultView>
        <displayMode>collapsed</displayMode>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

4. 上記の手順で変更されたエレメントを使用していた他のエレメントを検索して変更します。

```

<xs:element name="Persons" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation xml:lang="fr-FR">
      <osd:label>Persons</osd:label>
    </xs:documentation>
    <xs:appinfo>
      <osd:table>
        <primaryKeys>/uuid</primaryKeys>
        <index name="dq_index_complete"/>DaqaMetaData/ClusterId /DaqaMetaData/Score /DaqaMetaData/State /uuid</index>
        <index name="dq_index_light"/>DaqaMetaData/ClusterId /DaqaMetaData/Score /uuid</index>
        <index name="dq_index_score"/>DaqaMetaData/Score</index>
        <index name="dq_index_state"/>DaqaMetaData/State</index>
      </osd:table>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

廃止されたアドオンの Java 拡張機能

廃止されたアドオンによって提供される Java 拡張機能は、データモデルから削除する必要があります。すべての XSD ファイルを確認し、以下の非網羅的なリストへの参照をすべて削除する必要があります。

- TIBCO EBX 情報ガバナンス アドオン (IGOV) の場合は、`com.orchestranetworks.addon.igov.IGovLabelingSchemaDocumentation` の使用を確認します。
- TIBCO EBX ルール ポートフォリオ アドオン (RPFL) の場合は、データモデル拡張機能 `com.orchestranetworks.addon.rpfl.DefaultSchemaExtension` の使用を確認します。
- TIBCO EBX Information Search Add-on (TESE) の場合、テーブルフィルタ `com.orchestranetworks.addon.tese.SearchTableFilter` の使用を確認してください。このアドオンは廃止されていませんが、これを改良バージョンに置き換え、従来のバージョンを展開しない場合は、トリガーを削除する必要があります。
- TIBCO EBX Insight Add-on (DQID) の場合、トリガー `com.orchestranetworks.addon.dqid.controller.DQIdTrigger` の使用を確認してください。このアドオンは廃止されていませんが、これを改良バージョンに置き換え、レガシーバージョンを展開しない場合は、トリガーを削除する必要があります。

上記の項目を確認することに加えて、データモデルによって定義されたツールバーを確認することもお勧めします。ツールバーにはアドオンからのユーザーサービスが含まれることがあるためです。

URL ポリシー

5.9 の URL ポリシーは非推奨です。更新しないと、UI にアクセスしようとしたときに問題が発生する可能性があります。URL ポリシーが 6.x の要件に準拠していることを確認するには、>>>> に移動して、既存の設定を元に戻します。URL コンピューティング構成を更新するには、

`ebx.properties` ファイルを編集します。詳細については、[URL コンピューティング](#) を参照してください。

タスク 3: クリーンなリポジトリのバックアップ

現在のリポジトリをバックアップします。

1. EBX® 5.9 インスタンスをホストしているアプリケーションサーバーをシャットダウンします。
2. 5.9 リポジトリをバックアップします。名前が現在のリポジトリ プレフィックスで始まるリレーショナルデータベースオブジェクトをすべて含めます。このプレフィックスは、`ebx.properties` ファイルの `ebx.persistene.table.prefix` の下に指定されています。
3. EBX® ログをバックアップして削除するか、別のフォルダーに移動します。
4. アドオン構成をバックアップします。

TIBCO EBX® データモデルおよびデータ視覚化アドオン、または TIBCO EBX® GO アドオンをお持ちの場合は、構成アーカイブをエクスポートし、移行の完了後に再度インポートする必要があります。手順については、[アドオンの手動移行](#) を参照してください。

ステップ 3: 展開

概要

以下は、展開するための大まかな手順です。質問がある場合は、より詳しい情報とプラットフォーム固有の注意事項が記載されている『[TIBCO EBX® インストールガイド](#)』を参照してください。

注意:

アドオンを展開する場合は、アドオンバンドルに含まれる再パッケージ化 `ant` タスクを起動することが前提条件です。その場所は次のとおりです (詳細については、『[TIBCO EBX® Add-ons Versioning and Packaging Guide](#)』を参照)。`<Add-ons Bundle location>\script\ebxRepackageWithLGPL.xml`。

展開するには、次の手順を実行します。

1. アプリケーションサーバーを停止します

2. ライブラリを展開します

- a. EBX® コア製品の場合、次のファイルが `<EBX® 6 bundle location>\ebx.software\lib` にあります

- EBX® 6.0.x に移行する場合
 - EBX®: `ebx.jar`
 - 新しい圧縮アルゴリズムのサポート: `ebx-lz4.jar`
 - メールサポート: `jakarta.mail-2.0.5.jar`
- EBX® 6.x.x に移行する場合
 - EBX®: `ebx.jar`
 - バージョン 6.2.3 より前のバージョンに移行する場合にのみ、新しい圧縮アルゴリズムをサポート: `ebx-lz4.jar`
 - メールサポート: `jakarta.mail-2.0.5.jar`

- b. EBX® アドオン: `<EBX® Add-ons Bundle location>\lib\ebx-addons.jar`

- c. EBX® 6 を使用してコンパイルされたカスタムライブラリ

3. Web アプリケーションを展開します

- a. EBX® コア製品の場合 (`<EBX® 6 bundle location>\ebx.software\webapps\wars-packaging` にあります)

- `ebx-dataservices.war`
- `ebx-dma.war`
- `ebx-manager.war`
- `ebx-root-1.0.war`
- `ebx-ui.war`
- `ebx.war`
- `ebx-authentication.war` (バージョン 6.1.3 以降のみ)
- `ebx-ide.war` (バージョン 6.2.0 以降のみ)
- `ebx-hub.war` (バージョン 6.2.0 以降のみ)
- `ebx-assistant.war` (バージョン 6.2.2 以降のみ)

- b. EBX® アドオンの場合 (`<EBX® Add-ons bundle location>\wars` にあります)

- `ebx-addon-common.war` (必須)
- `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`

c. プロジェクト固有のすべての WAR ファイルは、EBX® 6 を使用してコンパイルされます

4. アプリケーションサーバーを起動します。自動移行プロセスにかかる時間は、移行されるデータの量によって異なります

注意:

移行を開始する前に、Java (Xmx) に可能な限り多くのメモリを割り当ててください。これにより、移行パフォーマンスが向上する可能性があります。移行後、割り当てられたメモリを物理メモリの 25% ~ 33% にリセットしてください。

移行が成功すると、`kernel.log` ファイルに次のメッセージが表示されます。

```
***** EBX® started and initialized *****
```

データモデルにまだエラーがある場合は、移行は停止します。`kernel.log` ファイルを使用して根本原因を特定し、[データモデルのクリーンアップ](#)の手順に従ってください。

Java API の変更

次のリソースには、バージョン間の Java API の変更がリストされています。移行プロセス中に、これらの変更を確認し、それに応じてカスタム コードを更新することが重要です。5.9 から 6.x に移行する場合は、両方のリンクの内容を確認する必要があります。リンクをクリックすると、TIBCO のドキュメント サイトで直接ドキュメントが開き、最新の変更にアクセスできることに注意してください。

- 6.0 と 6.x の間の変更点
- 5.9 と 6.0 の間の変更点

通知

- [ドキュメントとサポート](#)
- [Legal and Third-Party](#)

ドキュメントとサポート

この製品の詳細については、ドキュメントを読んだり、TIBCO サポートに連絡したり、TIBCO コミュニティに参加したりできます。

- [TIBCO ドキュメントへのアクセス方法](#)
- [製品固有のドキュメント](#)
- [TIBCO サポートへの問い合わせ方法](#)
- [TIBCO コミュニティへの参加方法](#)

TIBCO ドキュメントへのアクセス方法

TIBCO 製品のドキュメントは、[製品ドキュメント Web サイト](#)で、主に HTML および PDF 形式で入手できます。

[製品ドキュメント Web サイト](#)は頻繁に更新され、製品に含まれる他のどのドキュメントよりも最新です。

製品固有のドキュメント

TIBCO EBX® のドキュメントは、[TIBCO EBX® 製品ドキュメント](#) ページで入手できます。このページには各ドキュメントの最新バージョンが含まれています。

TIBCO EBX® アドオンのドキュメントは、[TIBCO EBX® アドオン製品ドキュメント](#) ページで入手できます。このページには各ドキュメントの最新バージョンが含まれています。

TIBCO EBX® の他のバージョンと互換性のあるアドオンバンドルのドキュメントを参照するには、[バンドルバージョンメニュー](#)を使用して目的のリリースを選択します。

TIBCO 製品のサポートへの問い合わせ方法

サポートチームには次の方法で連絡できます。

- サポートナレッジベースにアクセスし、関心のある製品に関するパーソナライズされたコンテンツを取得するには、弊社の[製品サポート Web サイト](#)にアクセスしてください
- サポートケースを作成するには、Cloud Software Group エンティティと有効なメンテナンス契約またはサポート契約を結んでいる必要があります。また、[製品サポート Web サイト](#)にログインするには、ユーザー名とパスワードが必要です。ユーザー名をお持ちでない場合は、Web サイトで [ここをクリックしてリクエスト](#)できます
- TIBCO 製品に関するアイデア、提案、またはフィードバックを送信するには、弊社の[製品サポート Web サイト](#)にアクセスしてください

TIBCO コミュニティへの参加方法

TIBCO コミュニティは、TIBCO の顧客、パートナー、従業員の対象分野の専門家が集合的なエクスペリエンスを共有し、アクセスするための公式チャンネルです。TIBCO コミュニティでは、Q&A フォーラム、製品 Wiki、ベストプラクティスへのアクセスが提供されます。また、拡張機能、アダプター、ソリューションアクセラレーター、およびお客様が TIBCO 製品の価値を最大限に活用できるようにするツールへのアクセスも提供されます。さらに、ユーザーは [TIBCO Ideas Portal](#) 内から機能リクエストを送信して投票できます。無料で登録するには、[TIBCO Community](#) にアクセスしてください。

Legal and Third-Party

Cloud Software Group, Inc. (以下「Cloud SG」) の一部のソフトウェアおよびクラウドサービスには、他の Cloud SG ソフトウェアを含む他のソフトウェアが組み込まれているか、バンドルされているか、またはその他の方法で含まれています (総称して「付属ソフトウェア」)。付属ソフトウェアの使用は、ライセンスされた Cloud SG ソフトウェアおよび/またはクラウドサービスの機能 (または限定的なアドオン機能) を有効にするためのみを目的としています。付属ソフトウェアは、他の Cloud SG ソフトウェアおよび/またはクラウドサービスによって使用またはアクセスされること、またはその他の目的で使用されることはライセンスされていません。

Cloud SG ソフトウェアおよびクラウドサービスの使用は、別途締結された契約、またはそのような別途の契約がない場合は、ソフトウェアまたはクラウドサービスへのアクセス、ダウンロード、またはインストール時に表示されるクリックラップエンドユーザー契約 (ライセンスファイルにも複製されています)、またはそのようなライセンス契約またはクリックラップエンドユーザー契約がない場合は、ソフトウェアの「LICENSE」ファイルにあるライセンスのいずれかに記載されている条件に従うものとし、本ドキュメントの使用は、これらの条件に従うものとし、本ドキュメントの使用は、これらの条件への同意および拘束されることへの合意を構成するものとします。

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.

本ドキュメントに記載されているその他すべての製品名、会社名、およびマークは、それぞれの所有者の財産であり、識別目的でのみ記載されています。これらのサードパーティのマークに対するすべての権利は、それぞれの所有者の独占的な財産であることを認識するものとします。詳細については、Cloud SG のサードパーティ商標通知 (<https://www.cloud.com/legal>) を参照してください。

本製品ドキュメントの公式バージョンは英語です。英語以外のバージョンは、お客様の便宜のためにのみ提供されており、機械翻訳されたコンテンツが含まれている場合があります。詳細については、<https://www.cloud.com/legal> の機械翻訳免責事項を参照してください。

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 ソフトウェアは複数のオペレーティングシステムで利用できる場合があります。ただし、特定のソフトウェアバージョンのすべてのオペレーティングシステムプラットフォームが同時にリリースされるわけではありません。特定のオペレーティングシステムプラットフォームでの特定のバージョンの Cloud SG ソフトウェアの利用可能性については、「readme」ファイルを参照してください。

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.

本ドキュメントには、技術的な不正確さや誤植が含まれている可能性があります。ここに記載されている情報には定期的に変更が加えられ、これらの変更は本ドキュメントの新版に組み込まれ

ます。Cloud SG は、本ドキュメントに記載されている製品、プログラム、および/またはサービスを、予告なくいつでも改善および/または変更する場合があります。

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.

Cloud SG の本製品およびその他の製品は、登録特許によって保護されている場合があります。詳細については、<https://www.cloud.com/legal> にある仮想特許表示ドキュメントを参照してください。



Copyright© 2006-2026. Cloud Software Group, Inc. All Rights Reserved

リリースノート

- [バージョン 6.2.3](#)
- [すべてのリリースノート](#)

6.2.3

- [6.2 サービスパック](#)
 - [6.2.3](#)
 - [6.2.2](#)
 - [6.2.1](#)
- [ワークフロー](#)
- [パフォーマンス](#)
- [データ検証](#)
- [アプリケーションサーバー](#)
- [データサービス](#)
 - [REST Toolkit API](#)
 - [組み込みRESTfulサービス](#)
- [組み込みLDAPディレクトリ](#)
- [プリフィルタによるパフォーマンス向上](#)
- [EBX® Container Edition](#)
- [EBX® Script IDE](#)
- [ホームページ](#)
- [メタデータ管理](#)
- [サイドバーでのレコードの履歴表示](#)
- [製品移行](#)
- [本リリースでの制限事項](#)
 - [階層の制限事項](#)

- クエリの制限事項
- 値ラベリングの制限事項
- データ検証の制限事項
- バージョンアップグレード
 - 新しいWebアプリケーションリソース (WAR) ファイル
 - コンパイルおよびビルド環境
 - サポート環境の変更
 - 既存のEBX® 6環境からのアップグレード
 - 下位互換性
- 6.2.3 既知の問題
 - ユーザーインターフェース
- 6.2.3 解決済みの問題
 - アシスタント
 - コアエンジン
 - データモデリング
 - データサービス
 - データ検証
 - データワークフロー
 - 分散データ配信
 - ドキュメンテーション
 - 階層
 - インフラストラクチャ
 - Mame
 - ユーザーインターフェース
 - ユーザーと権限
- 6.2.2 解決済みの問題
 - Container Edition
 - コアエンジン

- データモデリング
- データサービス
- データ検証
- データワークフロー
- ドキュメンテーション
- 階層
- インフラストラクチャ
- スクリプト
- ユーザーインターフェース
- ユーザーと権限
- 6.2.1 解決済みの問題
 - Container Edition
 - コアエンジン
 - データモデリング
 - データサービス
 - データワークフロー
 - ドキュメンテーション
 - 階層
 - インポート/エクスポート
 - インフラストラクチャ
 - メタデータ
 - スクリプト
 - ステージング
 - ユーザーインターフェース
 - ユーザーと権限
- 6.2.0 解決済みの問題
 - Container Edition
 - コアエンジン
 - データモデリング

- データサービス
- データ検証
- データワークフロー
- 階層
- インフラストラクチャ
- スクリプト
- Team Up
- ユーザーインターフェース
- ユーザーと権限

6.2 サービスパック

6.2.3

6.2.3 は 2026 年 4 月 29 日にビルドされました。このサービスパックの概要は次のとおりです。

Javaランタイム環境	JRE 25 LTS がサポートされるようになりました。JRE をこの最新のサポート対象 LTS バージョンにアップグレードすることをお勧めします。詳細については、「 サポートされているアプリケーションサーバー 」を参照してください。
サーブレットおよびJSPコンテナ	EBX® は Jakarta EE 10 以降を必要とするようになりました。この変更には、以下のサードパーティライブラリのアップグレードが必要です: Servlet は 6.0 以降、JSP は 3.1 以降、JMS は 3.1 以降、Mail は 2.1 以降。特定のコンポーネント要件の詳細については、以下のリンクを参照してください。詳細については、「 サポートされているアプリケーションサーバー 」、「 Javaメッセージサービス (JMS) 」、「 インストールノート 」、および「 WebLogic ASのサポート 」を参照してください。

新しいユーザーインターフェース (UI) のプレビュー	本リリースには、新しいEBX®ユーザーインターフェースのプレビューが含まれており、更新されたナビゲーションエクスペリエンスと新しいインタラクションパターンを提供します。このプレビューはオプションであり、管理者が有効にでき、現在のUIと並行して動作します。追加機能は今後のアップデートで導入される予定です。詳細については、『管理ガイド』および『ユーザーガイド』を参照してください。 注記: 新しいUIの使用はオプションです。既存のUIと並行して評価できますが、既存のUIが本番環境での推奨インターフェースです。
下位互換性 – リポジトリAPIの動作	本リリース以降、プロシージャからメソッド <code>Repository.deleteHome(AdaptationHome, Session)</code> を呼び出してはなりません。呼び出した場合、このメソッドは <code>IllegalStateException</code> をスローします。この変更により、トランザクションの不整合が防止され、データの整合性が確保されます。以前は、この呼び出しにより内部エラーが発生したり、ネストされたプロシージャが親プロシージャの前に変更をコミットした場合にリポジトリが一貫性のない状態になる可能性があります。 注意: この変更は、プロシージャ内からデータスペース (ブランチまたはスナップショット) を削除しようとするカスタムコードに影響を与える可能性があります。コードがこれに該当する場合は、プロシージャコンテキストの外に削除を移動する必要があります。

ワークフローの変更	履歴サイズの削減 - データコンテキスト変数は、更新された場合にのみ新しい履歴エントリとして記録されるようになりました。この変更により、ワークフロー履歴のサイズが時間の経過とともに大幅に減少する可能性があります。 削減される総量は、ワークフローモデルと、実行中の各ステップで更新される変数の数に完全に依存します。たとえば、20ステップで20変数が定義され、1ステップあたり平均2変数を更新するモデルでは、保存されるレコードの数が少なくとも90%削減されます。 注記: この変更は、新しいワークフロー実行にのみ適用されます。既存のワークフロー履歴は現状維持されます。 メッセージテンプレートの新しい組み込み変数 - 組み込み変数「workflow.workItem.key」が導入されました。これはメール内のワークアイテムキーに解決されます。
ワークフローユーザータスクの設定	ワークフローで以前のタスクを最も最近完了したユーザーに現在のタスクを割り当てる設定をできるようになりました。以前のタスクを完了したユーザーが現在のタスクが提供されるプロファイルのいずれかに属している場合、システムはそのユーザーにタスクを割り当てます。そのユーザーがそれらのプロファイルのいずれにも属していない場合、システムは通常どおり、対象となるプロファイルのユーザーにタスクを提供しま

	す。設定情報については、「ユーザー割り当て設定」を参照してください。 注記: この機能はレガシーユーザータスクでは利用できません。
EBX® Script	この言語では、次のことが可能になりました。 • スクリプト化されたRESTサービスを作成できます。これらのサービスは、EBX® 内のRESTエンドポイントとそれに関連するスクリプトを定義します。これらのサービスはScript IDEで作成および保守します。IDEからサービスをコンパイルして公開すると、RESTクライアントで利用できるようになります。 • 検証制約をサポートします。データモデル内のデータの一貫性を確保するために制約を使用します。 • ユーザー定義の重大度レベルでカスタマイズされたメッセージを作成できます。EBX® Scriptを使用して、検証レポートにメッセージを追加します。 • EBX® Script IDEを使用して、バンドルからローカライズされたメッセージを作成、変更、および公開できます。 参照: • スクリプト化されたRESTサービス • スクリプト化されたRESTサービスの作成チュートリアル • 制約検証

ワークフローを起動するための事前定義されたスケジュールタスク	新しい <code>SchedulerWorkflowLaunch</code> タスクは、ワークフローの起動をスケジュールします。このタスクでは、どのワークフロー公開が起動されるかを決定するために、ワークフローランチャーの入力が必要です。
組み込みLDAP	キャッシングシステムは高可用性をサポートするように改善されました。新しいプロパティ <code>[request].cache.refreshInSeconds</code> により、キャッシュエントリの有効期限が切れる前に更新できるようになりました。 詳細については、「LDAPディレクトリ」を参照してください。
EBX® メタデータ管理アプリケーション	- EBX® バージョン 6.2.0 以降、EBX® Container Edition を使用している場合にメタデータ管理アプリケーションが利用可能になりました。EBX® メタデータ管理アプリケーションを従来の (オンプレミス) デプロイメントにインストールできるようになりました。 - EBX® メタデータ分類子データベースは、Microsoft SQL Server および Oracle データベースをサポートするようになりました。 詳細については、「EBX® Container Edition のインストールノート」および「イメージのビルド」を参照してください。 - Spring Framework バージョンのアップグレードに伴い、メタデータエージェンツスクリプトが改訂されました。メタ

	データに接続するには、最新のエージェントスクリプトを使用してください。 メタデータエージェントのインストール手順については、「TIBCO® Metadata Agent のインストール」を参照してください。
EBX® Container Edition	JDK は LTS バージョン 25 にアップグレードされました。 環境変数を使用して EBX® 実行モードを定義できるようになりました。 メタデータ用の環境変数名のプレフィックスが <code>METADATA_</code> から <code>EBX_METADATA_CLASSIFIER_</code> に更新されました。古い変数名は下位互換性のために引き続きサポートされますが、将来のリリースで削除されます。 SQLite 以外のデータベースを使用する場合のメタデータ分類子用の Docker イメージのカスタマイズが簡素化されました。 EBX® の JNDI メールセッションを構成するには、Docker イメージをカスタマイズする必要があります。 詳細については、「概要」、「EBX® 実行モード」、「メタデータ分類子データベース接続」、「メタデータ分類子データベース用のドライバの追加」、および「JNDIでのメールの構成」を参照してください。
ストレージと圧縮	本リリースでは、ストレージ要件を削減するために、G_BLK形式と圧縮が完全に再設計されました。

- 新しい形式は、別のブロック (別の G_BLK テーブル行) との部分的なコンテンツ共有をサポートします。これにより、データスペース間の分離を維持しながら、データの重複が削減されます。
- 圧縮では辞書を使用できるようになり、効率が向上します。この改善は、新しい共有メカニズムでより一般的になった小さなブロックで特に顕著です。

これらの更新によって得られるストレージの節約は、次の範囲になると予想されます。

- 新規インポートデータの場合、10% から 25%。
- 専用データスペースで大幅に修正されたデータの場合、最大 90%。

圧縮速度が大幅に低下する可能性があるため、パフォーマンスに影響が出る場合があります。ただし、ほとんどの場合、これは問題になりません。

- 圧縮するデータが少ない (共有のため)。
- 大規模な初期インポートを除き、圧縮時間は完全な処理パイプラインと比較して無視できるレベルです。

圧縮パフォーマンスが問題となる場合は、`ebx.properties` で以下のパラメータを使用して調整してください。

- `ebx.compression.dictSize` は辞書サイズを制御し、デフォルトは 16 KB です。推奨範囲は 1 KB から 64 KB です。ただし、辞書が大きい場合に発生する以下のトレードオフに注意してください。

圧縮率 (対数進行) は向上しますが、圧縮は遅くなります (線形増加)。

注記:

この値は事前に設定する必要があります。EBX® が特定のテーブル定義の辞書を作成した後、別の辞書は作成しません。新しい値は、辞書を持たないテーブル、または新しいテーブル定義にのみ適用されます。

- `ebx.compression.level` は圧縮レベルを制御します。可能な値は FAST、BALANCED、および HIGH (デフォルト) です。レベルを下げると圧縮が高速化されますが、圧縮率は低下します。この設定は、次の圧縮操作にすぐに適用されます。この設定は、解凍には目立った影響を与えません。

圧縮パフォーマンスについては、Java 17 は完全にサポートされていますが、Java 21 または Java 25 での実行を強くお勧めします。最速のネイティブ圧縮実装は、次の環境でのみ利用可能です。

- Java 21 以降、および
- Linux または Windows、および
- x86_64 CPU アーキテクチャ。

その他の環境では、圧縮は完全な Java 実装にフォールバックします。フォールバックは遅く、最適化された辞書を作成できません。どちらの実装も完全に互換性があり、切り替えることができます。

開発者への影響: 最適化された圧縮実装は、直接/ネイティブメモリを使用するようになりました。これにより、メモリ (インデック

	スおよびデータベースリソース) が迅速に返されるように、 <code>RequestResult</code> および <code>QueryResult</code> オブジェクトをできるだけ早く明示的に閉じることの重要性が増します。 Public API の動作変更 : <code>ReverseRelationshipResult.getResult()</code> は以前は同じ <code>RequestResult</code> インスタンスを返していましたが、現在は使用後に閉じる必要がある新しい <code>RequestResult</code> を常に返します。 インストールおよびデプロイメントアーティファクト: 本リリース以降、 <code>ebx-lz4.jar</code> ライブラリは不要になり、削除されました。
クエリのハイライト	リンクされたフィールドでの検索時のハイライト計算が最適化されました。 特に大規模なセカンダリテーブルの場合、メモリ消費量が大幅に削減されました。以前は、各検索結果が完全なクエリコンテキストへの参照を保持していたため、メモリのクリーンアップが妨げられていました。これにより、多数の結果ページを閲覧する際に、各基になる結果がメモリに保持されるため、大きな影響が出る可能性がありました。 数百万レコードのボリュームでは、数ギガバイトの保持ヒープが発生し、ガベージコレクションのブロックにつながる可能性がありました。 主な改善点:

	• ハイライト計算は、完全なクエリコンテキストを保持する代わりに、軽量な構造を使用して遅延されるようになりました • 計算後、参照は即座のガベージコレクションのために解放されます • 保持ヒープの削減は、場合によっては50～100倍 (例: 10 GB超から約100 MB) になります。これは状況によって異なります。
サポートされているデータベース	PostgreSQL 18 がサポートされるようになり、PostgreSQL 13 はサポートされなくなりました。最小サポートバージョンは PostgreSQL 14 です。 Oracle 21c (Innovative Release) はサポートされなくなりました。Oracle の必須バージョンは 19c または 23c です。 SQL Server 2014 はサポートされなくなりました。最小サポートバージョンは SQL Server 2016 です。 詳細については、「サポートされているデータベース」を参照してください。
セッションタイムアウトのリダイレクトプロパティ	新しいプロパティにより、セッションがタイムアウトした後にユーザーをどこにリダイレクトするかを宣言できるようになりました。 詳細については、「セキュリティ設定」を参照してください。
SSO - データサービス	認証されたユーザーを識別するためにどのJSON Webトークン (JWT) クレームを使用するかを設定できるようになりました。

	詳細については、「 OpenID Connect 」および「 セキュリティ 」を参照してください。
既知の問題	「 6.2.3 既知の問題 」セクションを参照してください。
解決済みの問題	「 6.2.3 解決済みの問題 」セクションを参照してください。

6.2.2

6.2.2 は 2025 年 9 月 2 日にビルドされました。このサービスパックの概要は次のとおりです。

新機能: AIアシスタント	生成AIを利用するEBX®ユーザーをサポートするために、新しいAIアシスタント機能が導入されました。この機能は、大規模言語モデル (LLM) を活用して、ユーザーの入力とレコードデータに基づいて提案を生成することでユーザーを支援します。これは、テキストの生成、コンテンツの要約、値の翻訳、形式の標準化などのタスクに特に役立ちます。AIアシスタントを使用するには、管理者はEBX®を外部LLMプロバイダーに接続し、アシスタンス機能を定義する必要があります。各機能には、レコード内の1つ以上のフィールドに対して意味のある結果を生成する方法をLLMに指示するプロンプトが含まれています。 詳細については、以下のドキュメントリソースを参照してください。 • AIアシスタントには、外部LLM接続の構成とアシスタンス機能の定義に関する管理者向けの指示が含まれています。
---------------	---

	データの編集には、AIアシスタントの使用 方法に関する指示が記載されていま す。
EBX® Script	EBX® Script言語は、スクリプトから外部 RESTサービス呼び出すことをサポートす るようになりました。 ロールマッパースクリプトは、IDプロバイ ダーからの外部ロールを内部EBX®ロールに マッピングするための柔軟なアプローチを 提供します。 詳細については、以下のドキュメントリソー スを参照してください。 - 外部サービス構成では、データの暗号化 に使用される構成プロパティとキースト アファイルについて説明しています。 - RESTサービスには、スクリプトで RESTサービス呼び出すための手順と 例が含まれています。 - ロールマッパー。
同期ディレクトリ	シングルサインオン (SSO) 構成で外部ユー ザーとロールを処理するために、新しい同 期ディレクトリが導入されました。さまざ まな外部ユーザーディレクトリとの統合を可能 にするために、LDAP、SCIM、OnLogin の 組み込み実装が提供されています。 詳細については、「同期ディレクトリ」を参 照してください。 下位互換性: LDAP および SCIM の場合、外 部ディレクトリからすべてのロールをイン

	ポートし、同じ権限を使い続けるには手動移行が必要です。 詳細については、「移行」を参照してください。
ライセンスキー	ライセンスキーマネージャーが復活しました。ライセンスキーは、EBX® インスタンスの契約タイプ (サブスクリプション、永続、またはトライアル) に合わせて調整されるようになりました。デプロイされたアドオンは、システム起動時に登録および有効化されます。 詳細については、「EBX® ライセンスキープロパティファイルの設定」、「EBX® Container Edition の構成」、および「<code>TestHelper.prepareEnvironment</code>」を参照してください。
データ検証	最適化された検証: パターンファセットの検証が最適化されました。Luceneインデックスの利用により、大規模なテーブルでも使用できるようになりました。つまり、フィールドがこのファセットを定義している場合、検証プロセスは、関連するフィールドの値が指定されたパターンに準拠しているかどうかを確認するために、すべてのレコードを反復処理することはありません。検証プロセスに関する詳細については、「データ検証」を参照してください。 制約の事前検証の無効化: ユーザーインターフェースで、関連フィールドがユーザー入力で更新されたときに、制約が動的に事前検証されるのを防ぐ新しいメソッドが追加されました。これは、パフォーマンスを向上させたり、フォームの更新中に不要な検証エラーを回避したりするのに役立ちます。ただ

	し、フォームが送信されたとき、またはビューでレコードがロードされたときには、制約は引き続き検証されます。 詳細については、 「<code>ConstraintContext.disableAjaxPrevalidation</code>」および 「<code>ConstraintContextOnTable.disableAjaxPrevalidation</code>」を参照してください。
ワークフローの変更	ワークフロービューに新しい変更が加えられました。詳細は以下を参照してください。
テーブルビューを取得するためのJava API	UIコンテキストとは独立してテーブルビューを取得できるようになりました。これは、テーブルに関連するすべてのビューを特定のユーザーに対して取得するのに役立ちます。 詳細については、 「<code>UICustomViewContextFactory.newCustomViewContext</code>」を参照してください。
データモデルを取得および公開するためのJava API	Data Model Assistant で定義されたデータモデルを取得および公開するための新しいJava API が追加されました。 詳細については、 「<code>DataModelingManager</code>」および 「<code>DMADataModel</code>」を参照してください。
データサービス	Data Model Assistant で定義されたデータモデルの公開をサポートする新しい操作が追加されました。 詳細については、「データモデル操作」を参照してください。
Tomcat 11のサポート	EBX® バージョン 6.2.0 以降は Tomcat 11 と互換性がありません。

パフォーマンスの向上とリソースの最適化

- 中間オブジェクトの割り当てを削減することで、CSVインポート操作の速度が向上しました (スループット約10%増)。
- 「onDemand」レプリケーションユニットの更新と「onCommit」レプリケーションユニットの初期同期の改善により、より効率的なリソース使用と、中間オブジェクトおよびデータベースとのラウンドトリップの削減が実現しました。
 - レプリカが空であると仮定できる場合、差分は計算されず、挿入が直接適用されます。
 - 差分は、適用前にメモリ構造に格納されるのではなく、オンデマンドでデータベースに適用されます。
- 新しいインデックス事前ロード構成オプション: 起動時にデータスペースインデックスを非同期プロセスとして自動的に再構築するタイミングを指定できます。起動はブロックされませんが、不足しているインデックスが準備され、最初のアクセス前に利用可能になります。詳細については、「[高度なインデックス構成](#)」を参照してください。
- ディスク上のインデックスサイズ:
 - インデックスのリベースとより積極的なマージポリシーにより、インデックスの効率が向上し、インデックスチェーンが短縮されます。これにより、初期化が高速化され、メモリ消費量が削減されます。

- データスペースにアタッチされなくなったテーブルリビジョンのインデックスはパーズされます。これにより、インデックスの数、サイズ、初期化時間、およびオブジェクト数が削減されます。
- `IndexWriter` および `IndexReader` のライフサイクル管理が改善され、使用できなくなったファイルが確実に削除されるようになりました。デフォルトでは、Lucene は未削除のファイルを問題なしと見なします。これは、`IndexWriter` が次に開かれたときに削除される可能性があるためです。ただし、EBX® では、一部のインデックスは読み取り専用になります。この改善以前は、`IndexWriter` は再オープンされず、ファイルは削除されませんでした。
- Elias-Fano エンコーディングがライブドキュメントの保存に使用されるようになりました。これは、ストレージを節約できる場合はすぐにディスク上で、メモリの少なくとも 50% を節約できる場合はメモリ内で使用されます。一部のクエリが遅くなる可能性があります。
- メモリを節約するために、共通の文字列が重複排除され、小さく、長寿命で、ほとんど不変のコレクションに対してより効率的な実装が適用されました。
- Java ヒープデータ構造上のインデックスプーリングの一般化: これは、インデッ

クスリーフ上の主キーブルームフィルターをプールする6.2.1の改善と同等であり、すべてのインデックスファイルタイプに適用されます。Javaヒープデータ構造上のインデックスは、ブルームフィルター上のインデックスよりもはるかに小さいです。ただし、節約されたメモリにより、より多くのインデックスを開いたままにでき、それらの共有 (またはプーリングが適用されない場合の重複) を増やすことができます。この更新は、ファイルがアクセスされるたびにチェックサムを必要としないため、初期化時間を改善することもできます。さらに、OSメモリマップファイル記述子などの他のリソースへの負荷を軽減するのに役立ちます。

- Lucene は 9.4 からバージョン 9.12.1 にアップグレードされました。このアップグレードと上記で言及されたその他のアップグレードには、完全なインデックス再構築が必要です。ただし、インデックス再構築の改善と新しいインデックス事前ロードサービスにより、以前のバージョンよりも高速になります。Lucene 9.12 は現在、最後の Lucene 9 マイナーバージョンとなる予定です。Lucene 10 へのアップグレードには EBX® を Java 21 でコンパイルする必要があるため、インデックス形式は安定化され、近い将来の新しいリリースでは完全なインデックス再構築は不要になります。

EBX® Container Edition

EBX® の JNDI データソースを構成するには、Docker イメージをカスタマイズする必要があります。

	詳細については、「JNDIでのデータベースの構成」を参照してください。
解決済みの問題	「6.2.2 解決済みの問題」セクションを参照してください。
6.2.1 リリースノートへの追加	EBX® 6.2.1 リリースに含まれていましたが、リリースノートに記載されていなかった内容は次のとおりです。 • 完全なテーブルリビジョン再設計: ◦ 圧縮を使用せずにストレージ効率を向上させる新しいエンコーディングが導入されました。これにより、速度が向上し、リソースへの要求が軽減されます。 ◦ G_SHR データベーステーブルの更新により、類似した (ただし異なる) 行間での共有が可能になりました。新しいエンコーディングと共有検出により、本番環境で G_TRV と G_SHR テーブルの合計サイズが大幅に削減されました (サイズは 90GB から 800MB 未満に削減)。 ◦ 複数のレベルでテーブルリビジョンを表す新しいインメモリデータ構造が使用されます。これは、メモリ消費を削減するためのデータ共有をサポートします。コピーが軽量になり、プロシージャとクエリ実行の分離が高速化されます。 • インデックスファイルでのチェックサム計算が高速化されました。これにより、ファイルを1バイトずつ読み取ることを

回避することで、インデックスのオープン時間が高速化されます。

- インデックスリーフ上の主キーブルームフィルターのプーリングが更新されました。以前は、Javaヒープ上のインデックスを表すために使用される構造は、複数のデータスペースで使用されるディスク上の同じファイルを表しているにもかかわらず、共有されていませんでした。新しいインデックス形式により、このタイプのファイル (最大数百MBのメモリを消費する) の共有が可能になります。EBX®をアップグレードする際にインデックスの再構築は必須ではありませんが、この改善を活用するには必要です。実際の例では、同じ大きなファイルが複数のデータスペースで使用される場合 (ヒープダンプでは、以前は同じ512MBのファイルが22回開かれ11GBのRAMを使用していたが、現在は512MBしか使用しない)、大量のメモリを節約できます。
- インデックスリーフ上の主キーブルームフィルターのサイズは、偽陽性確率を上げることで削減されました。これはパフォーマンスに大きな影響を与えませんが、ディスクとメモリの消費量を2分の1にすることができます (ほとんどの場合)。
- 読み取り専用と見なされるインデックス上のインデックスライターは開かれなくなりました。オブジェクトが少なくなることで、メモリ消費量とGCへの負荷が軽減されます。

6.2.1

6.2.1 は 2025 年 3 月 19 日にビルドされました。このサービスパックの概要は次のとおりです。

インデックス作成パフォーマンスの向上

最適化されたインデックス再構築チェーン:

インデックス再構築は、再構築チェーンを短縮するためにステップをスキップするようになり、ベースインデックスをより効果的に再利用することで、再構築期間とインデックス構造が改善されます。この変更により、中間スナップショットの再構築時間が短縮されます。ただし、スキップされたスナップショットにアクセスするには、それらのインデックスを最も近い利用可能なポイントから増分的に再構築する必要があるため、わずかに時間がかかる場合があります。

増分再構築の実現可能性の動的評価: EBX®

は、テーブルリビジョン間の「距離」を推定して、増分インデックス再構築を実行するか、完全なインデックス再構築を実行するかを決定するようになりました。ベースインデックスとターゲットインデックスの差が大きすぎる場合、完全な再構築が開始されます。この調整により、再構築が高速化され、インデックス作成がより効率的になります。この分岐点よりも古いスナップショットにアクセスするには、新しい再構築チェーンを最初から計算する必要があるため、時間がかかる場合があります。

インデックス再構築のバッチ処理: インデックス再構築操作はバッチ処理されるようになり、削除と作成のアクションをグループ化して効率を高めます。

構成可能なマルチスレッド再構築:

`ebx.index.threads` プロパティを使用すると、インデックス再構築中に使用されるス

	レッド数を制御できます。スレッド数を増やすと、特に更新、移行、または大幅なデータモデル変更後に多数の大きなインデックスがあるコンテキストで、インデックス再構築を高速化できます。ただし、通常の操作では、デフォルト設定の「2」スレッドが推奨されます。 詳細については、『パフォーマンスとチューニング』章の「アーキテクチャと設計」を参照してください。
コアエンジン	主キーのルックアップを高速化するために使用される確率的フィルターのメモリフットプリントは、偽陽性確率を上げることで削減されました。より高い精度は、追加のメモリ消費とチェックするハッシュ関数に見合うものではありませんでした。 一部の操作は、不要な場合にデータセットを表す重いオブジェクトのロードを回避するように改善されました。これにより、たとえば、関連するリレーションシップのターゲットテーブルを解決するためにデータセットにアクセスする必要がある外部キーおよび継承フィールドに対するリクエストが強化されます。
クエリの制限事項	継承されたデータセット内のすべてのフィールドを管理する際のパフォーマンスが最適化されました。 詳細については、「クエリの制限事項」を参照してください。
EBX® Script	データセットトリガーを作成できるようになりました。データセットトリガーは、データセットに対して特定の操作が実行さ

	れたときに暗黙的かつ自動的に実行されるプロシージャを定義します。 詳細については、「データセットトリガー」を参照してください。
シングルサインオン (SSO)	EBX® は SAML 2.0 と OpenID Connect プロトコルの両方をサポートするようになりました。これらの機能を有効にするには、EBX® を ID プロバイダーでアプリケーションとして宣言し、有効なプロトコルに応じて構成する必要があります。 詳細については、「シングルサインオン (SSO)」を参照してください。
SCIM機能	SCIM (System for Cross-domain Identity Management) の組み込みサポートが追加されました。SCIM 機能は、IdP (ID プロバイダー) と EBX® 間でのユーザー ID とロールのプロビジョニング、管理、同期を自動化します。 詳細については、「SCIMディレクトリ」を参照してください。
EBX® Container Edition	環境変数を使用して、シングルサインオン (SSO)、ベアラー認証、SCIM機能、LDAP組み込みディレクトリ、およびカスタムディレクトリを構成できるようになりました。 詳細については、「SSO SAML 2.0接続」、「SSO OpenID Connect接続」、「ベアラー認証」、「SCIMディレクトリ」、「LDAP接続」、および「ディレクトリ接続」を参照してください。
データサービス	組み込みRESTfulサービス、SOAP操作、およびカスタムRESTサービス (REST Toolkit を使用して構築) で「ベアラー認証スキーム

	メソッド」を使用できるようになりました。 REST Toolkit の「例外処理」セクションが、カスタム例外マッパーの説明で更新されました。 詳細については、「REST認証」および「SOAP認証」を参照してください。
サポートされているデータベース	以下のデータベースのサポートが追加されました: Microsoft Azure Database for PostgreSQL (PostgreSQL 16 以降と互換性あり)。 PostgreSQL 17 がサポートされるようになり、PostgreSQL 12 はサポートされなくなりました。PostgreSQL の最小サポートバージョンは 13 です。 詳細については、「サポートされているデータベース」を参照してください。
ユーザーインターフェースの検証	APV の動作をカスタマイズするための新しいオプションが追加されました。 参照: • APV は無効な入力を DataContext に保存する必要があります
セキュリティ - ログインページ	ユーザーがログインまたはログアウトし、カスタムURLが提供された場合、EBX®は <code>resume</code> クエリパラメータを追加します。以下のプロパティを使用することで、この動作を無効にできるようになりました。 • <code>ebx.security.loginPage.appendResumeUrl</code> • <code>ebx.security.exitPage.appendResumeUrl</code>

	• <code>ebx.security.exitErrorPage.appendResumeUrl</code> 詳細については、「セキュリティ設定」を参照してください。
セキュリティ - HTMLヘッダー	追加のJavaScriptファイルを含めるため、またはCSPをカスタマイズするための新しいオプションが追加されました。 • <code>ebx.security.csp.script-src</code> • <code>ebx.security.csp.style-src</code> • <code>ebx.security.csp.img-src</code> • <code>ebx.html.header.javascript</code> 詳細については、「セキュリティ設定」および「HTMLヘッダー」を参照してください。
サイドバーでのレコードの履歴表示	多値リスト型を含む、単純型と複合型の両方のデータ型を表示できるようになりました。さらに、履歴トランザクションをフィールドでフィルタリングすることも可能です。
HTML入力	以前は、CKEditorライブラリがHTML入力に使用されていました。 セキュリティとライセンス上の理由から、オープンソースツールTipTapをベースとしたウィジェットに置き換えられました。 その結果、ルックアンドフィールが変更されました。以下に示すように、一部の機能は移動または削除されました。 ルックアンドフィール • 全体的なスタイルが見直されました • エディタのサイズは固定ではなくなり、コンテンツに合わせて調整されるようになりました。

UI管理の「[人間工学とレイアウト](#)」にある以下の4つの新しい設定で制限をカスタマイズできます: **フォーム: HTMLエディタの最小幅**、**フォーム: HTMLエディタの最大幅**、**フォーム: HTMLエディタの最小高さ**、および**フォーム: HTMLエディタの最大高さ**。

- エディタの高さはワークスペースのコンテンツの高さを超えることはできません。
- 入力支援が追加されました。たとえば、ユーザーが「- foo」と入力すると、テキストが箇条書きリストに変換されます。

ツールバー

- フォントサイズメニューは、ユーザーが異なる見出しレベルを選択できる見出しメニューに置き換えられました。
- 取り消し線テキストスタイルボタンが追加されました。
- コピー/切り取り/貼り付けツールバーは削除されました (キーボードショートカットとコンテキストメニューは引き続きサポートされます)。
- リンクツールバーは削除されましたが、リンクは引き続きサポートされます。これはユーザーにとって透過的になりました。

リンクはコピー/貼り付けで作成されるようになりました。ユーザーがテキストを選択中にリンクを貼り付けると、選択範囲がリンクになります。ユーザーがテキストを選択していない場合、リンクはそ

	のテキストをリンク値として作成されま す。 • ブロック引用テキストスタイルボタンが追加されました。 • テキストの背景色と前景色メニューが見直されました。 • 印刷ボタンとプレビューボタンは削除されました。 • 最大化ボタンは削除されました。 • 画像メニューはエディタコンテンツとインラインに移動しました。ユーザーが画像をクリックすると表示されます。 • ツールバーは展開できなくなり、1行に縮小されました。
バックグラウンドタスク	設定されたタスクはバックグラウンドで実行できるようになり、ユーザーはタスクの完了を待つことなくEBX®での作業を続行できます。一元化されたユーザーインターフェース(バックグラウンドタスクサイドバー)により、ユーザーはこれらのタスクを表示およびアクセスできます。 詳細については、「バックグラウンドタスク」を参照してください。
テーブルビューの検証ポリシー	テーブルビューでレコードを表示する際に使用する検証ポリシーを定義できるようになりました。このオプションは、関連するテーブルのデフォルトビュー、および/またはすべてのテーブルのデータモデルのグローバルプロパティで指定できます。 詳細については、以下を参照してください。 • <code>ValidationPolicyForTableViewEnum</code>

	• <code>SchemaDefaultTableView</code> • <code>SchemaNode.getSchemaDefaultTableView</code>
クイック検索結果のハイライト	クイック検索結果がハイライト表示されるようになり、視認性が向上しました。 詳細については、「クイック検索結果のハイライト表示」を参照してください。
リンクされたフィールドでのハイライトのパフォーマンス	リンクされたフィールドで検索する際の動作が改善され、リンクされたテーブルでのハイライト計算が表示された要素のみに制限されるようになりました。
最適なクエリプラン計算フェーズ中のパフォーマンス	Calciteオプティマイザが使用する一部の頻度計算、特にLuceneフィルターの選択性を決定してクエリを実行する最適な方法を見つけるための計算が改善されました。
新しいソフトウェア配信サイト	EBX® ソフトウェアを TIBCO eDelivery からダウンロードできるようになりました。既存の eDelivery アカウントをお持ちの場合は、既存の資格情報を使用して新しいソフトウェア配信サイトにログインできます。
解決済みの問題	「6.2.1 解決済みの問題」セクションを参照してください。

Java API

Java API は以下のトピックで改善されました。

- `WidgetDisplayContext`

新しいメソッド `WidgetDisplayContext.isInherited()`。

- `PerspectiveTheme`

新しいメソッド `getButtonsHighlightTextColor()`、`getFormHTMLEditorsMaxHeight()`、`getFormHTMLEditorsMaxWidth()`、`getFormHTMLEditorsMinHeight()`、

`getFormHTMLEditorsMinWidth()`、`getPerspectiveNavigationPaneSelectionTextColor()`、および
`getWorkflowBadgeTextAndBorderColor()`。

- `ValidationPolicyForTableViewEnum`

新しい列挙型 `ValidationPolicyForTableViewEnum`。

- `SchemaDefaultTableView`

新しいインターフェース `SchemaDefaultTableView`。

- `SchemaNode`

新しいメソッド `SchemaNode.getSchemaDefaultTableView()`。

- `DifferenceHelper`

新しいメソッド `compareDatasetByDate(Adaptation,boolean,java.util.Date,java.util.Date)`、
`compareDatasetByTransactionID(Adaptation,boolean,long,long)`、
`compareDataspaceByDate(AdaptationHome,boolean,java.util.Date,java.util.Date)`、
`compareDataspaceByTransactionID(AdaptationHome,boolean,long,long)`、
`compareRecordByDate(Adaptation,boolean,java.util.Date,java.util.Date)`、
`compareRecordByTransactionID(Adaptation,boolean,long,long)`、
`compareTableByDate(AdaptationTable,boolean,java.util.Date,java.util.Date)`、 および
`compareTableByTxId(AdaptationTable,boolean,long,long)`。

- `MergeSpecForInstanceContents`

新しいコンストラクタ `MergeSpecForInstanceContents(boolean)`。

- `UserTask.WorkItem`

新しいメソッド `UserTask.WorkItem.isCommentEnabled()`。

- `WorkflowEngine`

新しいメソッド

`WorkflowEngine.getProcessInstance...History(PublishedProcessKey,String,String)`。

注記: 6.2.0 は 2024 年 6 月 21 日にビルドされました。

ワークフロー

ワークフローに関する注目すべき点:

ワークフロービューのパフォーマンス

- データワークフロービューの特定の列のパフォーマンスが向上しました。特定の列はソート可能であるべきであり、それらに対するフィルタリングが高速化されました。しかし、トレードオフとして、組み込みワークフロー変数はサポートされなくなり、データコンテキスト変数のみがサポートされます。式は、表示時ではなく、ワークフローデータコンテキストが更新されるたびに評価されるようになりましたが、結果は以前の動作とユーザーにとってシームレスであるはずで

す。

ワークフロー関連テーブル (アクティブ/完了済みワークフローおよびワークフロー管理) のラベル (dwfLabel) を置き換える新しいプロパティが追加されました。

公開関連テーブル (公開およびワークフロー管理) の公開の現在の公開名 (dwfPublicationLabel) を置き換える新しいプロパティが追加されました。

ワークアイテム関連テーブル (受信トレイ、ワークアイテム、およびワークフロー管理) のワークアイテムラベルと説明 (workItemLabel および workItemDescription) を置き換える新しいプロパティが追加されました。

ワークフロー関連テーブル (アクティブ/完了済みワークフローおよびワークフロー管理) のワークフロー現在のステップ (dwfTokenCurrentStep)、ワークフロー説明 (dwfDescription)、ワークフロー作成者 (dwfWorkflowCreator)、現在

のユーザー

(dwfProcessInstanceCurrentUser) を置き換える新しいプロパティが追加されました。

ワークアイテム関連テーブル (受信トレイ、ワークアイテム、およびワークフロー管理) のユーザー (allocatedTo)、提供先 (offeredTo) を置き換える新しいプロパティが追加されました。

ワークアイテム関連テーブル (受信トレイ、ワークアイテム、およびワークフロー管理) のコメント (dwfComments) を置き換える新しいプロパティが追加されました。これは特殊で、古いプロパティは表示用ですが、フィルタリング用は新しいプロパティです。これらは同じラベルを共有します。

新しいプロパティは現在の以前のプロパティのローカライズされたラベルを使用し、古いプロパティは非推奨として追加されます。新しいプロパティは正しくフィルタリング可能になりましたが、まだソートはできません。フィルタリングは高速化されました。

他のいくつかのプロパティもこれらの変更の影響を受けます: 受信トレイ/ワークアイテムビューのデータワークフロー、受信トレイ、ワークアイテム、アクティブ/完了済みワークフローの公開ラベル、およびアクティブ/完了済みワークフロービューのワークフロー親。これらは新しいプロパティにマッピングされ、フィルタリングは高速化されますが、ソートは

まだ解決されておらず、当面はデフォルトで主キーによるソートが行われます。

ラベルの一部を検索する「contains」のようなフィルターを使用する場合、2番目のフィルターを同じ値の検索フィルターとするANDフィルターを試すことができます。これにより、結果がより速く返される可能性があります。

ワークフローの最適化

- 受信トレイカウンターの最適化:** ワークフロー受信トレイカウンターは、その値をより効率的にキャッシュするようにリファクタリングされ、データベース負荷を大幅に削減し、応答時間を改善しました。新しい実装は、高度なキャッシュメカニズムと最適化されたクエリを活用し、受信トレイの読み込みを高速化し、特に多数のワークフロータスクがある環境で、より応答性の高いユーザーインターフェースを実現します。
- 期限計算の改善:** 期限計算プロセスがより効率的になるように強化されました。以前は、期限は動的に計算され、カウンターが更新されるたびに各期限の再計算が必要でした。現在、期限はワークアイテム作成時に計算され、適切にキャッシュされるため、期限付きの多くのワークフローが同時に実行されている場合のシステムリソース使用量が削減されます。この最適化により、動的な再計算の必要がなくなり、特に複雑な要件を持つ環境でのパフォーマンスが向上します。
- 適応フィルターの強化:** ワークフローコンテキストで使用される適応フィルターは、パフォーマンスを向上させるために

	最適化されました。期限計算での使用が削減され、新しいバッチ計算アプローチが以前の動的フィルタリングシステムに置き換わったため、より体系的に適用されることはなくなりました。
ワークフロー履歴	ワークフロー履歴管理で利用可能なサービス「選択したワークフローからクリア」および「日付からクリーンアップ」を修正しました。6.0以降、ワークフローテーブルのレコードのみがクリアされ、関連する一部のデータが無視されていたため、ユーザーの知らないうちに履歴が肥大化していました。ワークフロー管理ドキュメントに記載されているように、標準のページタスクを定期的に行うことを引き続き推奨します。 参照: ワークフロー管理

パフォーマンス

クエリパフォーマンスまたはテーブル参照検証を向上させるために、`AdaptationFilter` または `TableRefFilter` にプリフィルタを定義できるようになりました。

詳細については、それらのJavaドキュメントまたは「[プリフィルタによるパフォーマンス向上](#)」セクションを参照してください。

データ検証

このサービスパックは、検証レポートを含む多数のデータセットを持つデータスペースから子データスペースを作成する際のパフォーマンスを向上させます。この改善の一環として、検証レポートのメタデータスキーマが変更されました。その結果、EBX® 6.0 または 6.1 から 6.2 以降にアップグレードする際に、すべての検証レポートがリセットされます。リセットされた検証レポート内の古いデータは、次のページ中に削除されます。

アプリケーションサーバー

このセクションでは、アーキテクチャの近代化のためのアプリケーションサーバーの変更について説明します。

サーブレットおよびJSPコンテナ	EBX® は Jakarta EE 9 以降をサポートするようになりました。この変更には、Servlet 5.0 以降、JSP 3.0 以降、JMS 3.0 以降、Mail 2.0 以降など、いくつかのサードパーティライブラリのアップグレードが必要です。特定のコンポーネントの要件に関する詳細については、以下のリンクを参照してください。EBX® 6.2.x 以降の開発者は、Java EE から Jakarta EE への移行の結果、実装されたすべての API のプライマリパッケージ名が <code>javax.*</code> から <code>jakarta.*</code> に変更されたことに注意する必要があります。これにより、EBX® 6.1 以前から EBX® 6.2 以降にモジュールやその他のライブラリを移行するには、ほぼ確実にコード変更が必要になります。 詳細については、「サポートされているアプリケーションサーバー」、「Javaメッセージサービス (JMS)」、「分散データ配信 (D3) 用JMS」、および「インストールノート」を参照してください。
Javaコンパイル時環境	EBX® のソースは JDK 17 でコンパイルされるようになりました。JRE 11 は互換性がありません。パフォーマンスの向上を活用するために、JRE を最新の LTS バージョンにアップグレードすることをお勧めします。 詳細については、「サポートされているアプリケーションサーバー」を参照してください。

データサービス

以下のセクションでは、データサービスの新しい機能と改善点について説明します。

REST Toolkit API

REST Toolkit は [Jakarta RESTful Web Services 3.1](#) をサポートするようになりました。

組み込みRESTfulサービス

組み込みRESTfulサービスは以下の領域で改善されました。

ビジネスオブジェクト

`data-bo` および `data-compact-bo` を使用したビジネスオブジェクト選択操作を改善し、ビジネスオブジェクトの1対多リレーションシップのコンテンツを返すようになりました。応答が大きくなる可能性があるため、

`ebx.dataservices.rest.bo.maxResponseSizeInKB` プロパティを使用して EBX® サーバーで最大サイズをオーバーロードできます。実際、応答が最大サイズ値に近づくか超える場合、要求されたレコード数よりも少ないレコード数を含む HTTP `206 Partial Content` 応答が返され、一貫したページングコンテキストも含まれ、すべてが整形形式の JSON 応答になります。

詳細については、「[ebx.dataservices.rest.bo.maxResponseSizeInKB](#)」、「[組み込みRESTfulサービスの下位互換性](#)」、「[パフォーマンス](#)」、および「[選択操作](#)」を参照してください。

組み込みLDAPディレクトリ

LDAP (Lightweight Directory Access Protocol) の組み込みサポートが追加され、カスタムディレクトリ実装として設計されました。この機能により、既存のLDAPディレクトリとEBX®の統合が可能になります。

詳細については、「[LDAPディレクトリ](#)」および「[組み込みLDAPディレクトリの構成](#)」を参照してください。

プリフィルタによるパフォーマンス向上

プログラムによるフィルタには、単純なチェックとより複雑なJavaコードの組み合わせを含めることができます。特定の条件下では、単純なチェックはXPath述語の形式のプリフィルタに変換できます。プリフィルタはEBX®によってクエリを最適化するために使用できます。プリフィルタは一貫性条件を満たす必要があり、パフォーマンス上の理由から特定の他の条件を満たす必要があります。関連する関数のJavaドキュメントを参照してください。

`PrefilteredAdaptationFilter` インターフェースは、`AdaptationFilter` にプリフィルタを追加するために追加されました。

`TableRefFilter` は、テーブル参照フィルタの設定中にXPathプリフィルタを定義できます。`TableRefFilterContext.setPrefilter` を参照してください。

EBX® Container Edition

Container Edition は以下のトピックで改善されました。

EBX® メタデータ管理	EBX® メタデータコンポーネントを含むイメージをビルドできるようになりました。 詳細については、「 イメージのビルド 」を参照してください。
Java SE Development Kit	JDK は LTS バージョン 21 にアップグレードされました。 詳細については、「 概要 」を参照してください。

EBX® Script IDE

本リリースでは、EBX® Script Language (ESL) で記述されたスクリプトを作成および管理できるWebベースの開発環境であるScript IDEが導入されました。このWebベースのIDEは、基本的なデータ操作、自動トリガー、さらには複雑なワークフロータスクに適用される関数やプロセスの記述など、スクリプト作成タスクを簡素化します。

Script IDEは、カスタムモジュールの記述、コンパイル、デプロイに関連するオーバーヘッドを回避するのに役立ちます。ESLと新しいScript IDEの組み合わせにより、Javaプログラミングのバックグラウンドを持たないユーザーでも、プログラムによるデータ管理タスクにアクセスしやすくなります。

Script IDEは、IDEに期待される多くの機能を提供します。

- 構文ハイライト
- コンテキストコード補完
- エラー検出
- バージョン履歴とロールバック

参照:

- スクリプト化されたRESTサービス
- スクリプト化されたRESTサービスの作成チュートリアル
- 制約検証

TIBCO EBX® ドキュメンテーション - 6.2.3

関連項目

- スクリプトIDEの概要
- スクリプトのリファレンスドキュメンテーション