

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
 - 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
 - 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での作業
 - 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

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
- 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、マスターデータの単一の管理されたバージョンを持つことが重要です。

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、役割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。

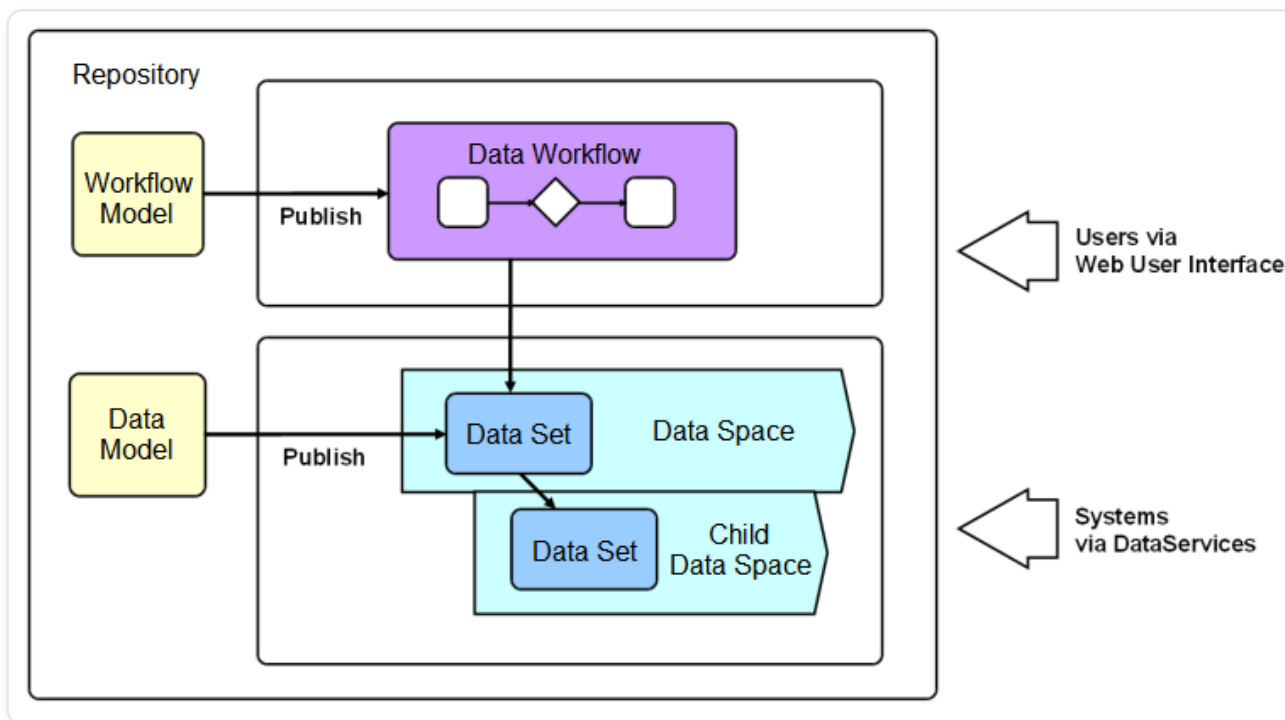
EBX®を使用したMDMプロジェクトは、データモデルの作成から始まります。ここで、マスターデータに関連するテーブル、項目、リンク、およびビジネスルールが定義されます。モデル化されたデータの例には、製品カタログ、財務階層、サプライヤーのリスト、または単純な参照テーブルが含まれます。

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格納します。データセットは、更新を相互に分離するコンテナであるデータスペース内に編成および含まれています。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で、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したり、Webサービスを介してデータスペースやワークフローを管理したり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。

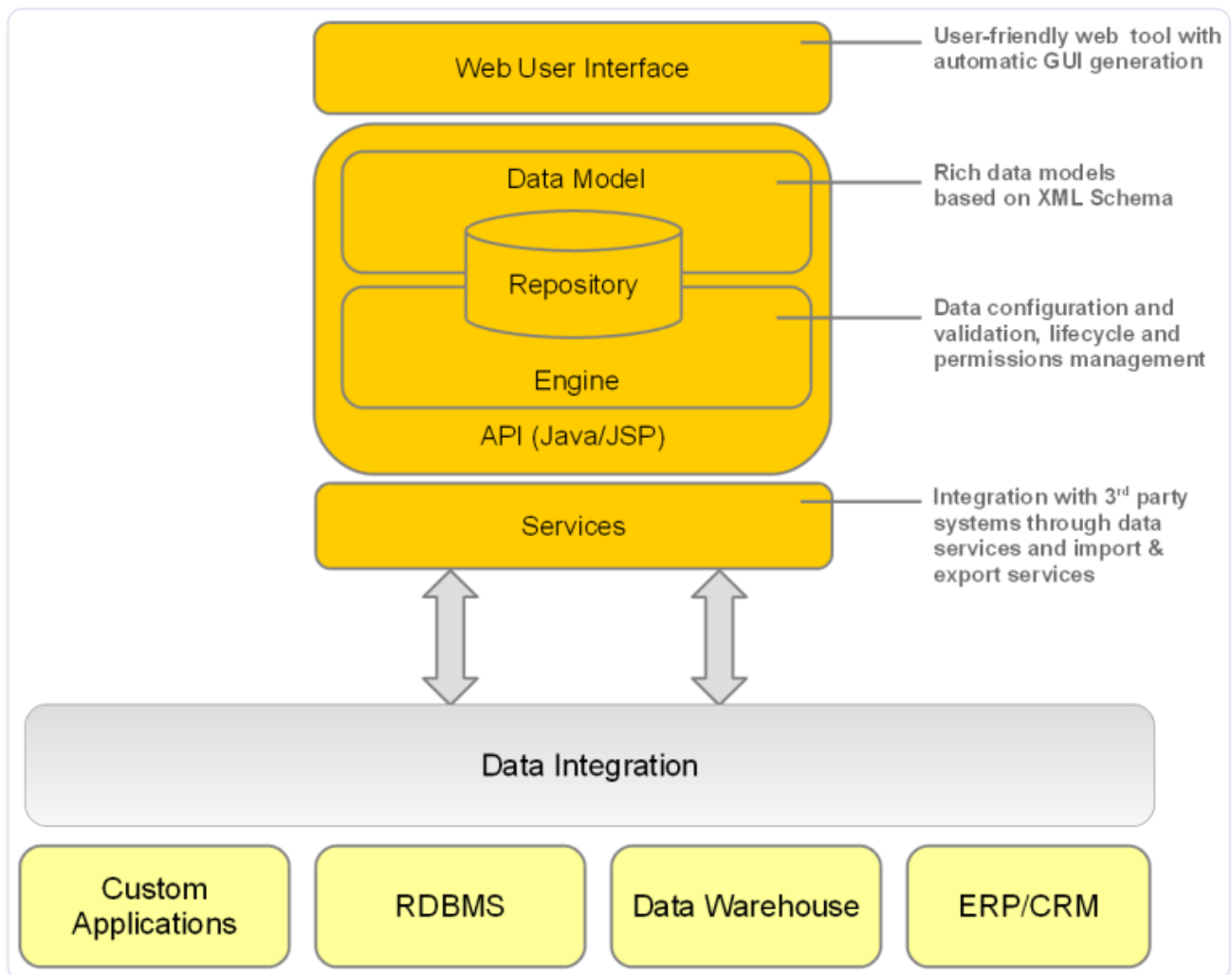


以下も参照してください。:

- 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

次のダイアグラムは、EBX®アーキテクチャを示しています。



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® ヘルプへのアクセス

- 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。

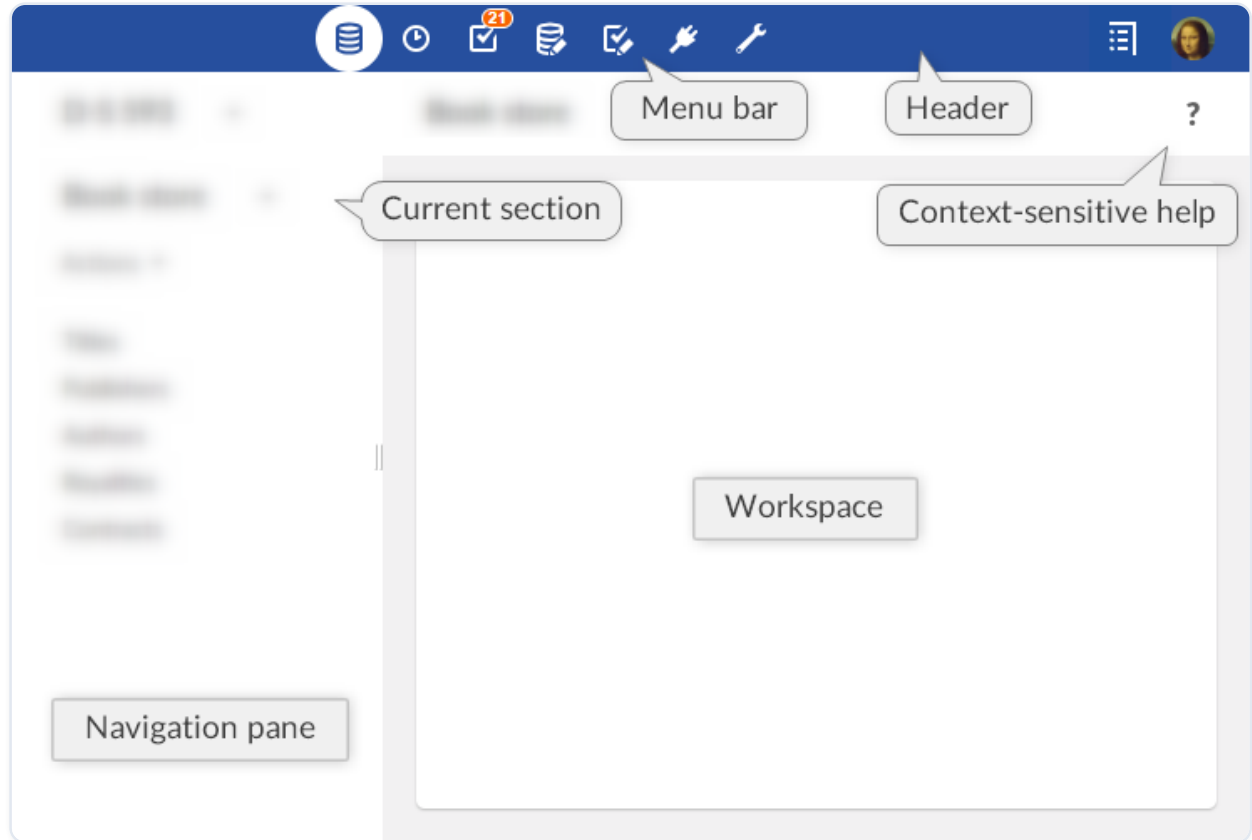
注意:

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。



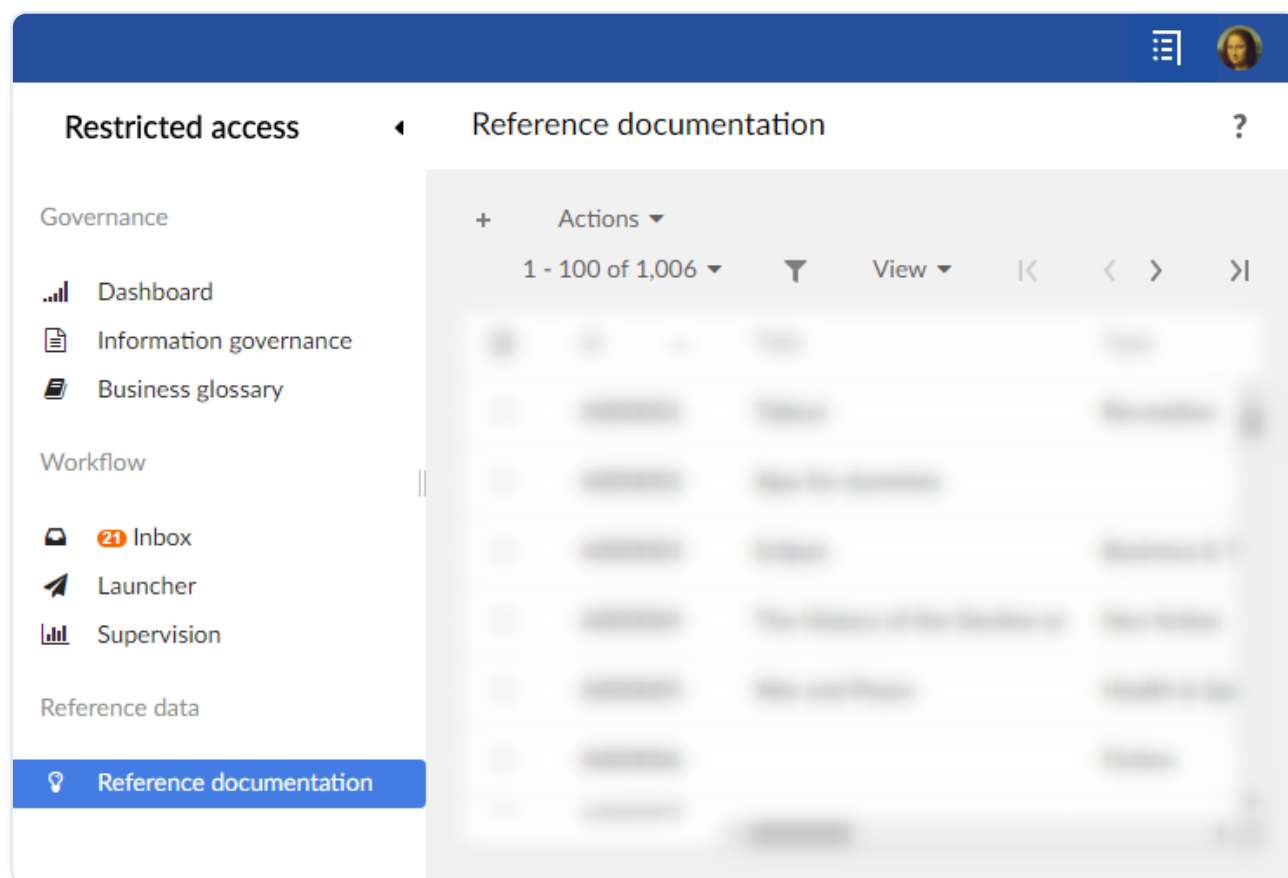
パースペクティブ

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

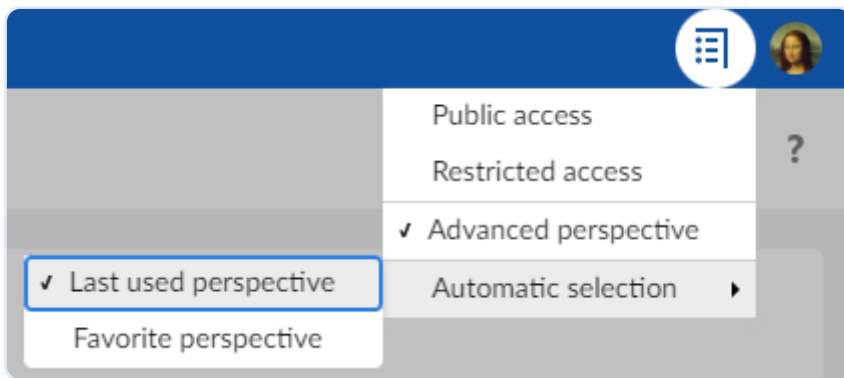
2つのパースペクティブモードから選択できるようになりました。

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。

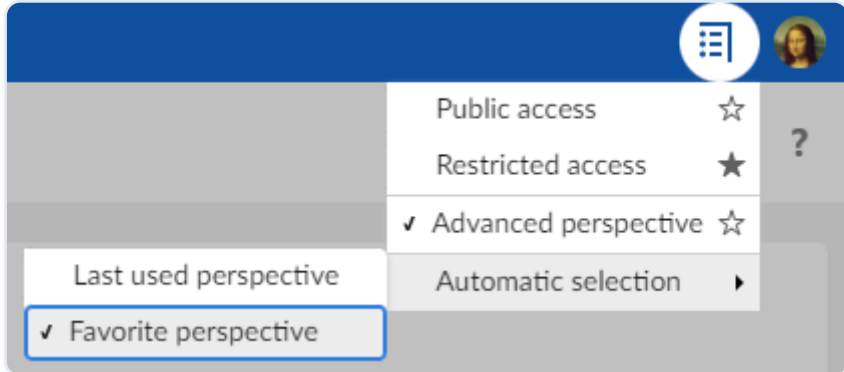
ユーザーがこのモードを選択すると、パースペクティブ名の横にある星が非表示になります。



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トア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であ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。

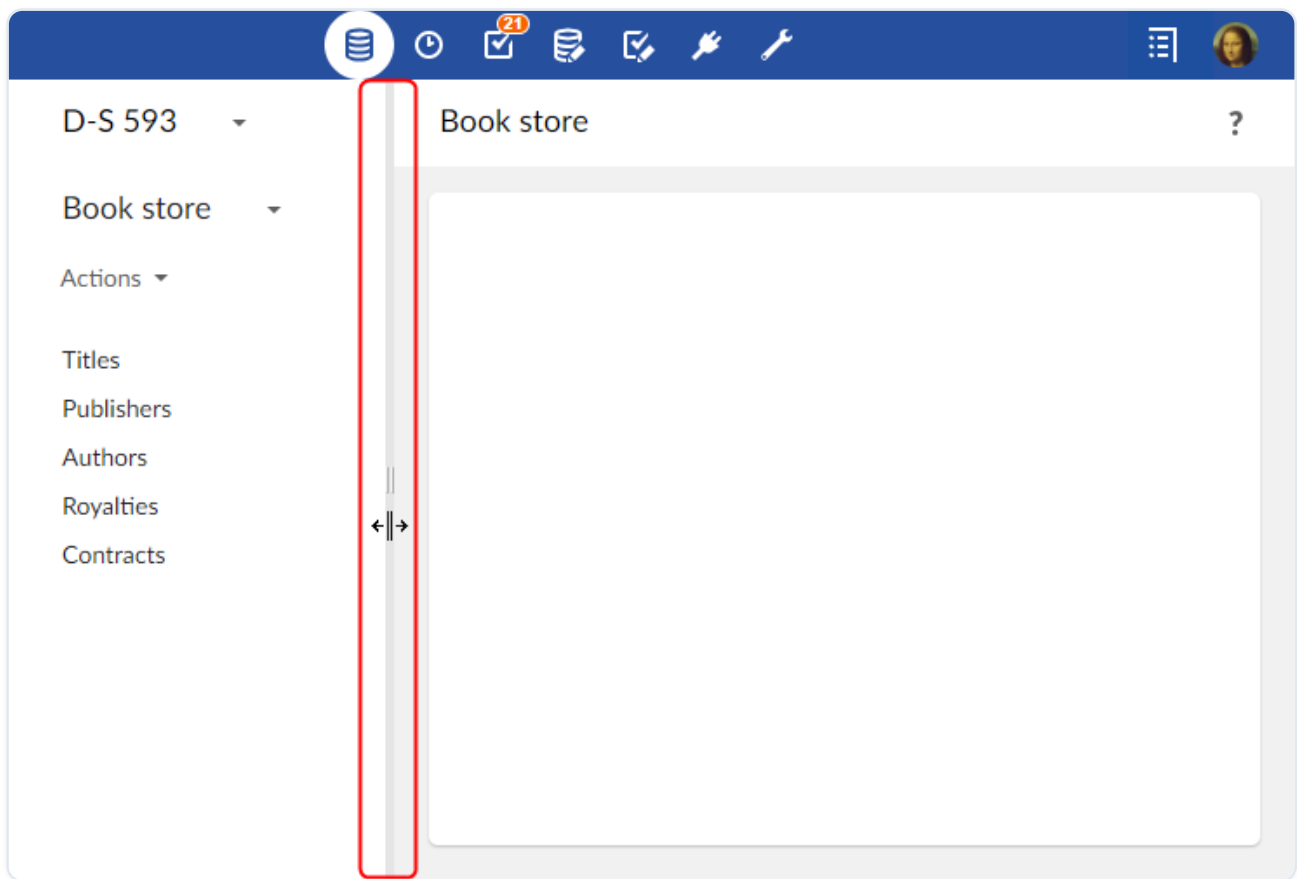
密度

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® ヘルプへのアクセス

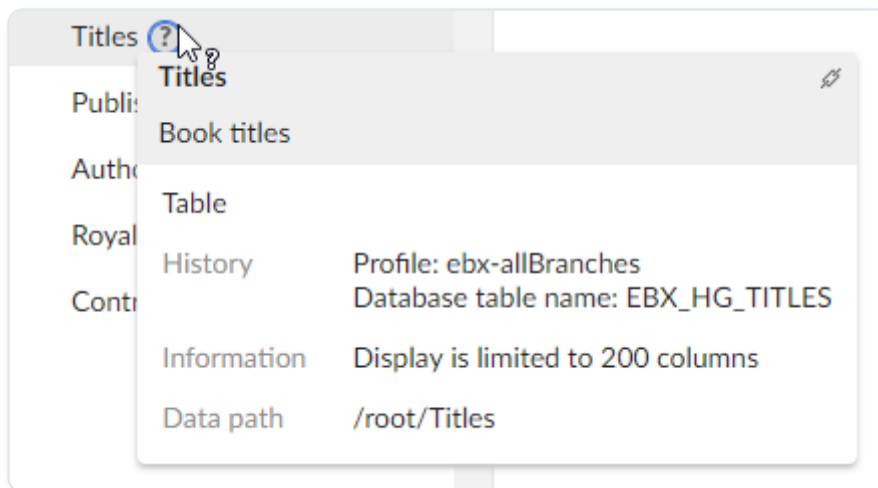
ユーザーペインからアクセスできる完全なスタンドアロン製品ドキュメントに加えて、ヘルプにはインターフェイス内のさまざまな形式でアクセスできます。

状況依存型ヘルプ

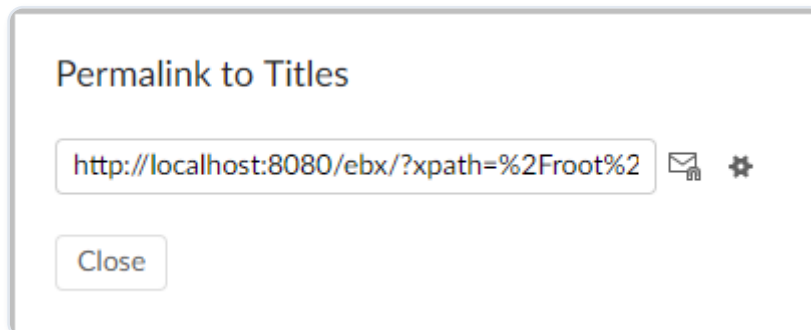
EBX® でワークスペースを参照する場合、2 番目のヘッダーの右側にある疑問符をクリックすると、状況依存のヘルプを利用できます。製品ドキュメントの対応する章が表示されます。

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。



用語集

- ガバナンス
 - リポジトリ
 - 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)
- グループ
- 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)
 - データセット
 - 継承
 - 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
 - データスペース
 - 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
 - 履歴プロファイル
 - 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
 - ノード

ガバナンス

リポジトリ

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。

プロファイル

ユーザーまたはロールの総称。プロファイルは、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](#)、[プロファイル](#)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® でロールプロファイ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](#)、[プロファイル](#)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、レコードの一意の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。

グループ

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。

データセット

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。

継承

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用す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](#)

データスペース

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。

履歴プロフィール

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。

[履歴プロフィール](#)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。

ワークフロー条件

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。

関連する概念は、[リネージの WSDL の生成](#)です。

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。

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](#)
 - [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 사용할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)
- グループ
- 再利用可能な型
- 検証ルール

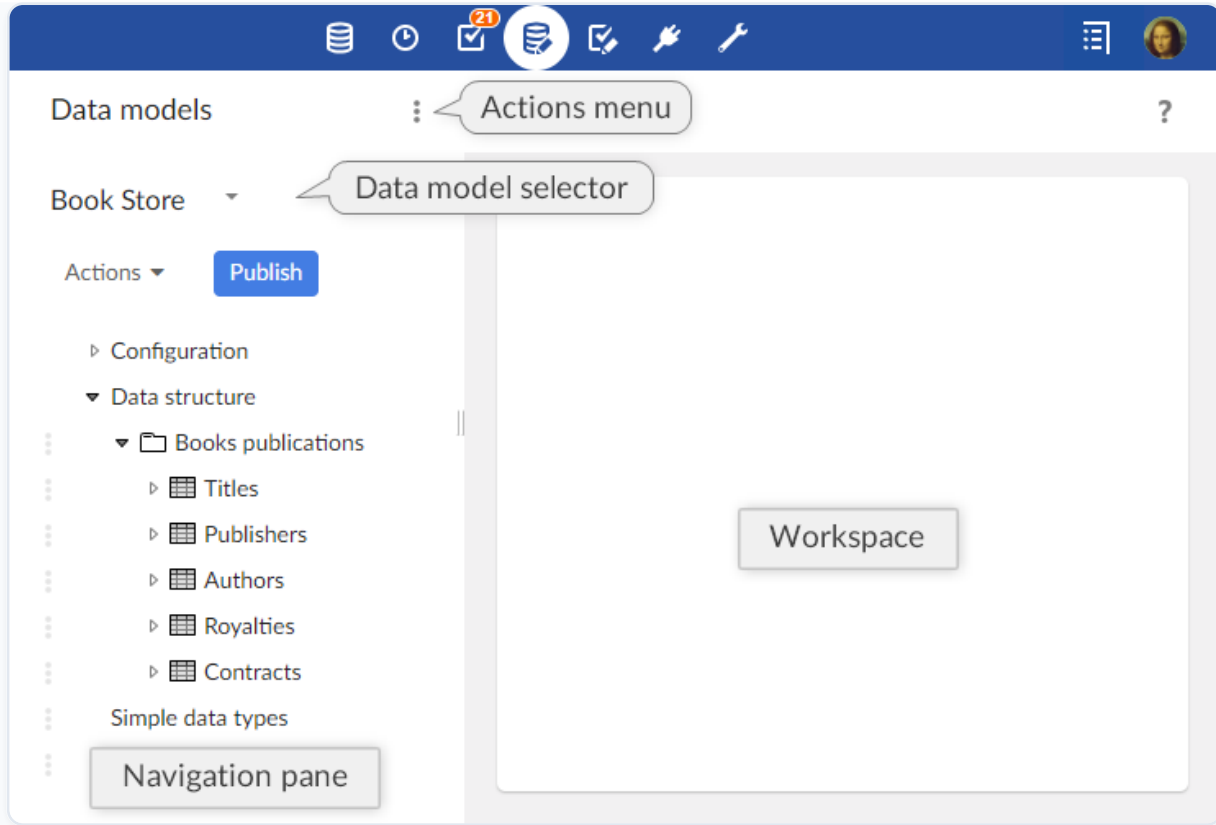
データモデル領域のユーザーインターフェイスの使用

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され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	データモデルで利用できるツールバー。
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。レプリケーション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のある 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

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

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
- 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。

データモデルの構成

- データモデルに関連する情報
- 権限
- データモデルのプロパティ
- 含まれるデータモデル
- データモデルで使用されるアドオン

データモデルに関連する情報

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、「詳細パースペクティブ」で許可されたユーザーのみが使用できます。

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され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。

ソースの場所	「コンポーネントライブラリ」でJavaコンポーネントを設定するときに使用されるソースパス。このパスが相対パスの場合、ベースパスとして「モジュールパス」を使用して解決されます。
公開モード	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公開として公開するか。埋め込みデータモデルは、公開のバージョン管理やロールバックなどの追加機能を提供します。 詳細については、公開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、要素でローカルに定義されたものよりも、関連する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。

特別な拡張機能	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許可。
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値を参照してください。
ユーザーサービスを有効にする（古いAPI）	リリース5.8.0より前に使用可能な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（少なくともこのセクションでサービスがすでに宣言されている場合を除く）。 リリース5.8.0以降、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）。 詳細については、UserServiceDeclarationを参照してください。

含まれるデータモデル

テーブルの>で他のデータモデルのエントリー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。含まれているデータモデルでは、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す。

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含まれる型定義の両方で一意である必要があります。つまり、含まれるデータ型の名前は、現在のデータモデルまたは他の含まれる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ているアドオンを指定できます。これらのアドオンには、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。

ユーザーインターフェイスを介してデータモデルで使用されるアドオンを定義するには、ナビゲーションペイン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。

名前	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	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。

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、ナビゲーションペインの下にある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める
- データモデルエレメントの作成の詳細
 - 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。

- フィールド
- グループ
- テーブル
- 主キー
- 外部キー
- 関連付け

既存のエントリー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要素の前または後に作成するかを選択できます。その後、要素作成ウィザードに従って、新しい要素を作成できます。

注意:

要素 `root` は、データモデルの作成時に常に追加されます。この要素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要素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、要素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。

要素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、および要素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要素の説明を表示します。

データモデルの要素の削除

エントリーに対応する下矢印



を使用して、任意の要素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、ネストされた要素がすべて削除されます。

既存の要素の複製

要素を複製するには、そのエントリーに対応する下矢印



をクリックします。データ構造のレベルで一意的な複製要素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、ソース要素からコピーされます。

複製された要素は、コピー元の要素と同じレベルでデータモデルに追加され、既存の要素の後に追加されます。他の要素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要素は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、そのエントリーに対応する下矢印



をクリックして、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:

エレメントをデータ構造内のレベル外の位置に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:

「単純データ型」または「複合データ型」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、「単純データ型」または「複合データ型」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、同じエレメント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、単純および複雑な再利用可能な型を定義できます。構造の定義とプロパティ。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、再利用可能なタイプ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、再利用可能なタイプのタイプ定義を共有します。

他のデータモデルで定義されたデータ型を含める

複数のデータモデル間で再利用可能なタイプ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、含まれているデータモデルのデータ型と同じ名前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型と同じ名前のデータ型を定義する外部データモデル、または含まれてい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。

含まれるデータ型は、ナビゲーションパネルの「含まれる単純なデータ型」および「含まれる複雑なデータ型」のセクションに表示されます。これらの含まれてい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、[含まれるデータモデル](#)を参照してください。

データモデルエレメントの作成の詳細

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、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。これは、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ーの下に子エレメントとして作成することにより、テーブルの主キーフィールドを作成できます。

新しいフィールドを主キーとして直接作成するだけでなく、テーブルの「詳細プロパティ」の「主キー」タブで、テーブルの既存の子フィールドをその主キーの定義に追加できます。

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ー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。タイプ「文字列」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。タイプ「文字列」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「詳細コントロール」で「外部キー制約」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ー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「関連付け」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。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。
- **リンクテーブル上。**この場合、関連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では、ソーステーブルを参照するものと、ターゲットテーブルを参照するものの2つの外部キーを定義します。また、リンクテーブルの主キーは、自動インクリメントされるフィールドや、関連のソーステーブルまたはターゲットテーブルの外部キーを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、関連付けエレメント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。

関連付けを作成したら、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、関連するレコードをフィルター処理します。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の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、既存の表形式の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「選択」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「詳細プロパティ」の「主キー」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。

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

基本的なエレメントのプロパティ

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エレメントで共有されます。

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 基準工

ディターを使用して、強力で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 `OResource` または `Password` 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、 `UIWidgetFactory` 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的である場合、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レクト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 RW に対応します。 • [読み取り専用] は、データモデル XSD のモード R- 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 CC 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 - 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 に設定します。

注意:

プロパティが 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ます。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ー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。

『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ます、すべてのプロパティにはゲッターアクセサとセッターアクセサ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

エレメントのデータ検証コントロール

エレメントの最初の作成後、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
- 簡単なコンテンツ検証
- 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使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。

Fixed length（固定長）	このフィールドに必要な正確な文字数。
Minimum length（最小の長さ）	このフィールドに許可される最小文字数。
Maximum length（最大の長さ）	このフィールドに許可される最大文字数。

Pattern（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。
Decimal places（小数位）	このフィールドに許可される小数点以下の最大桁数。
Maximum number of digits（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。
Enumeration（列挙）	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。
Greater than [constant]（[定数]より大きい）	このフィールドに許可される最小値を定義します。
Less than [constant]（[定数]より小さい）	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ているファセット](#)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	
Table (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。
Mode (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。
Referenced table (参照表)	テーブルの場所を説明するXPath式。たとえば、 <code>/root/MyTable</code> です。
Referenced dataset (参照されたデータセット)	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。
Referenced dataspace (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。
Label (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は適用されません。アクセス権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 (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。
Greater than [dynamic] ([動的]より大きい)	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。
Less than [dynamic] ([動的]より小さい)	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。
Fixed length [dynamic] (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。
Minimum length [dynamic] (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。
Maximum length [dynamic] (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。
Excluded values (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。

Excluded segment (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。 Minimum excluded value (最小除外値) : このフィールドに許可されていない最小値。 Maximum excluded value (最大除外値) : このフィールドに許可されていない最大値。
Specific constraint (component) (特定の制約 (コンポーネント))	JavaAPIの <code>Constraint</code> 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、プログラム上の制約を参照してください。
Specific enumeration (component) (特定の列挙 (コンポーネント))	列挙型を定義するJavaクラスを指定します。クラスは、JavaAPIの <code>ConstraintEnumeration</code> インターフェイスを実装することにより、値の順序付きリストを定義する必要があります。
Enumeration filled by another node (別のノードによって埋められた列挙)	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。
Dataspace set configuration (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。 Includes (含む) 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。

Pattern（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。

Type（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。

Include descendants（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。

- **Excludes (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。Includes (含む) が定義されていない場合、Excludes (除外) は無視されます。

Pattern (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。

Type (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。

Include descendants (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。フィルタは、JavaAPIの <code>DataspaceSetFilter</code> 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。
Dataset set configuration (データセット構成)	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。 - Includes (含む) 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 (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。 Include descendants (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。 - Excludes (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。Includes (含む) が定義されていない場合、Excludes (除外) は無視されます。 Pattern (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。

Include descendants（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。フィルターは、JavaAPI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。

Validation（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。
Severity（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、「Error（エラー）」、「Warning（警告）」、および「Information（情報）」です。

Error management policy（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。指定されていない場合、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ーを満たさない場合、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。このプロパティは、アーカイブをインポートする場合は無効です。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。
Message（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

データモデルの拡張

- データモデルで使用される拡張機能
- 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
- 関数
- レコード権限
- 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:

拡張機能は無効にすると、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下矢印



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、一部の拡張機能が使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、フォームに使用可能なデータモデル拡張機能が表示されます。

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。

の左側にある矢印



をクリックして、ナビゲーションペインの > セクションからビジネスオブジェクトモデルを作成します。次に、オプションを選択します。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である必要があります。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。

既存のビジネスオブジェクトの左側にある矢印



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の下に子エンティティを追加します。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。

の左側にある矢印



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。ツールバーは、次の情報を定義します。

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である必要があります。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
- セパレーター
- 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

既存のエレメントの左側にある矢印



をクリックし、メニューで作成オプションを選択して、これらのエレメントのいずれかをツールバーの下または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。
リリース	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。
リリース	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

削除するエレメントの左側にある矢印



を使用して、ツールバーのすべてのエレメントを削除できます。

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矢印



をクリックします。複製されたエレメントの名前とプロパティを指定します。ソースエレメントのすべて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、矢印



をクリックして、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矢印



をクリックし、オプションを選択します。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、テーブルプロパティを定義するテーブルまたは複合データ型を選択して、ツールバーに関連付けます。

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 **[データモデルの]** 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 **[データモデルの]** 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である必要があります。

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、ナビゲーションペイン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。一意である必要があります。
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である必要があります。
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である必要があります。

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 で設定できます。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。

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

データモデルをいつでも検証するには、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、**再検証**ボタンをクリックしてレポートされた検証ステータスを更新するか、**検証レポートの再初期化**ボタンをクリックして関連する現在の検証レポートをクリアするオプションがあります。完全な検証を最初から再実行できるようにするために、データモデルを使用します。

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**のメニューからを選択します。

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）を1つだけ含める必要があります。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ているXSDに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には、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ているモジュール。
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の構成に使用されるJavaソースへのパス。 このプロパティは、インポートされるデータモデルがプログラムエレメントを定義する場合に必要です。
モデル	インポートするモジュールのデータモデル。

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、既存のすべての公開が残り、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公開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公開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

- **公開について**
- **公開モード**
- **埋め込み公開モード**
 - **公開の表示と作成**

公開について

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。公開が存在すると、それに基づいてデータセットを作成できます。

注意:

公開ボタンは、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。

データセットは公開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公開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公開に再公開すると、その特定の公開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。

公開モード

「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」公開モードは、EBX®リポジトリ内で管理および永続化される公開を生成するため、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」公開モードは、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、使用する公開プロセスを自動的に決定します。データモデルが公開モード「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、公開プロセスは、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。

埋め込み公開モード

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公開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公開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、公開プロセス中に、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。

公開プロセスには、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公開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、公開の詳細を表示し、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公開を採用する必要があるかもしれません。複数の公開が存在する場合、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公開を選択するように求められるため、複数の公開は慎重に処理する必要があります。新しい公開を作成するアクションは、「管理者」の役割に属するユーザーのみが使用できます。

新しい公開を作成するには、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。公開に付ける名前は、リポジトリ内で一意である必要があります。

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るデータモデルスナップショットと混同しないでください。

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。

エントリーの右側にある** X**ボタンをクリックすると、バージョンを削除できます。公開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、左側が古いバージョン、右側が新しいバージョンです。

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によって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。

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

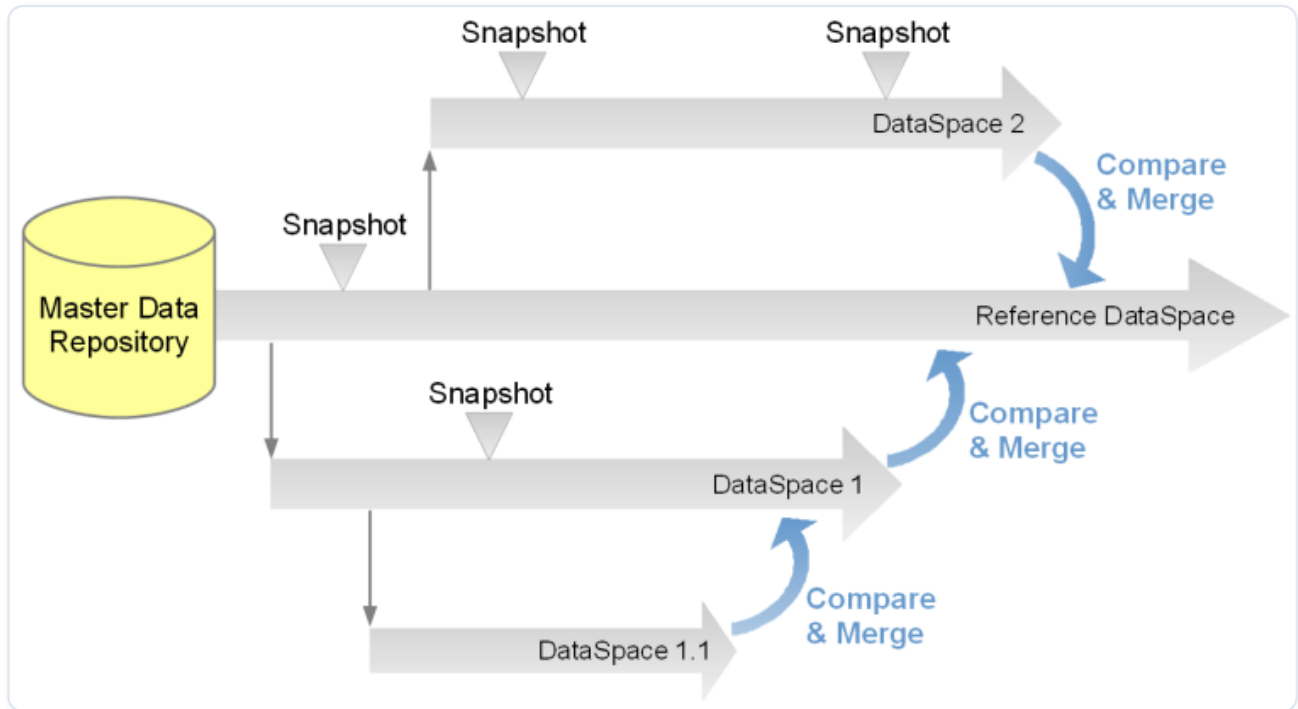
データスペースとは？

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になります。

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

スで個別に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、データスペース領域で作成、アクセス、および変更できます。

注意:

この領域は、「詳細パースペクティブ」で許可されたユーザーのみが使用できます。

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 titled 'Dataspace' with a tree structure: 'Master Data - Reference' (expanded) contains 'Samples' and 'D-S 593' (selected). Below 'D-S 593' are 'Workflow', 'Dynamic data hierarchy', and 'Dynamic data hierarchy - re'. The main area displays details for 'D-S 593'. At the top of this area are 'Create a dataspace' and 'Actions' (with a dropdown arrow). Below is a table of properties:

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

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、そのベース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**データスペースの作成**ボタンをクリックします。

注意:

この領域は、「詳細パースペクティブ」で許可されたユーザーのみが使用できます。

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。

既存のデータスペースの操作

- データスペース情報
- データスペースのアクセス許可
 - 一般的な権限
 - 許可されるアクション

- 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容：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。

	• 事前検証マージ：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。
所有者の変更	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。

データスペースのアクセス許可

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。 読み取り専用 • 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。 書き込み • 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。 非表示 • 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、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。
サービスの権利	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. 親データスペースと子データスペースの両方が、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：****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：

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、アクションメニューからデータスペースの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:

データサービスを使用して、またはプログラムでマージを実行する場合、この変更セットのレビューと受け入れ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**マージ>>**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せん。次のオプションは、ワークスペースの**アクション>進行中のマージ**メニューから使用できません。

- **キャンセル**。マージを中止し、子データスペースをマージ前の状態で回復します。
- **続行**。専用のマージデータスペースに必要な修正を加えた後、マージを再試行するために使用できます。

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**アクション>情報**を選択します。データスペースの情報ページで、**子マージポリシー**を**事前検証マージ**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**アクション>比較**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を行うには、フィルター「永続的な値のみ」を選択します。

以下も参照してください。:

- [内容の比較](#)

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**アクション>エクスポート**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	エクスポートされたアーカイブの名前。
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。「コンテンツ全体を含む更新」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、そこでマージする変更セットの違いを選択できます。

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、変更セットの差異を選択して、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データスペース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

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、データスペース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てから、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アクセス許可を変更できます。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。
所有者の変更	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を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を行うには、フィルター「永続的な値のみ」を選択します。

以下も参照してください。:

- [内容の比較](#)

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	エクスポートされたアーカイブの名前。
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、データの性質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の一部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。

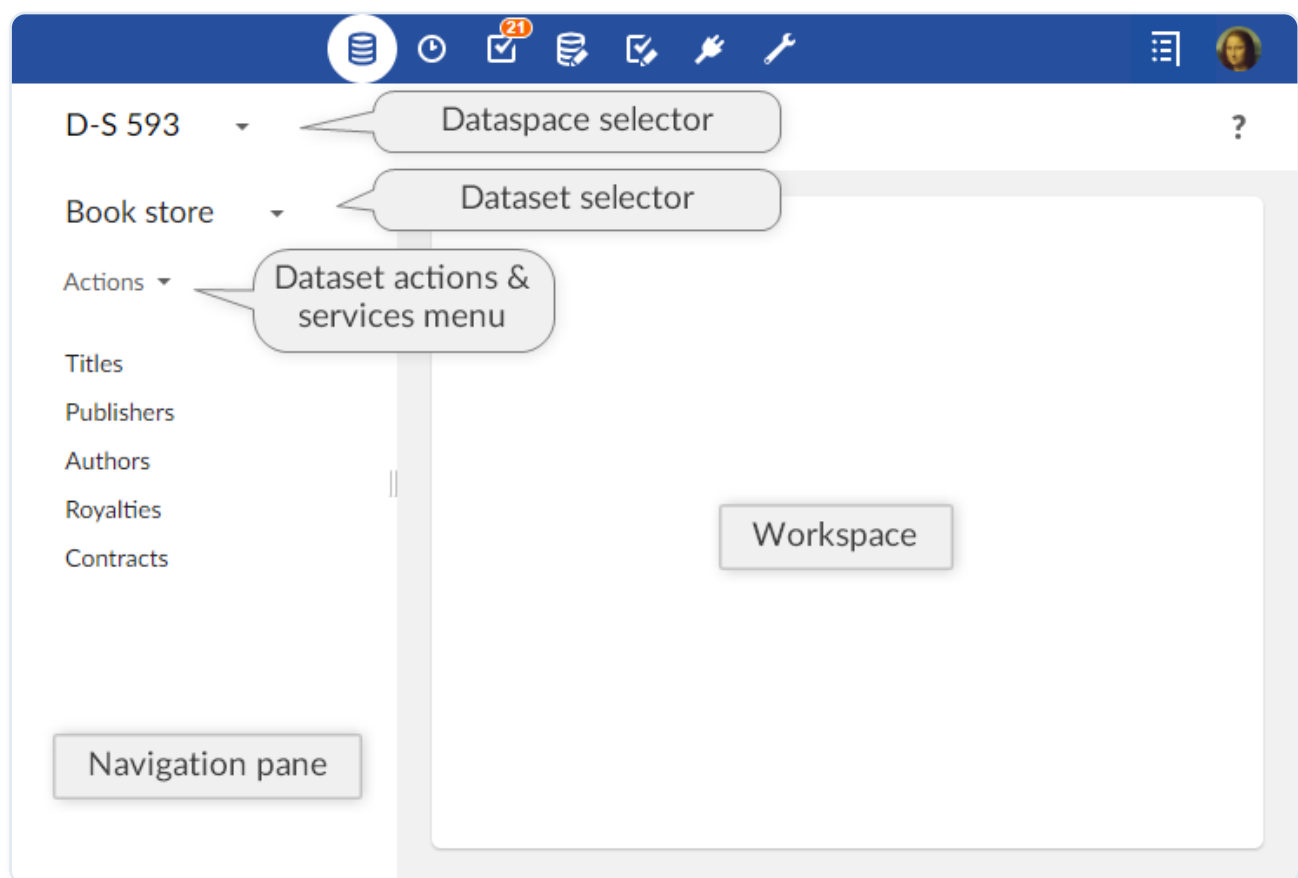
データセットに関連する基本的な概念

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。

- データスペース
- データセット
- 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）
- グループ

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、**詳細パースペクティブ**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、**データ領域**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。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すると、ボタン



に検索とフィルターが表示され、表示されるレコードを絞り込むために適用できます。

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

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

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
- 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
 - 説明の表示
 - 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
- 履歴

メニュー

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使用する [\[ビュー\] メニューツールバー](#) で直接管理されます。

ビューをグループ化することもできます。管理者は、テーブルの下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 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](#)で使用す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。

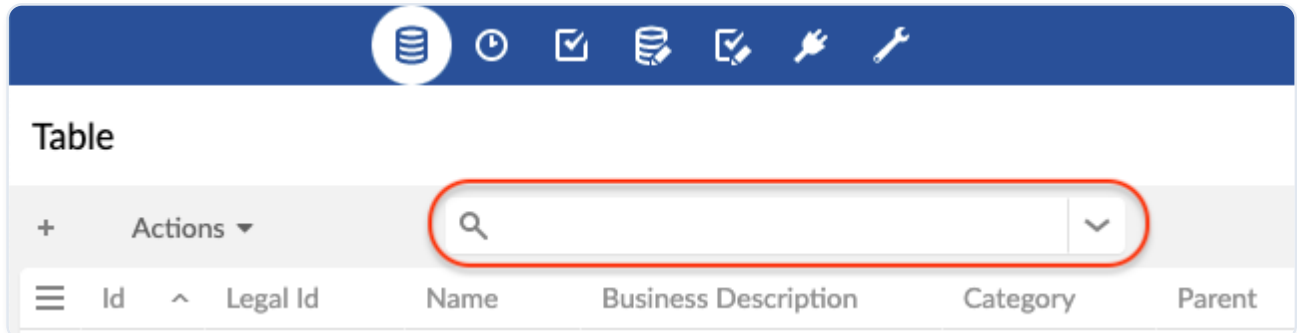
- 左側のソートされた列のリスト
- 右側のソートされていない列のリスト

← → 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、列の優先順位を変更します。

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。

	例 +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。 例 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 ?uttile

	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 <code>rachmanino*</code> <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 <code>"Johann Sebastian Bach"</code> 正確に <i>Johann Sebastian Bach</i> を含む結果を検索します。
(...) グループ	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 <code>bach +(flute piano)</code> <i>bach</i> に加えて、必須の <i>flute</i> または <i>piano</i> 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。

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/6-1-2/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 (「一致」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に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)。

単純表形式ビュー

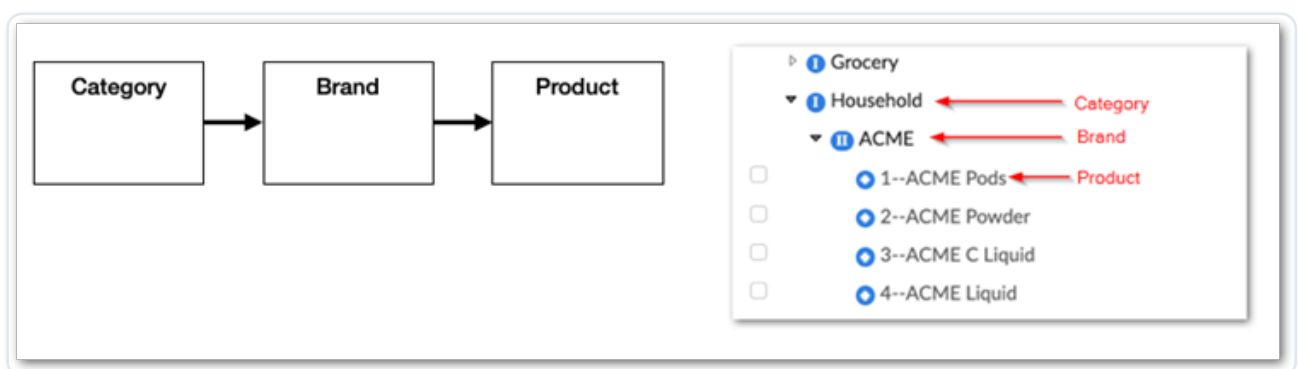
単純な表形式のビューでは、レコードをフィルタリングするための基準を定義したり、テーブルに表示される列を選択したりすることができます。

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。



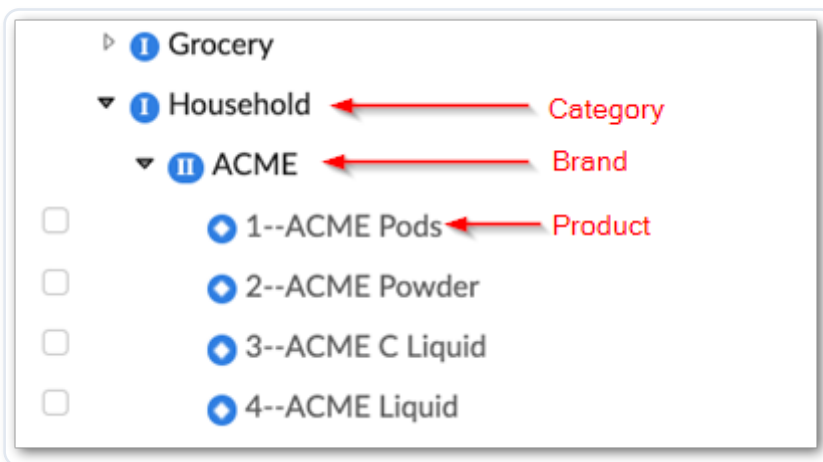
階層ディメンションとメンバー

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)

French (France)

lang

mxml

Toolbar on top of hierarchy [not defined]

Advanced properties

Remove recursive root leaves ☐ Yes ☒ No

Detect cycle ☐ Yes ☒ No

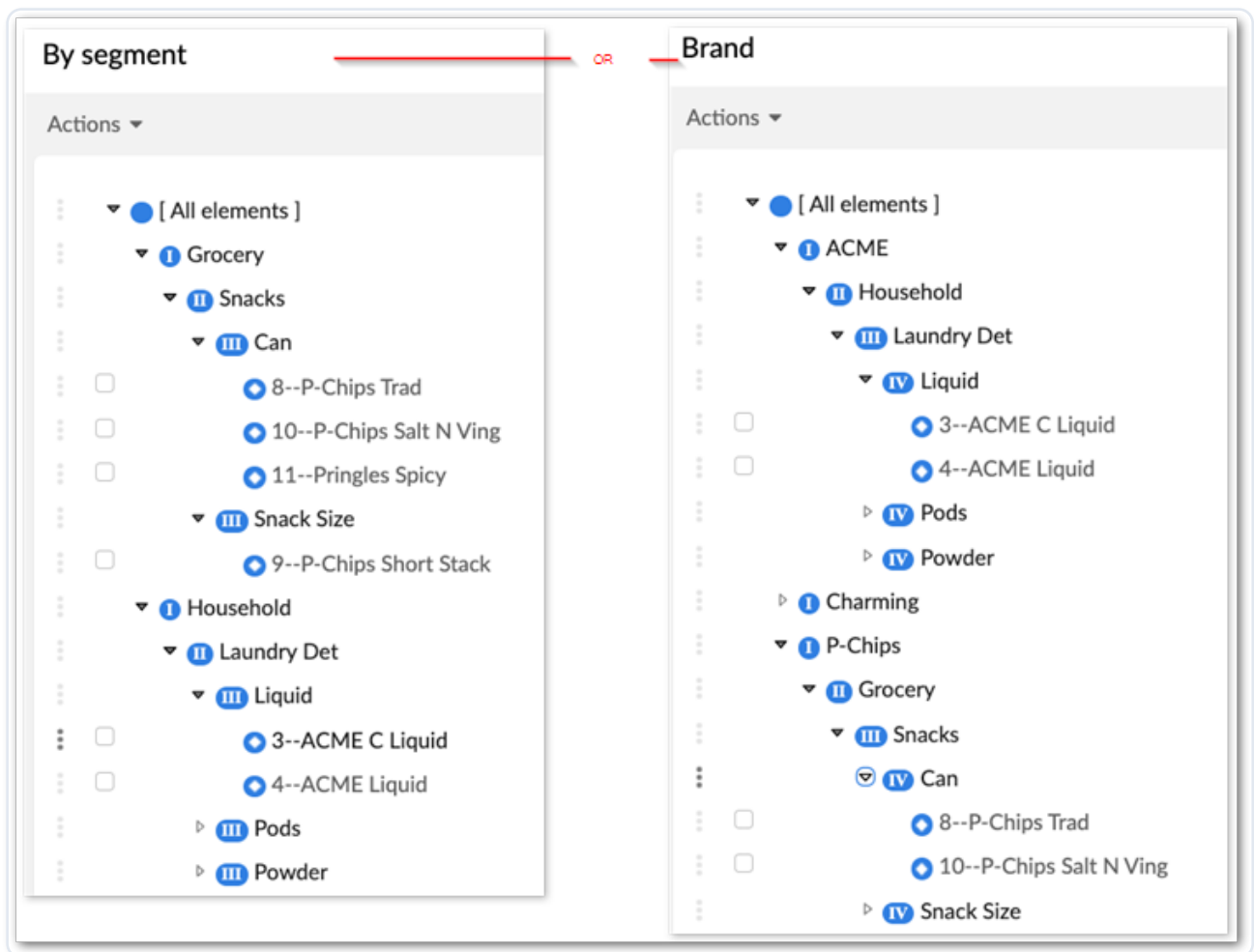
Allow skipping level ☒ Yes ☐ No

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。

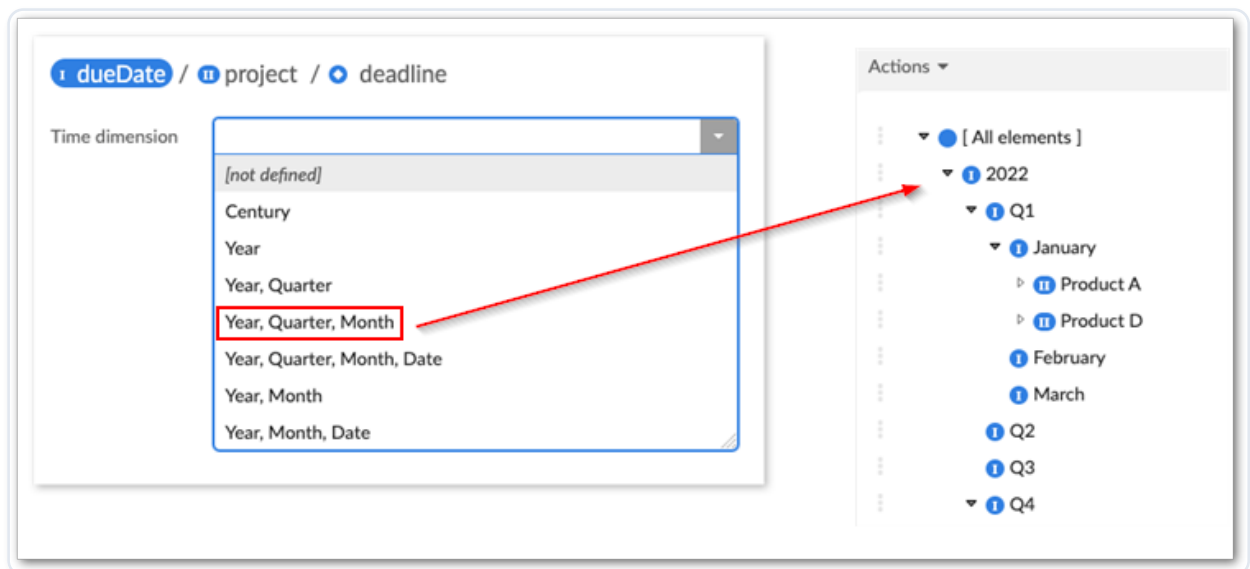


時間ベースの階層ディメンション

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。

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。

タイトルの説明 > 表示行数	行数 (3 から 6) を定義します。
タイトルの説明 > 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。
タイトルの説明 > コンテンツ	タイト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	このタイト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。
タイトルツールバー	このタイト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 権限を持つユーザーは、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 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

権限を持つユーザーは、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、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、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。

メニューツールバー

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。:

- [履歴](#)

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](#)
- [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。

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](#)

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](#)
 - [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、管理領域のユーザーインターフェイス>グラフィカルインターフェイス構成>デフォルトオプション値>インポート/エクスポートで設定できます。

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

「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JavaAPI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、JavaAPI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ません	「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。 - 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値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

ソースXMLドキュメントの状態	動作
	区別しない文字列の場合は、空の文字列。 • 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：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。 「デルタ」モードがデータサービスまたはJavaAPI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。 • <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。
エレメントは存在しますが、空です（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）のノードの場合、<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。空の文字列から。それ以外の場合、値は空の文字列に設定されます。

ソースXMLドキュメントの状態	動作
	• <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、<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「yes」の場合、nullと見なされます。このオプションが「いいえ」の場合、変更されません。

余分な列の無視

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、「管理」領域の「ユーザーインターフェイス」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。

Download file name	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XMLraw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：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。
技術データの包含	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：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	フォーマット	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

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
 - 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、管理領域のユーザーインターフェイス>グラフィカルインターフェイス構成>デフォルトオプション値>インポート/エクスポートで設定できます。

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。
ファイルエンコード	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。
継承の有効化	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：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、継承は常に無視されます。
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：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。
技術データの包含	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：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せん。

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。
ファイルエンコード	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。
継承の有効化	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：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、複数値フィールド、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では、値が `null` に設定されている文字列は空の文字列としてエクスポートされます。そのため、export-importの往復で、`null` の文字列値が空の文字列に置き換えられてしまいます。

プログラムサービスを使用して、特定の値 `ebx-csv:nil` に対して値が `null` に設定された文字列を割り当てることができます。このようにすると、往復のexport-import手続きの間に `null` 文字列の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	フォーマット	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

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、複数值グループ、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端末グループと、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

データセットをいつでも検証するには、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、**再検証**ボタンをクリックしてレポートされた検証ステータスを更新するか、**検証レポートの再初期化**ボタンをクリックして関連する現

在の検証レポートをクリアするオプションがあります。完全な検証を最初から再実行できるようにするために、データセットを使用します。

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**アクション>検証**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[[データセットの選択](#)]



メ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「アクティブ化」を「いいえ」に設定することで非アクティブ化できます。

データセットの権限の管理

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。:

- [Profile](#)

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、これらの権限が使用されることを示します。これは、定義された最も寛容な権利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。
データセットをアクティブ化/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。
オカルト継承レコード	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。ノードにカスタムアクセス許可が定義されていない場合は、デフォルトのアクセス許可が使用されます。 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（読み取り専用）」、「ReadWrite（読み取りと書き込み）」、および「Hidden（非表示）」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。
サービスの権利	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。

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、親データセットからすべてのフィールド値が継承されます。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	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	非表示にされたレコードは、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。

継承ボタン



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。非表示にされたレコードの場合は、ボタンをオンに切り替えて、継承に戻します。

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、その値は作成時に送信されたものになります。	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。オーバーライドされたレコードの個々の値は、継承に設定することも、変	上書きされたレコードを削除すると、その状態が非表示済みになります。

状態	作成	値の変更	削除
		更することもできます。	
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。	適用できない。非表示にしたレコードは変更できません。	適用できない。非表示にされた記録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](#)

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。

The screenshot displays the 'Authors' record page in the TIBCO EBX system. The left sidebar shows the 'Authors' section selected. The main content area is divided into two tabs: 'Authors' and 'EditorWithContracts'. The 'Authors' tab is active, showing a form with fields for 'Id' (00-0-006), 'Lastname' (lastNameAuthor6), 'Firstname' (firstNameAuthor6), 'Address' (addressAuthor6), 'Birth date', 'Death date', 'City' (cityAuthor6), and 'Country' (France). Below the form is a 'Books' table with columns: Id, Title, Type, Creation date, Publisher, Price, Sales. The table is empty, showing 'No records found.' On the right side of the main area, there is a vertical sidebar with three icons: a speech bubble (Comments), a star (Rating), and a tag (Tags). These icons are highlighted with a red box.

以下も参照してください。：

- [コメント](#)
- [評価](#)
- [タグ](#)

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 6.1.2 user interface. On the left, a sidebar menu shows 'Authors' selected under 'Tutorial'. The main content area is titled 'Author [lastNameAuthor1 firstNameAuthor1 [00-0-001]]'. It features a form for author details (Id, Lastname, Firstname, Address, Birth date, Death date, City, Country) and a table for 'Books' which currently shows 'No records found.'. On the right, a 'Comments' panel is open, showing three comments from 'User1 User1', 'User2 User2', and 'User3 User3'. Each comment includes a text area, a timestamp, and reaction icons (thumbs up, thumbs down, speech bubble). The bottom of the comments panel has a text input field labeled 'Write your comment' and a submit 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。

The screenshot shows the TIBCO EBX 6.1.2 interface. The main panel displays the 'Authors' section for the author 'lastNameAuthor1 firstNameAuthor1 [00-0-001]'. The 'Comments' panel on the right shows a list of comments. The comment input area at the bottom of the 'Comments' panel is highlighted with a red box, indicating the area for editing comments.

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。

The screenshot shows the TIBCO EBX 6.1.2 interface. The main panel displays the 'Authors' section for the author 'lastNameAuthor1 firstNameAuthor1 [00-0-001]'. The 'Comments' panel on the right shows a list of comments. The 'Edit' button in the comment toolbar is highlighted with a red box, indicating the area for editing comments.

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。

The screenshot shows the TIBCO EBX interface. On the left, the 'Master Data - Reference' sidebar is open, showing a list of entities under 'Authors'. The main area displays the 'Author [lastNameAuthor1 firstNameAuthor1 [00-0-001]]' record. On the right, the 'Comments' panel is open, showing two comments. The first comment, by 'User1 User1', has a red box highlighting the delete button (trash icon) in its toolbar. The second comment, by 'User3 User3', is also visible. At the bottom of the comments panel, there is a text input field labeled 'Write your comment' and a send button.

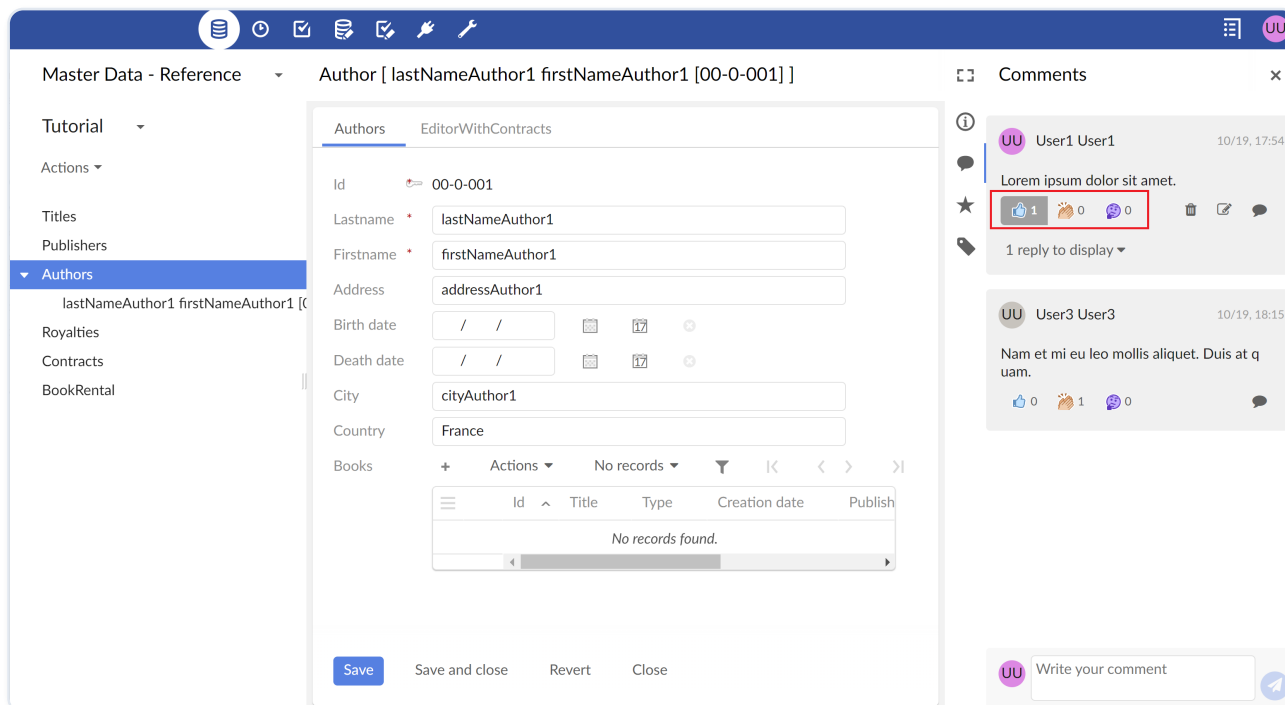
コメントへの返信

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。

This screenshot is identical to the one above, showing the TIBCO EBX interface with the 'Comments' panel open. In this instance, the red box highlights the 'Reply' button (speech bubble icon) in the toolbar of the first comment, 'User1 User1'. The rest of the interface, including the sidebar, main record, and the second comment, remains the same.

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。

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](#)

評価

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

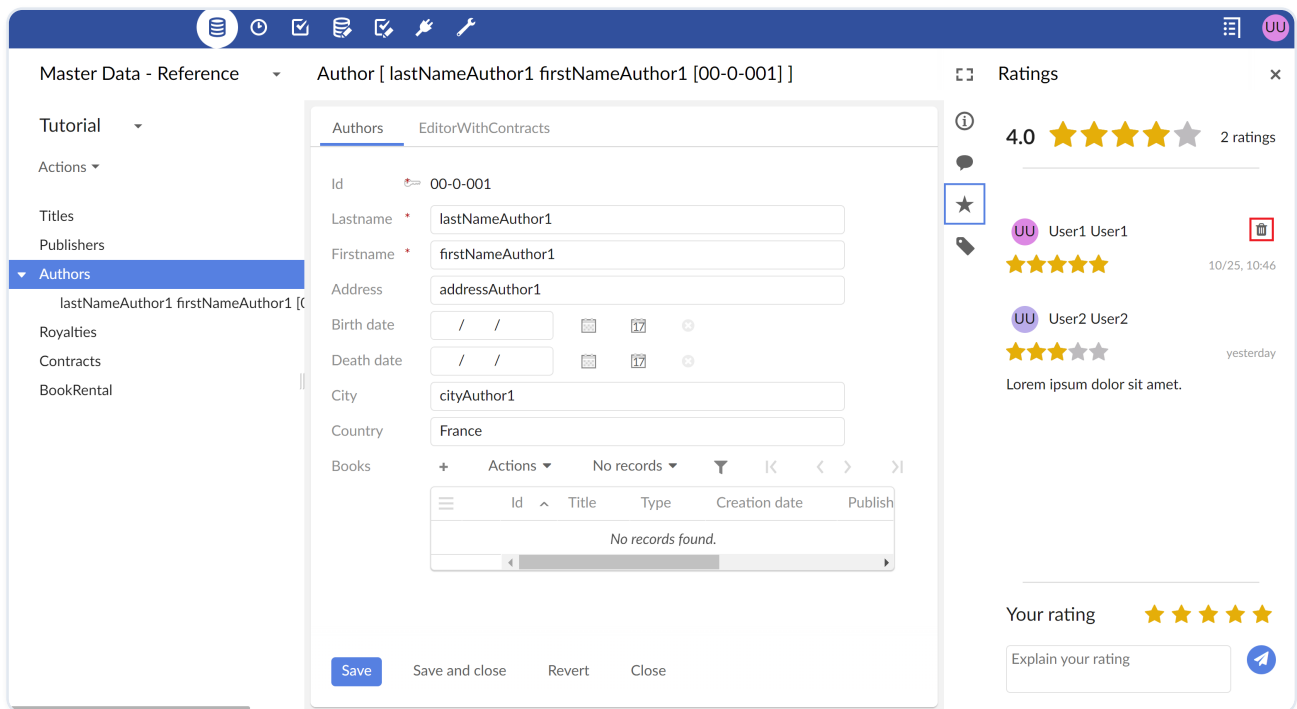
概要

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。

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](#)

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。

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

Most used click to apply

Tag B Tag C

My tags click to apply

Tag B Tag C

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

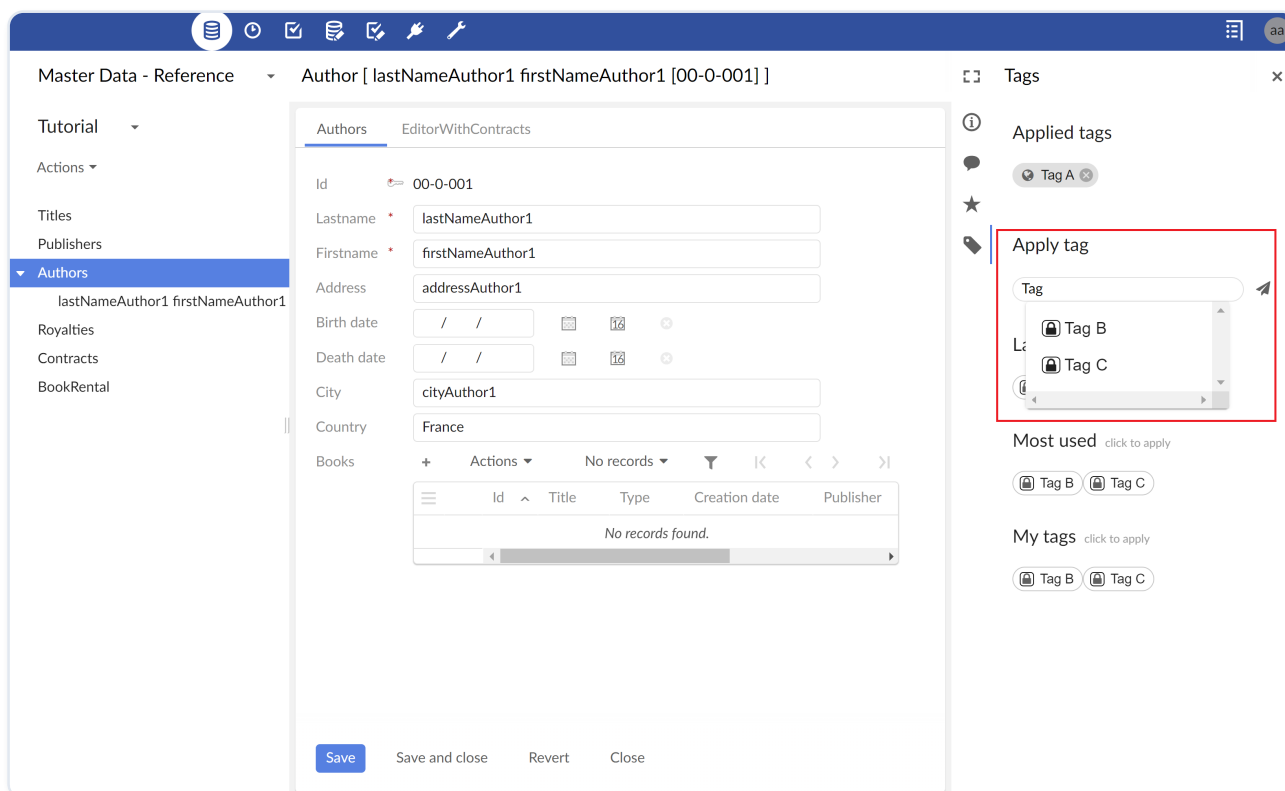
Most used click to apply

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。



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。

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 D Create public tag Create private tag

Last used

Tag B

Most used click to apply

Tag B Tag C

My tags click to apply

Tag B Tag C

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。

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

Most used click to apply

Tag B Tag C

My tags click to apply

Tag B Tag C

以下も参照してください。:

- コメント
- 評価

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
- 一般的な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公開として公開できます。次に、ワークフロー公開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。:

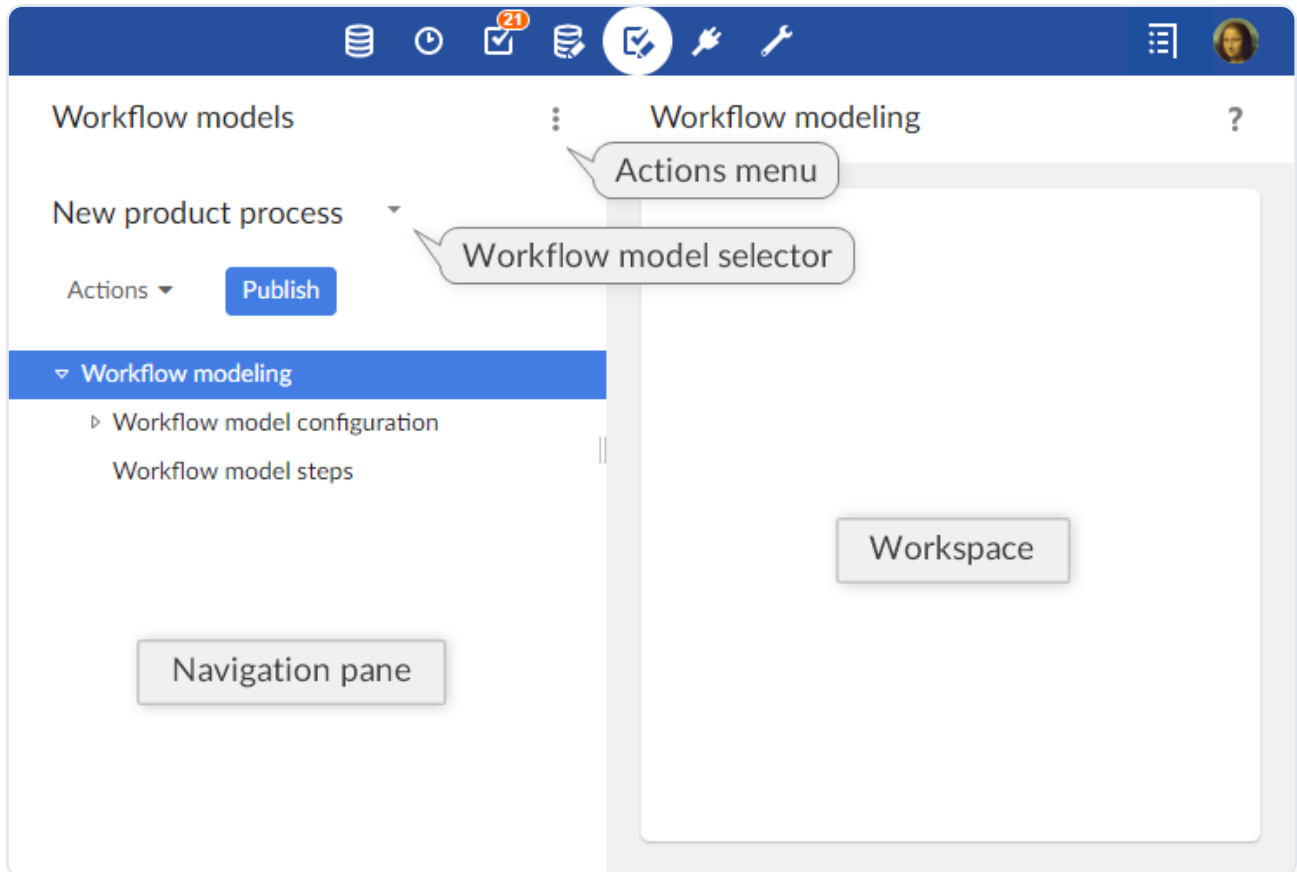
- [ワークフローモデル（用語集）](#)
- [データワークフロー（用語集）](#)

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ースの使用



注意:

この領域は、「詳細パースペクティブ」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。

一般的な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**アクションメニュー**からを選択します。

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：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：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.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役割
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 Component APIを使用して、作業項目の受信ボックス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いて、URLがワークフロー実行で電子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 Component APIを使用して、作業項目の受信ボックス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いて、URLがワークフロー実行で電子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：

本日 `${system.time}` に、新しい作業項目が提供されました。

結果の電子メール：

本日 15:19 に、新しい作業項目が提供されました。

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。
- タスク継続時間の**時間制限**。
- **ユーザープロファイル名は最大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](#)

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](#)
 - [プロファイルのリスト](#)
 - [サービス](#)
 - [構成](#)
- [スクリプトタスク](#)
 - [ライブラリスクリプトタスク](#)
 - [特定のスクリプトタスク](#)

- 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
 - 表示
 - ボタン
 - 編集
 - 機能

ワークフローモデルの作成

ワークフローモデル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、「次のステップを自動的に開く」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、作業項目の受信ボックスに戻らずに、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せん。
- 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。

詳細情報について

以下も参照してください。:

- [拡張機能](#)

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（デフォルトのサービス）
- データスペースマージビューへのアクセス
- 内容の比較
- 新しいレコードの作成
- レコードの複製
- データのCSVファイルへのエクスポート
- データのXMLファイルへのエクスポート
- CSVファイルからのデータインポート
- XMLファイルからのデータインポート
- 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

主なオプション > 拒否の有効化

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

このフィールドを「はい」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。

主なオプション > 確認リクエストの有効化

デフォルトでは、ユーザーが「承認」または「拒否」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「はい」に設定します。

主なオプション > コメントの有効化

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。

このプロパティをいいえに設定すると、この ボタンを非表示にできます。

主なオプション > 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。

主なオプション > 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、最初に応答した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、「すべての作業項目が受け入れられたとき」です。

注意:

メソッド `UserTask.handleWorkItemCompletion` をオーバーライドするサービス拡張機能を指定してユーザータスクの完了の決定を処理する場合、拡張機能の開発者は、コード内からタスク終了基準が定義されていることを確認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満たされています。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直後」

拡張

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` メソッドがオーバーライドされた場合、完了戦略は自動的にチェックされなくなりました。開発者は、このメソッド内で完了を確認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。:

- [一般的な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。

2種類のスクリプトタスクが存在し、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。

ライブラリスクリプトタスク	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に対して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。
特定のスクリプトタスク	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。

TIBCO EBX®モジュールのパッケージ化

ライブラリスクリプトタスク

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：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JavaBean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、JavaBean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。

特定のスクリプトタスク

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JavaBeanからアクセスできます。

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。

ライブラリの状態	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、<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。

ライブラリの状態

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ましたか？
- データは有効か？
- 最後のユーザータスクは受け入れられましたか？
- 値は null または空ですか？
- 値は等しいか？

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。

実装がJavaBean仕様に従っている場合、ライブラリ条件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には、JavaBean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 `条件` 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JavaBean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「フォアグラウンドサブワークフロー」は、複数のサブワークフローが起動されたときに進行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」である場合、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、受信ボックス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自動的にフォアグラウンドで考慮されます）。詳細については、進捗戦略を参照してください：[次のステップの進捗戦略](#)

2種類のサブワークフロー呼び出しが存在します。

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、JavaBean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、<code>module.xml</code> 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履歴書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、待機タスクの再開を許可されているプロファイルを指定します。待機タスクBeanを実装するJavaクラス：`WaitTaskBean`。

ワークフローデータコンテキストには、JavaBean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、生成された履歴書識別子は、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従ってデータコンテキストを更新することができます。

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把握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利用できます。PDFページの向き（横向き、縦向き）、サイズ（A3、レターサイズなど）を設定することができます。また、モデルに対してより詳細な情報を提供するための2つのプロパティがあります。 - 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。 - インデックスを追加(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したインデックスが追加されます。これは、いくつかのステップのタイトルが長すぎて、完全に表示できないような場合に便利です。 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。

表示

レイアウト>デフォルトのレイアウト	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、マウスが図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。
平面図 > 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取りします。
閉じる	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。 • ステップを複製します。 • 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 • 元に戻る
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。表示以下も参照してください
複数選択	Ctrlボタンを押しながら選択したノードまたはリンクをクリックします/選択長方形を描画します（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 メニューから 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有効化されているかどうかを指定します。公開できるようにするには、ワークフローモデルを有効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、ナビゲーションペインの 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の起動時に、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。 <i>一般的なメッセージテンプレートを参照してください。</i>
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態ですべて完了した場合にのみ送信さ

	れます。つまり、管理アクションが原因ではありません。 一般的な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。 一般的な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、作業項目の優先順位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。
クイック起動のアクティブ化	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。[クイック起動を有効化] プロパティを[はい]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。

1 番目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ボックス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：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、<code>module.xml</code> 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。

許可	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、<code>module.xml</code> 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。
プログラムに従ったアクションの許可	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 [許可] で定義されているすべてのアクセス許可を上書きします。

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。それ以外の場合、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は表示されません。

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。特定の列ごとに、ワークフローの表示時に評価されるデータコンテキスト変数を含むことができる式を関連付けることができます。

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「ワークフローの可視化」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「ワークフローの可視化」権限が自動的に付与されます。[モニタリング] > [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「ワークフローの可視化」権限が自動的に付与されます。[モニタリング] > [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。[完了済みワークフロー] 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「ワークフローの可視化」権限が自動的に付与されます。[モニタリング] > [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	ワークフロー公開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「ワークフローの可視化」権限が自動的に付与されます。[モニタリング] > [パブリケーション] セクションのボタンを使用して、公開を無効にすることができます。アクティブな公開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、公開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	ワークフロー公開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「ワークフローの可視化」権限が自動的に付与されます。[モニタリング]>[パブリケーション]セクションのボタンは、無効になっている公開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「ワークフローの可視化」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「ワークフローの可視化」権限が自動的に付与されます。[モニタリング]>[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「ワークフローの可視化」権限が自動的に付与されます。[モニタリング]>[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「ワークフローの可視化」権限が自動的に付与されます。[モニタリング] > [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公開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。

ワークフローの可視化	ワークフローの可視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公開、ワークフロー、および作業項目を [モニタリング] セクションと [完了済みワークフロー] 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可視化」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可視化権限があります。
ワークフローの可視化 > ワークフロー作成者による可視化の許可	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。
ワークフローの可視化 > 完了済みワークフローの可視化	完了したワークフローを可視化できるプロファイルを定義します。この権限により、「完了済みワークフロー」セクションで完了したワークフローを可視化し、それらの履歴にアクセスできます。「ワークフローの可視化」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を可視化する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、関連する公開はデータワークフロー領域から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公開を置き換えるように要求します。

以下も参照してください。:

- [ワークフローモデルの公開](#)

ワークフローモデルの公開

- [ワークフローの公開について](#)
- [公開とワークフローモデルのスナップショット](#)
- [公開のサブワークフロー](#)

ワークフローの公開について

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**公開**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ワークフローモデルと同じレベルのツールバーにある**一括公開**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、一括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。公開は、公開名で識別されます。

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である必要があります。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公開の名前は、ドロップダウンメニューで利用できます。公開の更新の場合、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](#)

公開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、依存するモデルごとに1つの公開を自動的に作成します。これらの技術資料は、サブワークフローを起動するための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利用できません。

複数の公開は、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既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。

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公開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。

- ワークフロー起動権限を持つユーザーとして、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公開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:

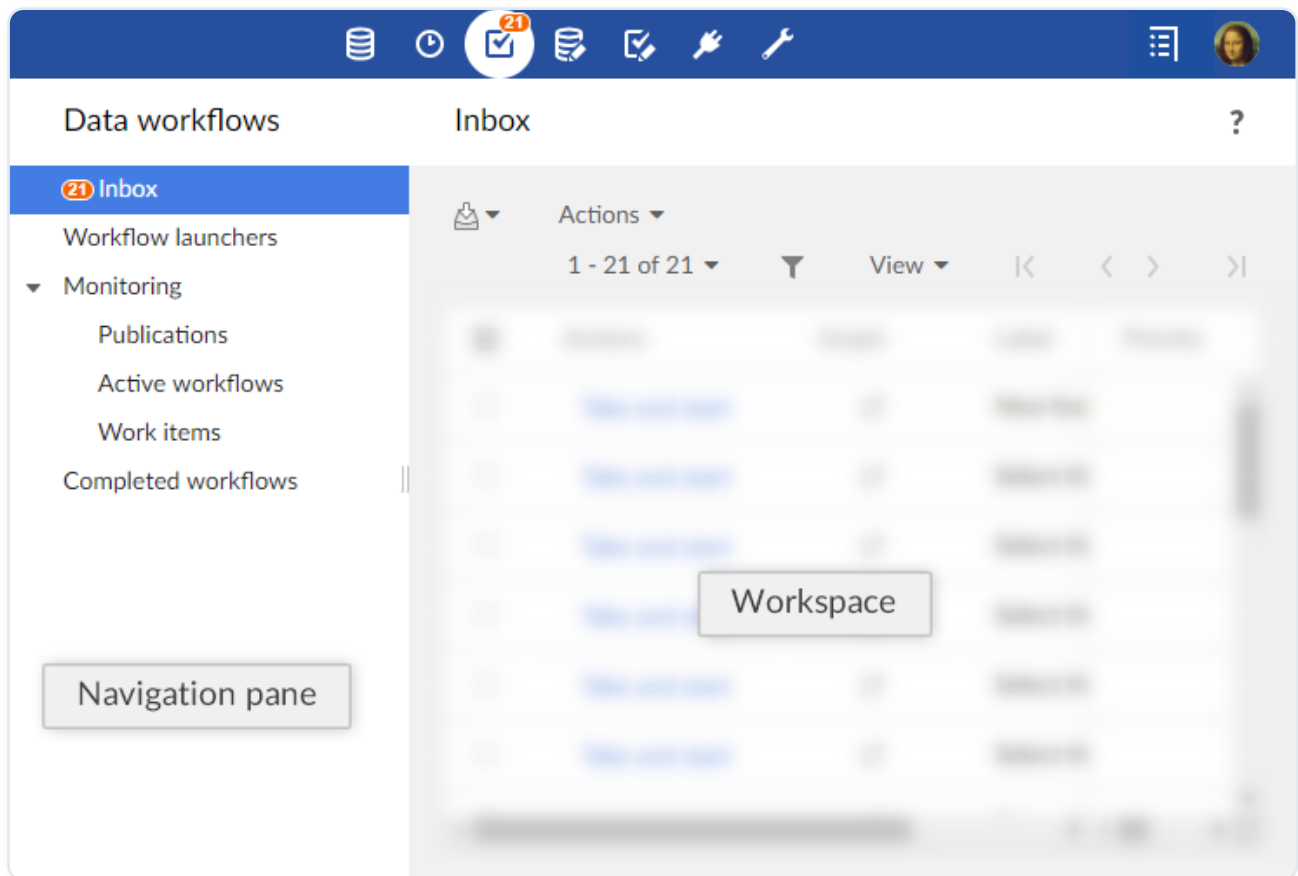
- [ワークフローモデル](#)

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で構成されています。これらのエントリは、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JavaAPI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。

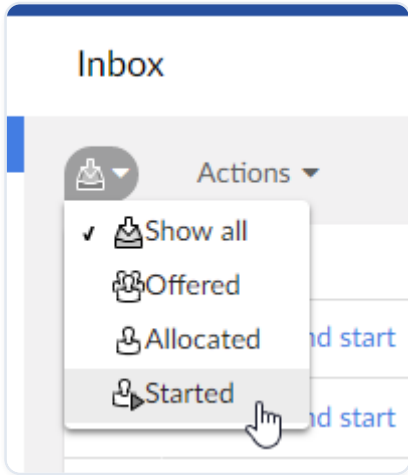
特定の列

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。式が列に対して定義されており、データコンテキスト変数が含まれている場合、これらの変数は表示時に評価されます。式にロケールに依存するビルトイン式が含まれている場合、式はデフォルトのロケールで評価されます。

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、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。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あります。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、単一の作業項目に関する主な情報は、該当する場合、ワークフローの進行状況ビューに直接表示されます。作業項目（承認または拒否）のために取得されました。

作業項目

- [作業項目について](#)

- 作業項目の状態
- 参加者として作業項目に取り組む
- 作業項目の優先順位

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、そのモデルの公開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場合があります、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場合があります、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

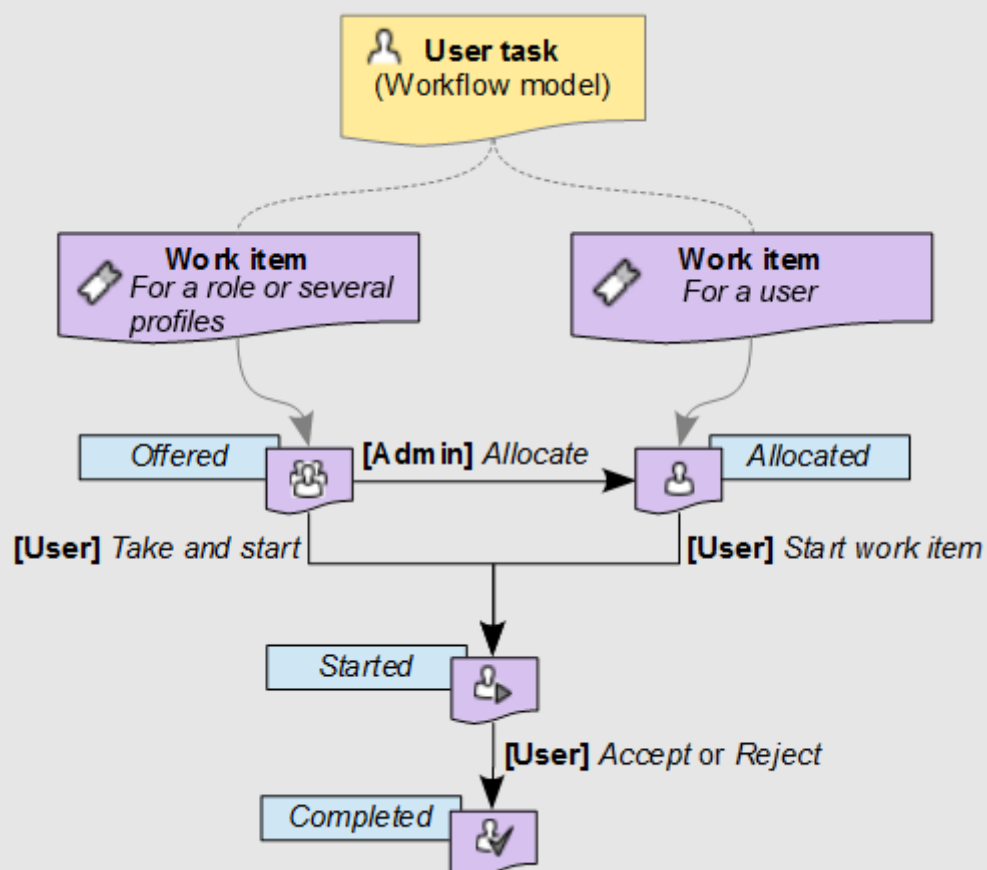
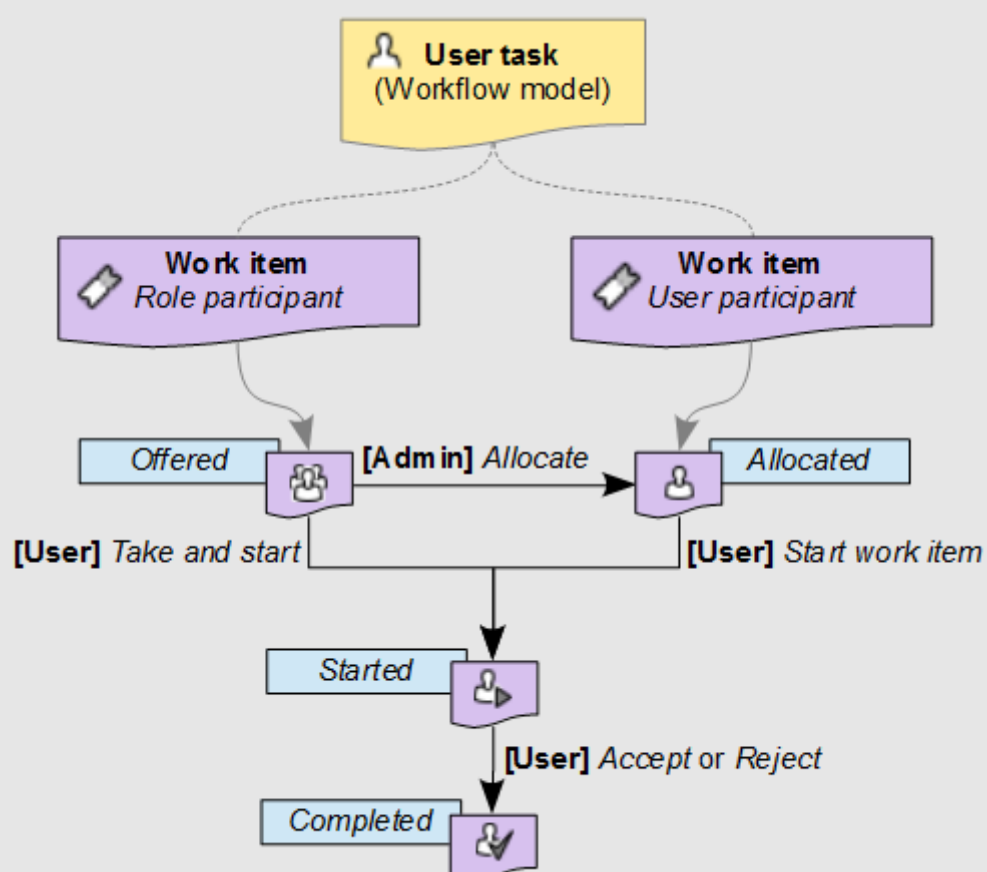
作業項目が*割り当て済み*状態の場合、定義されたユーザーは、「取得して開始」アクションを使用して、割り当てられた作業項目の作業を直接開始できます。作業項目の状態は*開始*になります。

作業項目が*提供*状態の場合、作業項目が提供されるロールのユーザーまたはメンバーは、「実行して開始」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「作業項目の開始」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。

作業項目の状態の図



参加者としての作業項目への取り組み

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、作業項目の受信ボックスに表示されます。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、**または**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。

作業項目の受信ボックス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。

作業項目の優先順位

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

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で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ー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、公開のエントリーで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ーの**アクションメニュー**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>提供状態</i> の作業項目で使用できます。
割り当て解除	<i>割り当て済み状態</i> の作業項目を <i>提供済み状態</i> にリセットします。
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>割り当て済み状態</i> 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

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、関連する公開、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ーの下に表示されます。これらの監視ビューから、テーブルエントリーの**アクションメニュー**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、公開、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
 - 公開に対する措置
 - 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マークする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。スクリプトタスクは、定義されたアクションの実行が終了すると完了します。
- 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「承認」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。

- ****実行待ち：**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：**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

公開に対するアクション

ワークフロー公開の無効化

特定のワークフロー公開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ーを選択し、無効にする公開のエントリーで>を選択します。

無効にすると、公開はユーザーの「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、進行中の既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:

公開を無効にすると、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公開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。

ワークフロー公開の非公開

ワークフローの公開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。次の手順を実行します。

1. [ワークフロー公開の無効化](#)で説明されているように、ワークフロー公開を無効にして、ユーザーがワークフロー公開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公開のエントリーから** > **を選択して、ワークフロー公開を非公開にします。

注意:

ワークフロー公開の非公開を選択すると、このワークフロー公開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**アクション**メニューからアクションを実行できます。

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ーから** > **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ーから** > **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ーから** > **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ーから > 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、「イベントを待機中」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ンアップ

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に表示できます。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、「完了したワークフロー」テーブルのワークフローのエントリーから** > ** 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、その履歴は技術管理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](#)

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、「データサービス」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す。

以下も参照してください。:

- WSDL/SOAP
- REST

リネージ

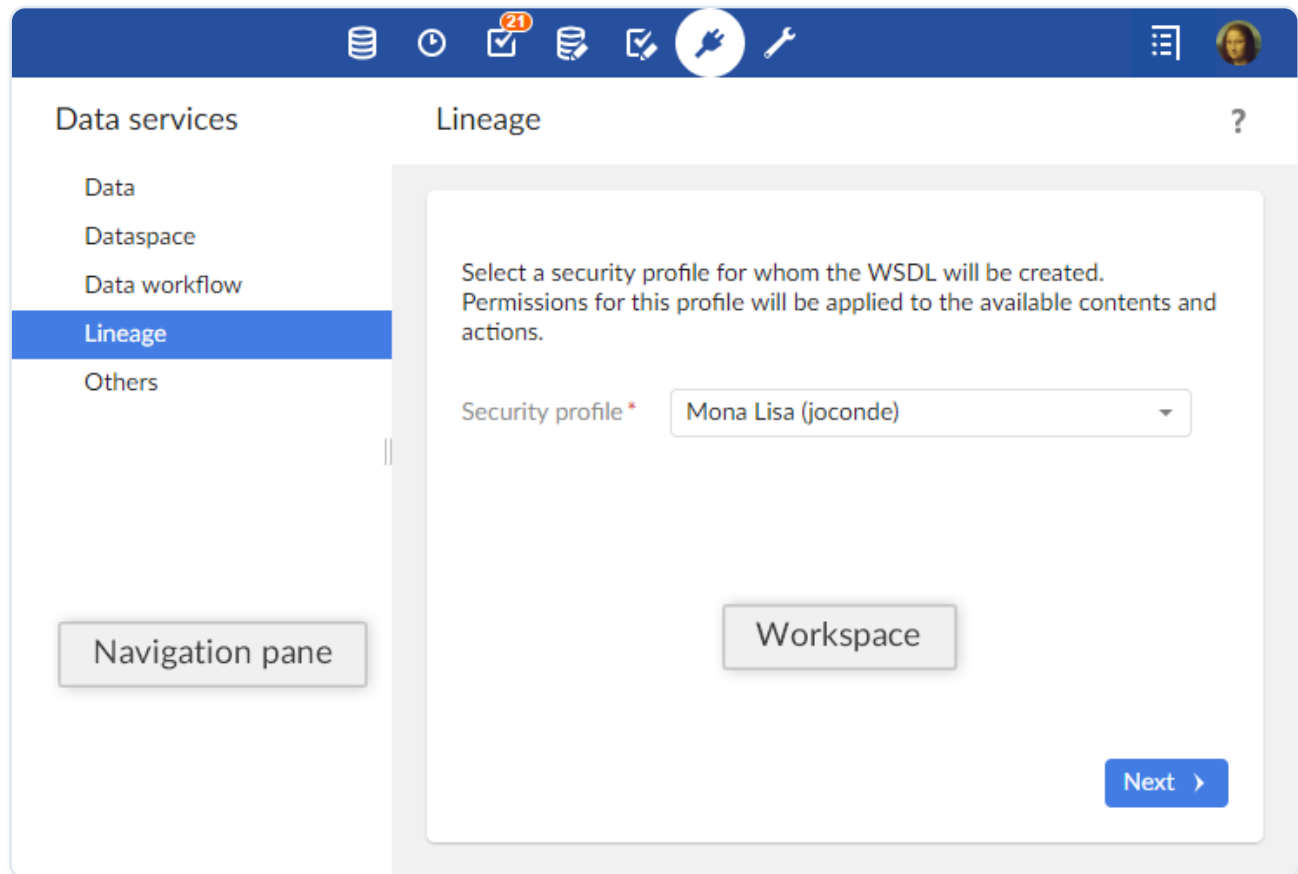
リネージは、人間以外のユーザー、つまりデータサービスのユーザー権限プロファイルを確立するために使用されます。WSDLインターフェイスを使用してデータにアクセスする場合、データサービスはリネージを通じて確立されたアクセス許可プロファイルを使用します。

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、「詳細パースペクティブ」で許可されたユーザーのみが使用できます。

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、**データサービス領域**のナビゲーションパネルでを選択し、ウィザードの手順を実行します。

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

以下の操作は、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の操作の実行

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、**データサービス領域**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の検証
- 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作用の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、**データサービス**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ではないため、追加情報は必要ありません。

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、**データサービス**領域からを選択します。これは、**管理**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いています。

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. **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。

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利用できます。

次のWSDLを生成するには:

- ユーザーインターフェイスの管理
- システム情報の取得

データサービス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
- システム情報を取得する

以下も参照してください。:

- HTTPプロトコルからのWSDL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、データサービス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ダウンロード](#)
- [ディレクトリサービス](#)

参考マニュアル

- 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
 - ラベリングとローカリゼーション
 - TIBCO EBX®の国際化の拡張
- 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
 - パフォーマンスガイドライン

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て、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を介し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
- 代表的な状態転送（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

プログラマティックサービス

プログラマティック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プロシージャを実行できます。

プログラマティックサービスの例は次のとおりです。

- 外部ソースからデータをインポートする。
- データを複数のシステムへエクスポートする。
- 監視システムによって開始されたデータを履歴化する。
- データの最適化とリファクタリングを行う。

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

- 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、要求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。:

- [範囲](#)

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「フル」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 «受信ボックス» 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

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` の配備記述子 `/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>ディレクトリ</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閉じる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 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である必要があります。	<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。この公開名は、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 %5F 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、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=record</code> <code>firstCallDisplay=record:{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.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。 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</code>	いいえ

パラメーター	説明	必須
	<code>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、階層、またはタイルビュー (テーブルレベル) で表示す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	いいえ

パラメーター	説明	必須
	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。	
<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	いいえ

パラメーター	説明	必須
	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</pre> <pre>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。	
<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	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。

パラメーター	説明	必須
	<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。	

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。

データセット「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
- 内容の比較
 - 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>	開始時に承認を無効化	デフォルトでは、このサービスに関連付けられた対話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SessionInteraction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。パースペクティブはこのパラメーターを使用しません。
<code>firstCallDisplay</code>	ファーストコール表示モード	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（値= 'auto'）：選択に応じて表示が自動的に設定されます。ビュー（値= 'view'）：表形式のビューまたは階層ビューの表示を強制します。レコード（値= 'record'）：述部に少なくとも1つのレ

パラメーター	ラベル	説明
		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である必要があります。
<code>scope</code>	範囲	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「履歴」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、その他の目的にも使用できます。
<code>version</code>	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。
<code>viewPublication</code>	ビュー	表示するビューの公開名。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。
<code>xpath</code>	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である必要があります。表記は、簡略化された構文で、簡略化

パラメーター	ラベル	説明
		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
<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である必要があります-このフィールドは、このサービスに必要です。
<code>scope</code>	範囲	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
trackingInfo	追跡情報	追跡情報は「履歴」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、その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位置パスである必要があります。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
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。

パラメーター	ラベル	説明
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である必要があります-このフィールドは、このサービスに必要です。
scope	範囲	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「履歴」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、その他の目的にも使用できます。
xpath	複製するレコード (XPath)	値は、既存のレコードの有効な絶対ロケーションパスである必要があります。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。

データのXMLファイルへのエクスポート

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である必要があります-このフィールドは、このサービスに必要です。
<code>scope</code>	範囲	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
trackingInfo	追跡情報	追跡情報は「履歴」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、その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位置パスである必要があります。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。

データのCSVファイルへのエクスポート

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である必要があります-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	範囲	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「履歴」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、その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位置パスである必要があります。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。

XMLファイルからのデータインポート

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しません。

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
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である必要があります-このフィールドは、このサービスに必要です。
scope	範囲	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「履歴」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、その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位置パスである必要があります。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。

CSVファイルからのデータインポート

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します。

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
<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である必要があります-このフィールドは、このサービスに必要です。
<code>scope</code>	範囲	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「履歴」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、その他の目的にも使用できます。
<code>xpath</code>	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位置パスである必要があります。

パラメーター	ラベル	説明
		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
<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>	範囲	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「履歴」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、その他の目的にも使用できます。

パラメーター	ラベル	説明
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である必要があります。
scope	範囲	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「履歴」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

パラメーター	ラベル	説明
		報など、その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	見つけられたエラー	検証でエラーが発生した場合は、「true」が含まれます。
hasFatal	見つけられた致命的エラー	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	見つけられた情報	検証によって情報が生成された場合は、「true」が含まれます。
hasWarning	見つけられた警告	検証によって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。

パラメーター	ラベル	説明
scope	範囲	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「履歴」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、その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
mergeResult	マージの成功	マージが成功した場合は「true」を含み、それ以外の場合は「false」を含みます。
mergeState	マージ状態	マージの戻りコードが含まれています。 InteractionMergeStateUIHttpManagerComponent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
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	範囲	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「履歴」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、その他の目的にも使用できます。

内容の比較

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
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、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

パラメーター	ラベル	説明
		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、比較では解決モードが使用されます。
<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である必要があります。
<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である必要があります。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。

パラメーター	ラベル	説明
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である必要があります。
scope	範囲	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「履歴」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、その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と、比較する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である必要があります。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
<code>scope</code>	範囲	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。
<code>viewPublication</code>	ビューの公開	このサービスに適用するビューの公開名を定義します。
<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 (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 パス言語 (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>../../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() およびアトミック式。
<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>

式	形式	注意/例
<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>	
<code><boolean comparator></code>	<code>=</code> または <code>!=</code>	
<code><numeric comparator></code>	<code>=</code> , <code>!=</code> , <code><</code> , <code>></code> , <code><=</code> , または <code>>=</code>	
<code><string comparator></code>	<code>=</code> または <code>!=</code>	
<code><method></code>	<code><date method></code> <code><string method></code> <code>osd:is-null</code> メソッドまたは <code>osd:is-not-null</code> メソッド	
<code><date, time & dateTime method></code>	<code>date-less-than</code> <code>date-equal</code> または <code>date-greater-than</code>	
<code><string method></code>	<code>contains</code> <code>osd:contains-case-insensitive</code>	

式	形式	注意/例
	<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> <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><criteriaion></code>	<code><boolean criteriaion></code> <code><date criteriaion></code> <code><dateTime criteriaion></code> <code><numeric criteriaion></code> <code><string criteriaion></code> または <code><time criteriaion></code>	
<code><boolean criteriaion></code>	<code>true</code> , <code>false</code>	
<code><numeric criteriaion></code>	整数または小数	<code>-4.6</code>

式	形式	注意/例
<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' として引用およびフォーマット設定	<code>'2007-12-31T11:55:00'</code>

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) はノードパスである必要があります。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。基準値は単一の値である必要があります。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。

書式設定ポリシーは、**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` にあるファイルで構成されます。たとえば、ギリシャ語（`e1`）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/e1/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推奨は、区切り文字としてハイフン「-」を使用する[IETF BCP 47](#)推奨に従います。
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®の国際化の拡張](#)

TIBCO EBX® ローカライゼーションの拡張

- [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
 - 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-lp_<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 + "\_ + 言語 + "\_ + 国 + "\_ + variant + ".mxml"`
- `resourceName + "\_ + 言語 + "\_ + 国 + ".mxml"`
- `resourceName + "\_ + 言語 + ".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

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](#)
- [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

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、テーブルが基盤と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。

マップされた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

基盤となるデータベースに直接永続化する性質があるため、マップモードで格納されているすべてのテーブルにいくつかの制限が適用されます。

- [サポートされているデータベースの制限](#)
- 無制限の長さの文字列: タイプ `xs:string`、その派生型、および `xs:anyURI` の外部キーを除く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、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ているため、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、これらの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、文字タイプの最大長に関する基盤となるデータベースの制限が適用されます。
- タイプ `type="osd:password"` のフィールドは無視されます。
- ターミナルコンプレックスタイプがサポートされています。ただし、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、基盤となるRDBMSの制限の対象となります。たとえば、テーブルの最大列数が適用されます（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ているフィールドの2倍のフィールドが含まれていることに注意してください（1つの機能フィールドと、オペコード用に生成された1つのフィールド）。

データモデルの進化は、既存のデータモデルによっては、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

- [概要](#)

- 履歴の構成
 - リポジトリでの履歴の構成
 - データモデルでの履歴の構成
 - 完全性
- 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
- 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、および削除) を追跡できる機能です。

これは、非推奨の [XML 監査証跡](#) を改善したものです。

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

テーブルの履歴をアクティブにするには、データモデル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らがテーブルに関連付けられる方法について説明します。

リポジトリでの履歴の構成

履歴プロファイルは、履歴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、**管理 > 履歴とログ**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。
- 履歴がアクティブ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考慮する必要があるかどうかを指定することができます。

一部のプロファイルは、リポジトリのインストール時に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します。ただし、内部データモデルはデータセットノードのみを定義するため、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

テーブル履歴のアクティブ化

履歴は、データモデルアシスタントを介して、または基になるデータモデルを編集することにより、テーブル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アクティブにするには、`historyProfile` エlementを使用してテーブルで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プロファイルを表示できます。

履歴は、テーブルごとに個別にアクティブ化する必要があります。詳細については、[モデル設計のドキュメント](#)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が履歴になります ([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>

```

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。

完全性

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デプロイするには、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、データ権限と読み取り専用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。

「トランザクション履歴」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、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。

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 <code>HV_TX</code> 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。

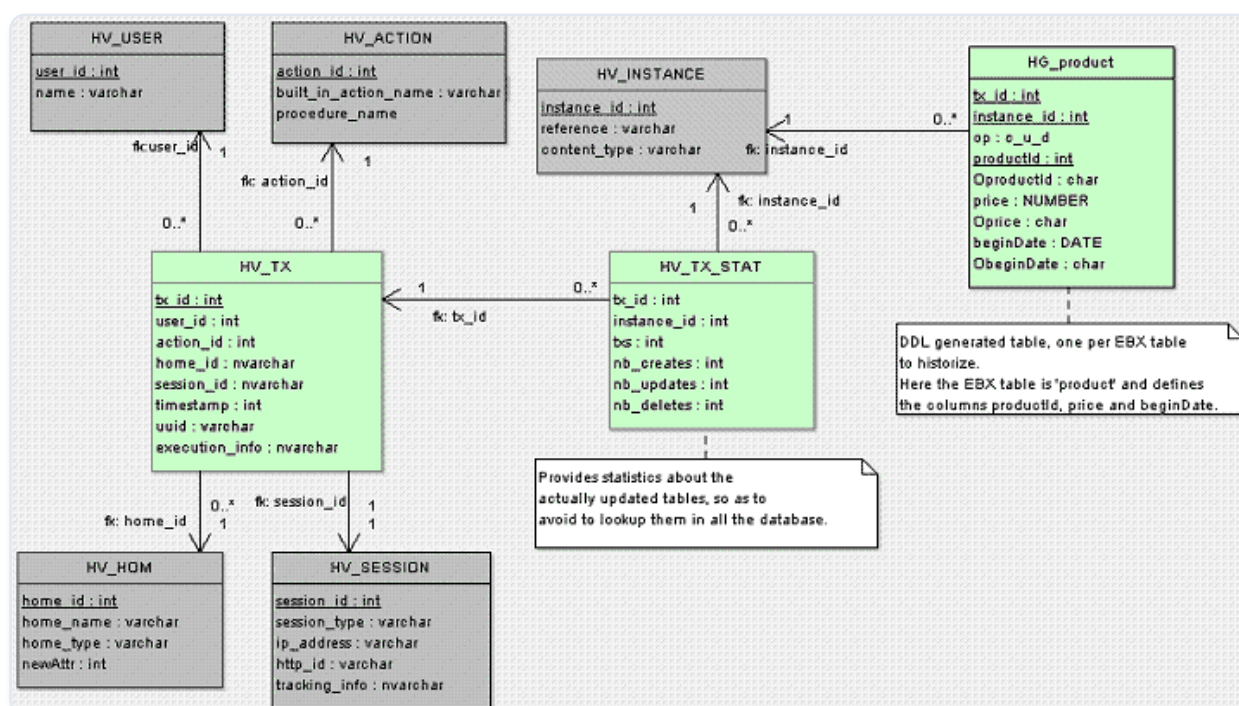
生成された履歴テーブルの例

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 -> {}` (履歴にエントリがありません)

操作フィールドの値

機能フィールドごとに、文字 `O` 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 (継承ではありません)。
- `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。
- `O` : 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ています。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、端末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。

関連するテーブルにすでに含まれているデータによっては、データモデルの進化が基盤となる 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
 - データベースで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
- レプリケーションの影響と制限
 - 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® リポジトリに格納されているデータを専用のリレーショナルテーブルにミラーリングして、SQL リクエストとビューによるデータへの直接アクセスを可能にすることができます。

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。特定のアクションは、データベース内のレプリカへの自動変更をトリガーします。

- 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、レプリケートされたテーブルに影響を与え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:

複製されたテーブル：複製されたプライマリデータテーブルを参照します。

レプリカテーブル (レプリカ)：レプリケーションのターゲットであるデータベーステーブルを参照します。

レプリケーションの構成

レプリケーションの有効化

データモデルでレプリケーションユニットを定義するには、エレメント `annotation/appinfo` の下にあるエレメント `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。一意である必要があります。	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。	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。	はい

エレメント	説明	必須
	• onCommit : データベース内のレプリカ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。 • onDemand : 指定されたテーブルのレプリケーションは、明示的な Update 操作が実行された場合にのみ実行されます。「「オンデマンド」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。	
table/path	データベースに複製される現在のデータモデルのテーブルのパスを指定します。	はい
table/nameInDatabase	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である必要があります。	はい

エレメント	説明	必須
<code>table/element/path</code>	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。	はい
<code>table/element/ nameInDatabase</code>	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である必要があります。	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：

- [レプリケート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、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せん。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

レプリケート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レプリケートされます (レプリケート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により、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介した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` の `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で定義されている場合、そのすべての子孫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、子孫でレプリケーション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](#)

データベースで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。

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

直接 SQL 読み取りは、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可能です。ただし、このようなアクセスの場合、EBX® 権限は考慮されません。その結果、読み取りを実行する権限を与えら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ている必要があります。

「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、レプリケートされたテーブルデータを更新するための明示的なリクエストが必要です。

レプリケーションの更新を要求するには、いくつかの方法があります。

- **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**：レプリケーションの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**：`ReplicationUnit.performRefresh` メソッド (`ReplicationUnit` API にあります) を呼び出して、レプリケーションユニットの更新を開始します。

レプリケーションの影響と制限

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

一部の EBX® データモデルの制約は、レプリケーションが有効になっている場合に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](#)を参照してください。

レプリケートされたテーブルのデータモデルの制限

複製される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か、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。
- フィールド継承は、「onDemand」更新ポリシーでのみサポートされます。これは、データモデルのコンパイル時に、更新モードが「onCommit」であり、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にエラーが報告されることを意味します。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。
- 別の集約リストの下集約リストと、端末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 (複製されません)。

- ユーザー定義の属性はサポートされていません。それらがレプリケーションユニットに含まれている場合、コンパイルエラーが発生します。

関連するテーブルにすでに含まれているデータによっては、データモデルの進化が基盤となる 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

Update 操作は、最後の更新以降に (作成と削除に関して) 変更された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、これは集中的な操作であり、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 Update 操作は多数の行に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を最適な条件下で実行できるように、データベースを適切に構成する必要があります。

Oracle の例は、次のとおりです。

- 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のが必須です。
- これらのトランザクションを最小限のディスクアクセスで実行できるように、バッファキャッシュに十分なスペースを提供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のを回避するのに十分な大きさの「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 プライマリと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です。

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行われます。

- アクションアイテム `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許可されていません。

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「値の継承」フラグ (`AdaptationValue.INHERIT_VALUE`) を保持できません。そのような場合にのみ継承された値を保持します。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にはいつでも、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/制限
 - 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管理](#)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つ以上のマップされた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管理](#)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

主キーの進化に関連する制限

主キー定義が変更された場合：

- テーブルのコンテンツは、すべてのデータセットとデータスペースで空のコンテンツにリセットされます。
- 新しい主キーが過去に使用された場合、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ときに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。
- テーブルに既存のデータスペースでアクティブ化された履歴がある場合、または履歴がアクティブ化されている場合、変更は拒否されます。考えられる回避策：最初に専用テーブルに関連付けられた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パージする手順については、[データベースマッピング管理](#)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連する制限

- `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された場合を除く。この場合、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、ターゲットの主キーの構造と一致するように設定されます。次に、`osd:tableRef` 制約を宣言する単一のフィールドを、ターゲットの主キーの数と型に対応するいくつかの列に分割できます。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で新しい `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は、フィールドの削除や作成の組み合わせに変換されます。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連する制限

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ものと見なされ、空のコンテンツで始まります。モデルが前の定義にロールバックされると、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的な表現を持ち、次のリストで相互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`（小数部分を失う）

- `xs:int` から `xs:integer`
- `xs:datetime` から `xs:date` (時間部分を失う)
- `xs:date` から `xs:datetime` (デフォルトでは時間部分は0)
- 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、次のように動作します。

- 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。他の値は失われます。

注意:

グループおよび複合型は、他の型への（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発生と複数発生の間で変化する場合、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](#)
- [パフォーマンスとチューニング](#)

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
 - 値検索メカニズム
 - 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
 - 値検索メカニズム
- サービスの最適化とリファクタリング
 - 手順の詳細
 - 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、フィールドの継承はデータセットの継承よりも優先されます。

計算値（関数）

データモデルでは、ノードが**計算値**を保持するように指定することもできます。この場合、指定されたJavaBean関数は、値が要求されるたびに実行されます。

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、現在のコンテキストを考慮に入れて、サードパーティのシステムに要求を送信することができます。

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、プロパティ `dataSetInheritance` を定義して、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で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いて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。

値検索メカニズム

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` にすることができます。

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：****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されたノード
- 計算値を定義するノード

レコードルックアップメカニズム

テーブルレコードも値と同様に、複数のコンテキストに1つの単位として継承されますが、部分的に再定義 (上書き) されたり、特定のコンテキストのために定義 (ルートモード) されたり、非表示にされたりすることもあります。

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である必要があります。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である必要があります。
- 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である必要があります。

- 。ソースノードは書き込み可能である必要があります。
- 。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。

値検索メカニズム

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。

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。

手順の詳細

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に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
- 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。

- その権限を管理します。
- データセットがルートデータセットの場合は、所有者を変更します。
- 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

UI で非機密情報を非表示にする目的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フィルタリング/結合/並べ替えに使用される可能性が高い場合は注意が必要です。そのような場合は、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非表示として設定したり、関係するユーザーのビューを作成したりします。

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は、テーブル内のレコードを正しくフィルター処理しますが、テーブルの履歴には適用できません。したがって、テーブル履歴へのアクセスはデフォルトで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も役立ちます。

権限ルールを編集することで、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。スクリプト化されたルールが適用されないことから、これらのユーザーは、すべてのレコードの履歴にアクセスできるため、注意が必要です。

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 を使用するときには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。

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非表示の場合に、`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、すべてのノードに機密性が適用されます。

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	認証
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。 - 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 323。

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。
重複するデータセット	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上書き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上書きします。

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。

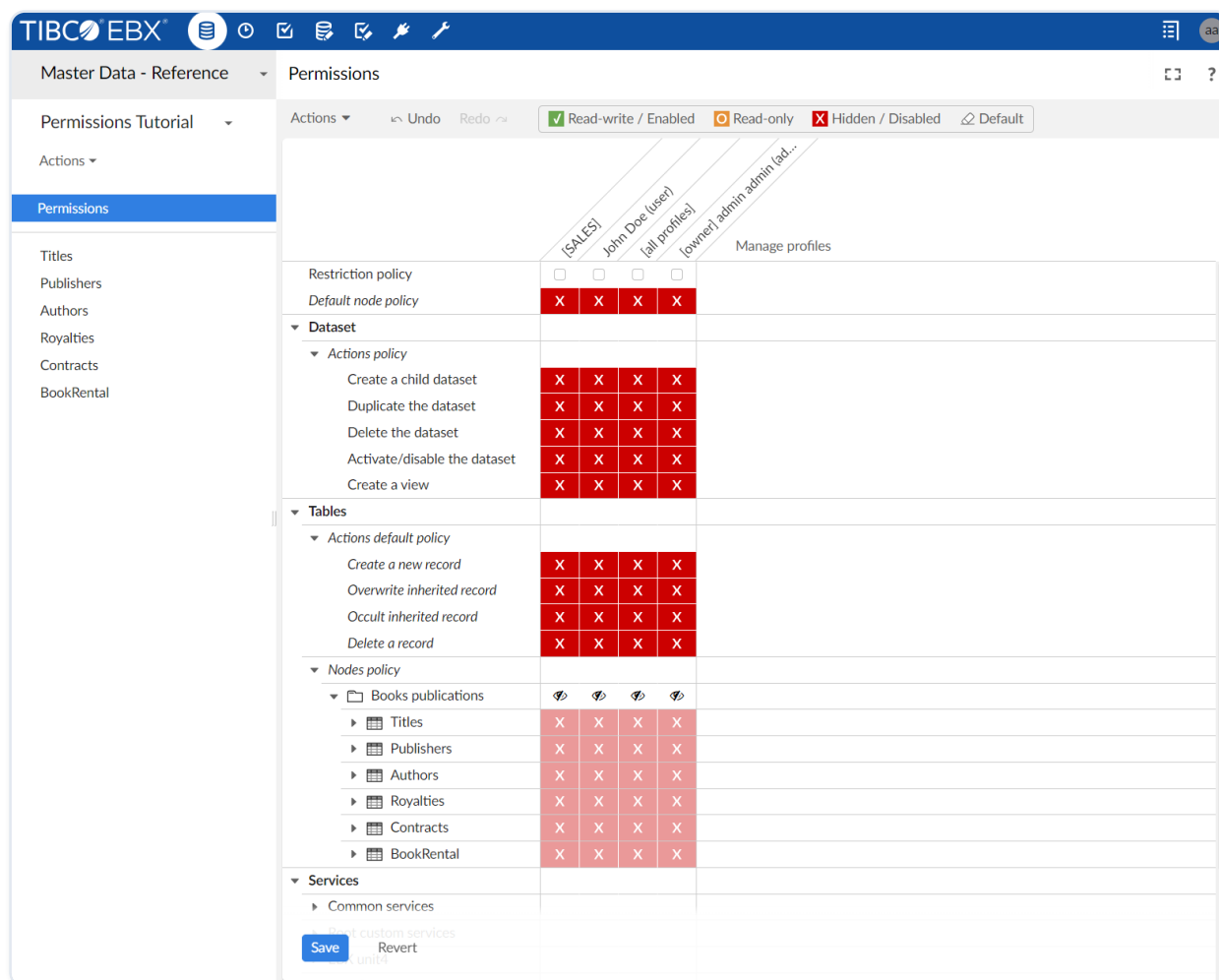
データセット権限構成サービスの使用

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。詳細については、[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。詳細については、[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

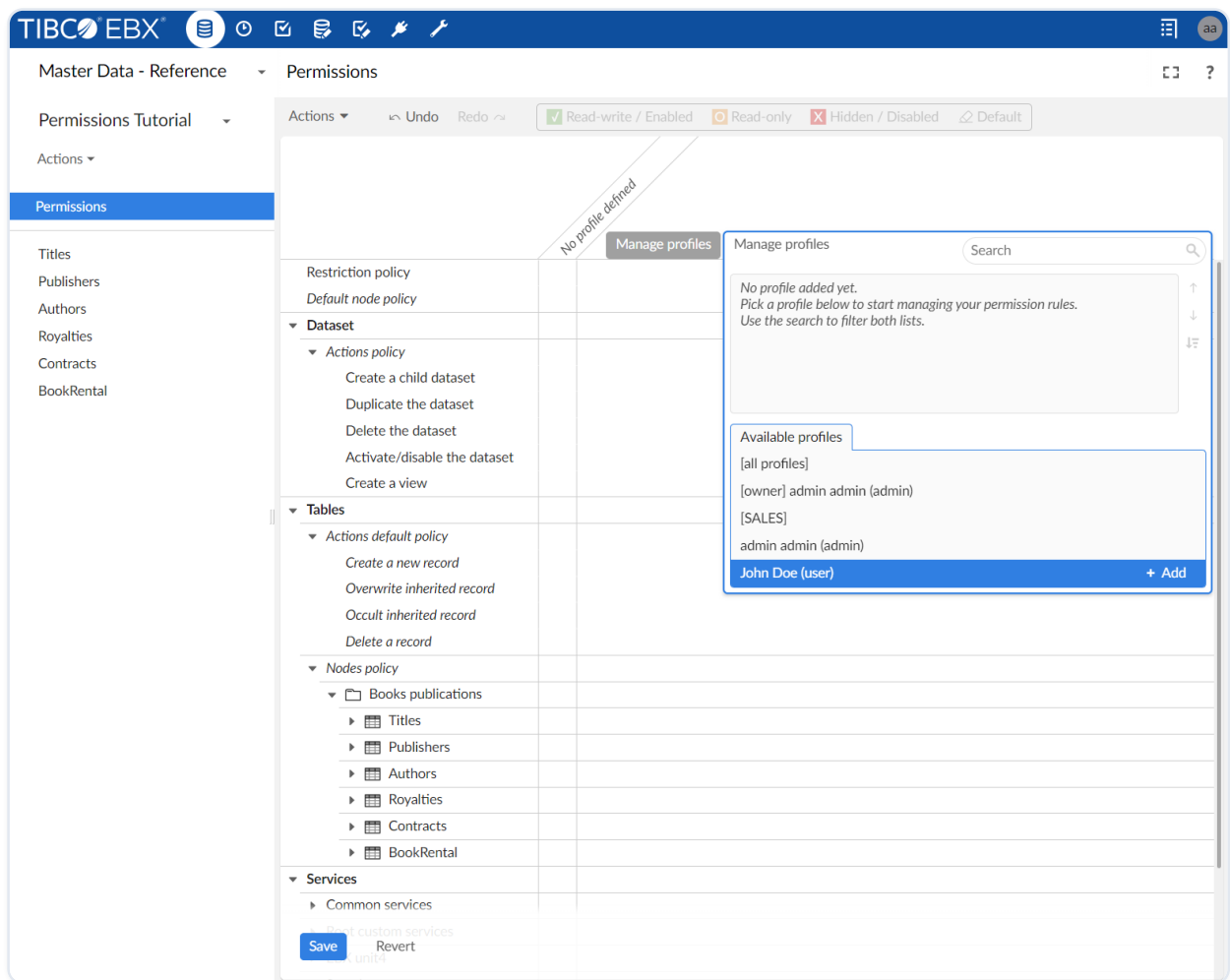
プロファイル管理タスクには次のものが含まれます。

- 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。

The screenshot shows the TIBCO EBX 6.1.2 Permissions configuration interface. The interface is divided into a sidebar and a main content area. The sidebar contains a 'Permissions Tutorial' section and a 'Permissions' section. The main content area displays a table of permissions for various entities (Titles, Publishers, Authors, etc.) across different users (SALES, John Doe, etc.). The table includes columns for 'Read-write / Enabled', 'Read-only', 'Hidden / Disabled', and 'Default'. The 'Read-write / Enabled' column is highlighted with a green checkmark.

Entity	SALES	John Doe (user)	[all profiles]	admin admin (admin)	Manage profiles
Restriction policy					
Default node policy	X	X	X	X	
Dataset					
Actions policy					
Create a child dataset	X	X	X	X	
Duplicate the dataset	X	X	X	X	
Delete the dataset	X	X	X	X	
Activate/disable the dataset	X	X	X	X	
Create a view	X	X	X	X	
Tables					
Actions default policy					
Create a new record	X	X	X	X	✓
Overwrite inherited record	X	X	X	X	
Occult inherited record	X	X	X	X	
Delete a record	X	X	X	X	
Nodes policy					
Books publications					
Titles	X	X	X	X	
Publishers	X	X	X	X	
Authors	X	X	X	X	
Royalties	X	X	X	X	
Contracts	X	X	X	X	
BookRental	X	X	X	X	
Services					
Common services					
Custom services					

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 of 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上書き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、解決プロセスには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、`SchemeExtensionsContext.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。

詳細については、`SchemeExtensionsContext.setAccessRuleOnNode` を参照してください。

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、すべての要求に応じて、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。

- `UserServiceDeclaration.defineActivation` メソッド内でも定義されている潜在的な動的アクティベーションルール (`ServiceActivationRule`)。

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)。

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](#)
 - [表現](#)

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。それらは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。

- **一致する基準がありません**：ブロック内のどの基準も一致しません。
- **すべての基準が一致するわけではありません**：ブロック内の少なくとも1つの基準が一致しません。

- **すべての基準が一致**：ブロック内のすべての基準が一致します。
- **少なくとも1つの基準が一致します**：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。
オペレーター	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。

表現

式は、固定値または数式のいずれかです。フィルタを作成するときは、固定値のみが許可されます。検証ルール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：****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
- 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、生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生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、生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 (`<osd:enumeration osd:path=...`) を使用して定義された列挙制約。別の解決策 (外部キー) の使用をお勧めします。

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。

- ValueFunction API を実装する計算フィールド、またはコンテナテーブルのフィールドのみに依存しない計算ルールとスクリプト。
- 継承フィールド

結果として、これらは通常、 `osd:search('', 'pattern')` による [クイック検索](#) および [テーブルの xpath 検索](#) 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)。
- 関連性ソート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ソート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領域を消費します。

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`) が使用できません。

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領域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」、SQLDISTINCT および COUNTDISTINCT、...) 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。

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ソート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ソート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

関連性の検索戦略

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である必要があります。

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。

パフォーマンスとチューニング

- 環境
 - メモリ管理
 - ディスク
 - CPU
 - ネイティブ LZ4 ライブラリの使用
 - 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
- 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
 - 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

アプリケーション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。
オペレーティングシステム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。

	<pre>ulimit -n 512000</pre> <pre>sysctl vm.max_map_count=262144</pre>
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。

ディスク

EBX® リポジトリデータは Lucene インデックスにインデックス付けされ、[ルートディレクトリ](#)の下のディスクに保存されます。

*ディスク容量：*ディスクサイズの経験則では、リレーショナルデータベースのテーブル `G_BLK` が占める容量の 10 倍を計画します。

*ディスク遅延：*全体的なパフォーマンスを良好に維持するには、Lucene インデックスを格納するディスクの遅延時間を短くすることが重要です。

以下も参照してください。：

- [EBX® ルートディレクトリの設定](#)

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。

ネイティブ LZ4 ライブラリの使用

LZ4 ライブラリは、データベースへのデータの保存とデータベースからのデータの取得に使用されます。データアクセスを高速化するには、`ebx-lz4.jar` ネイティブインストールを実行する必要があります。

詳細については、[データ圧縮ライブラリ](#)を参照してください。

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ます。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。

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。

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ければ、**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。

- ファセットパターン¹のチェックは大規模なテーブルでは最適化されていないため、使用を避けてください。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、検証プロセスはすべてのレコードを繰り返し処理して、関連付けられた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します。

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²、クエリ³ 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⁴、プログラムフィルター⁵、またはレコードレベルの権限ルール⁶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

テーブルの操作速度を向上させるために、永続的な Lucene インデックスは EBX® エンジンによって管理されます。

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.getSize` ではなく `RequestResult.isSizeGreaterOrEqual` 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。

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
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。

	1. 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 <code>プロシージャ</code> (または <code>TableTrigger</code>) 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。 2. 標準の <code>クエリ</code> (または <code>リクエスト</code>) の場合、現在の <code>プロシージャ</code> (または <code>TableTrigger</code>) 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、<code>doCreateOccurrence</code> または <code>doModifyContent</code> への最後の呼び出しから返された <code>Adaptation</code> 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。
- バッチプロセス内で **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 **プロセス** 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があります。

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。

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
 - Java EE 展開
 - インストールに関する注意事項
 - JBoss EAP 7.4.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 9.x のインストールに関する注意事項
 - WebLogic 14c 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
- EBX® Container Edition
 - イメージの構築
 - 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
 - ワークフロー管理
 - タスクスケジューラ

- 監査証跡
- コラボレーション機能 (チームアップ)
- 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

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
 - 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

- 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。[アーカイブディレクトリ](#)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](#)

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。

メール構成

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構成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® には自動削除機能が用意されており、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異なる数の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。

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 自動構成が最初に実行されます。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
- Java EE 展開
- インストールに関する注意事項
 - JBoss EAP 7.4.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 9.x のインストールに関する注意事項
- WebLogic 14c 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

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® の Web インターフェイスは、次のブラウザーをサポートします。

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を尽くします。
Mozilla Firefox ESR 102 (詳細を参照)	Mozilla Firefox は頻繁に更新されるため、Cloud Software Group, Inc. はバージョン ESR 102 のみを完全にサポートしていま

	す。詳細については、 Mozilla Firefox ESR を参照してください。
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を尽くします。

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

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

EBX® は次の構成をサポートします。

- Java ランタイム環境: JRE 11 または 17。

注意：ランタイムを最新の LTS バージョンにアップグレードして、提供されるパフォーマンスの向上を利用することをお勧めします。

- Servlet 3.0 (包括的) から 5.0 (排他的) までに準拠する任意のサーブレット/JSP コンテナ：たとえば、Tomcat 7.0 (包括的) から 10.0 (排他的)まで。Java SE 11 をサポートし、Jakarta EE 9 以降を使用しない Java アプリケーションサーバー：Java EE 8 用の WebSphere Liberty 20 以降、WebLogic Application Server 14c 以降、JBoss EAP 7.4 以降など。[Java EE 展開の概要](#)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` ライブラリに埋め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 (埋め込みのもののみを使用できます)。WebLogic Application Server 14c 以降はこの仕様をサポートし、WebSphere Application Server Liberty と JBoss EAP 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 以降。

`null` 値の区別には、特定の制限があります。単純な `xs:string` エレメントでは、Oracle は空の文字列と `null` 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 11 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、マッピングされた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 11.16 (PostgreSQL 11.16 以降と互換性があります)。	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 11 以降	PostgreSQL に関する上記セクションのコメントが適用されます。
SAP HANA Database 2.0 以降	基盤となるデータベースとして SAP HANA Database を使用する場合、特定のスキーマの進展はサポートされません。たとえば、列の長さを短くすることはできません。これは、SQL のリファレンスガイドで説明されているように、HANA の制限です (行テーブルの場合、VARCHAR および NVARCHAR タイプの列のサイズのみ大きくすることが許可されます)。
Microsoft SQL Server 2014 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 2.1.212 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

Java EE 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
 - 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およびロールディレクトリ。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 Log4j 2 API
- ArcGIS API for JavaScript
- Javax Activation

- Javax Annotations
- Javax JSON Binding
- Javax SAAJ API
- Javax WS RS
- Javax XML Bind
- LZ4 compression for Java
- MicroProfile OpenAPI
- Simple Logging Facade for Java (SLF4J) API

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_6.1.2_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-lz4.jar` という名前のライブラリは、`ebx.jar` とは別に展開する必要があります。これには、いくつかの圧縮実装が含まれています (JNI 専用アーキテクチャライブラリと Java フォールバック)。以下の要件により、EBX® リポジトリの最適な圧縮および解凍パフォーマンスを確保できます。前提条件を検証できない場合、EBX® は Java フォールバックの安全または安全でないモードで機能しますが、パフォーマンスは低下します。`ebx-lz4.jar` ライブラリのデフォルトの場所は、`ebx.jar` の横にあります。

EBX® リポジトリで実際に使用されている圧縮の実装を確認するには、> のセクションで の値を確認してください。最適なパフォーマンスを得るには、この値が「`JNI-検証済み`」である必要があります。それ以外の場合は、Java フォールバックに対して「`Java[Safe|Unsafe] - 検証済み`」になります。

パフォーマンス要件

JNI アクセスは、次のオペレーティングシステムアーキテクチャで許可されています。 `i386`、`x86`、`amd64`、`x86_64`、`aarch64` または `ppc64le`。この値を確認するには、> のセクションで の値を確認してください。

`ebx-lz4.jar` の JNI アクセスを有効にするには、ライブラリをシステムクラスローダー (アプリケーションクラスローダーとも呼ばれる) でロードする必要があります。展開は、[アプリケーションサーバー固有の手順](#)に従って行えます。

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 2.1.212 が検証されました。H2 は実稼働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 21c 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、使用する必要のあるドライバーを決定します。 JDK8 で <code>ojdbc8.jar</code> 認定ライブラリ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 2012 SP4 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。 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-8.4.1.jre8.jar</code> または <code>mssql-jdbc-8.4.1.jre11.jar</code> ライブラリを追加します。 Microsoft JDBC Driver 8.4.1 for SQL Server (zip) のダウンロード
PostgreSQL	PostgreSQL 10 以降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

使用される Web アプリケーションサーバーによっては、メール管理用のライブラリ `JavaMail API` がすでに提供されています。そうでない場合は、手動で追加する必要があります。

EBX® には、この API のバージョン 1.5.6 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、`javax.mail-1.5.6.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。:

- [JavaMail](#)

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 1.1 以降が必要です。

Java 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とおりです。

- Java EE (Java Platform 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がデフォルトで使用可能です。詳細については、<https://www.oracle.com/java/technologies/java-ee-glance.html> を参照してください。
- Apache Tomcat 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](#)

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
<code>ebx</code>	起動時に初期化を処理する 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

WEB アプリケーション名	説明	必須
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-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、 ebx-manager 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、 ebx-manager 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

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。

<code>/web-app/servlet/load-on-startup</code>	EBX® の起動時に <code>FrontServlet</code> が確実に初期化されるようにするには、 <code>web.xml</code> 展開記述子でエレメント <code><load-on-startup>1</load-on-startup></code> を指定する必要があります。
<code>/web-app/servlet-mapping/url-pattern</code>	<code>FrontServlet</code> はパス「 <code>/</code> 」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"/>
```

- バージョン 6.1.2 以降非推奨：iFrame を使用する場合など、複数の EBX® Web コンポーネントを同じ HTML ページに表示する場合、一部のインターネットブラウザの制限により Cookie の管理を無効にする必要がある場合があります。

たとえば、Tomcat では、この構成は、構成ファイル `server.xml` の属性 `cookies` によって次のように指定されます。

```
<Context path="/ebx" docBase="(...)" cookies="false"/>
```

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>	Weblogic : <code>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
mail/EBX_MAIL_SESSION	Weblogic : EBX_MAIL_SESSION JBoss : java:/ EBX_MAIL_SESSION	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 : javax.mail.Session

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
jms/ EBX_JMSConnectionFactory	Weblogic : EBX_JMSConnectionFactory JBoss : java:/ EBX_JMSConnectionFactory	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 : javax.jms.ConnectionFactory	はい

注意:

JNDI 機能を備えた WildFly、JBoss、および WebLogic アプリケーションサーバーでの展開の場合、`EBX.ear` または `EBXForWebLogic.ear` を更新して、必要なすべてのリソース名を JNDI 名にマッピング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 要求を受信すると、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
<code>jms/</code> <code>EBX_QueueIn</code>	Weblogic : <code>EBX_QueueIn</code> JBoss : <code>java:/</code> <code>jms/EBX_QueueIn</code>	他のアプリケーションによって EBX® に送信される着信 SOAP 要求の JMS キュー。 Javaタイプ : <code>javax.jms.Queue</code>	いいえ
<code>jms/</code> <code>EBX_QueueFailure</code>	Weblogic : <code>EBX_QueueFailure</code> JBoss : <code>java:/</code> <code>jms/</code> <code>EBX_QueueFailure</code>	エラー時の JMS キュー。これには、エラー発生時の受信 SOAP 要求が含まれています。これにより、必要に応じてこれ	いいえ

予約済みリソース名	デフォルトの JNDI 名	説明	必須
		らのメッセージを再生できます。 Javaタイプ： <code>javax.jms.Queue</code> **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 <code>ebx.jms.activate.queueFailure</code> 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。	

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、これは `catalina.properties` ファイルの `tomcat.util.scan.DefaultJarScanner.jarsToSkip` プロパティで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、サーブレット 3.0 から 5.0 までをサポートする任意の Java EE または Jakarta EE 8 アプリケーションサーバーに展開できます。展開とインストールに関する次のドキュメントが利用可能です。

- [JBoss EAP 7.4.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 9.x のインストールに関する注意事項](#)

- WebLogic 14c のインストールに関する注意事項

注意:

- Java EE または Jakarta EE 8 アプリケーションサーバーでの EBX® インストールに関する注意事項は、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を置き換えるものではありません。
- 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 の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。
- Oracle WebLogic サーバーに展開する場合は、[モジュール構造](#)のセクションを参照してください。

インストールに関する注意事項

- [JBoss EAP 7.4.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 9.x のインストールに関する注意事項](#)
- [WebLogic 14c のインストールに関する注意事項](#)

JBoss EAP 7.4.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完全な説明は、[Java 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に、クラスローディングシステム内の `ebx.jar`、`ebx-lz4.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。
- 安全な導入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 11 または 17 ([アップデート 7 以降](#))
- JBoss Application Server EAP 7.4
- データベースと JDBC ドライバー
- EBX® CD
- 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 7.4.0 `インストーラー.jar` バージョンを以下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、
 - 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. のチェックをオフにし、 をクリックします。
 - k. JBoss EAP 7.4 の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。
 - l. をクリックします。
4. JBoss EAP 7.4.0 の最新の更新をダウンロードして適用します。このリンクには認証が必要です。

<https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?downloadType=patches&product=appplatform&version=7.4>

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. 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上書きします。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 仮想マシンの設定

1. `<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)。

2. 「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。
3. 「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"
```

4. *EBX®* の CD から `ebx.software/lib/ebx-lz4.jar` データ圧縮ライブラリを専用ディレクトリ (例: `<JBoss_HOME>/compress`) にコピーします。
5. `<JBoss_HOME>/bin` にある `standalone.sh` スクリプトファイルを開きます。
6. 次のような 「CLASSPATH」 環境変数を作成します。

```
CLASSPATH="<path_to_the_data_compression_library>:${JBoss_HOME}/jboss-modules.jar:${CLASSPATH}"
```

```
# For our example
```

```
# CLASSPATH="${JBoss_HOME}/compress/ebx-lz4.jar:${JBoss_HOME}/jboss-modules.jar:${CLASSPATH}"
```

7. 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 [ "x$LAUNCH_JBOSS_IN_BACKGROUND" = "x" ]; then

    # Execute the JVM in the foreground

    eval \"\$JAVA\" -D\"[Standalone]\" $JAVA_OPTS \

    -cp \"$CLASSPATH\" \
```

```

        \"-Dorg.jboss.boot.log.file="$JBOSS_LOG_DIR"/server.log\" \

        \"-Dlogging.configuration=file:"$JBOSS_CONFIG_DIR"/
logging.properties\" \

        org.jboss.modules.Main \

        $MODULE_OPTS \

        -mp \""${JBOSS_MODULEPATH}\"\" \

        org.jboss.as.standalone \

        -Djboss.home.dir=\""$JBOSS_HOME\"\" \

        -Djboss.server.base.dir=\""$JBOSS_BASE_DIR\"\" \

        "$SERVER_OPTS"

        JBOSS_STATUS=$?

else

    # Execute the JVM in the background

    eval \"$JAVA\" -D\"[Standalone]\" $JAVA_OPTS \

        -cp "$CLASSPATH" \

        \"-Dorg.jboss.boot.log.file="$JBOSS_LOG_DIR"/server.log\" \

        \"-Dlogging.configuration=file:"$JBOSS_CONFIG_DIR"/
logging.properties\" \

        org.jboss.modules.Main \

        $MODULE_OPTS \

        -mp \""${JBOSS_MODULEPATH}\"\" \

        org.jboss.as.standalone \

```



```

-Djboss.home.dir=\ "${JBOSS_HOME}" \

-Djboss.server.base.dir=\ "${JBOSS_BASE_DIR}" \

"${SERVER_OPTS}" "&"

...

fi

```

JNDI エントリの構成

1. `<JBOSS_HOME>/Standalone/configuration` にある `standalone-full.xml` ファイルを開きます。
2. `messagesaging-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、[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=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.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. 「data-source add」 コマンドを使用してデータソースを定義し、適切な引数値を指定します。PostgreSQL 構成の設定例を示します。

```
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>
```

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 Message Service \(JMS\)](#) 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 -cstandalone-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。URL : <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完全な説明は、[Java 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`、`ebx-lz4.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。
- 安全な導入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 11 以降。IBM Semeru または Eclipse Adoptium Java ディストリビューションを使用することをお勧めします。
- Java EE 8 用の WebSphere Application Server Liberty 23.x
- データベースと JDBC ドライバー
- EBX® CD
- 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/23001-websphere-application-server-liberty-23001>.

次の例では、*Liberty with Java EE 8 Web Profile* zip ディストリビューションを使用しています。

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-8.0</feature>` タグを次の行に置き換えます。

```
<feature>javaMail-1.5</feature>
```

```
<feature>jdbc-4.2</feature>
```

```
<feature>jndi-1.0</feature>
```

```
<feature>servlet-4.0</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要求が受信されるまで、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、既存の要求パラメーターに `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. 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上書きします。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)
```

```
ebx-lz4.jar ( EBX® CD の ebx.software/lib/ ディレクトリから取得)
```

これらのコンポーネントの完全な説明は、[Java 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: `javaMail-1.5` 機能がこのサーバーに対してアクティブ化されているため、`javax.mail 1.5.x` がすでに存在する)。

JNDI エントリの構成

1. WebSphere Application Server Liberty での JMS 構成については、[JMS 接続ファクトリの構成](#)を参照してください。

2. WebSphere Application Server Liberty での *JavaMail* 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 ドライバー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、ドキュメント [Liberty の認証エイリアスの構成](#) 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
```

3. EBX® 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="ebxLibLz4">

  <file name="${EBX_LIB}/ebx-lz4.jar"/>

</library>
```

注意:

WebSphere Application Server Liberty は **LZ4 JNI 実装** をサポートしていません。

4. EBX.ear を展開するには、`<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。

```
<application id="ebxApp" location="${EBX_HOME}/ear/EBX.ear"
name="ebxApp" type="ear">

  <classloader commonLibraryRef="ebxLibLz4" />

</application>
```

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:9080/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。

Tomcat 9.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。
- Tomcat 10.x はサポートされていません。
- EBX® に必要なコンポーネントの完全な説明は、[Java 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に、クラスローディングシステム内の `ebx.jar`、`ebx-lz4.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。
- 安全な導入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 11 または 17

- Apache Tomcat 9.x
- データベースと JDBC ドライバー
- EBX® CD

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. 以下 から Tomcat 9.x コアバイナリディストリビューションをダウンロードします。

<https://tomcat.apache.org/download-90.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. 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上書きします。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。

属性 `encodedSolidusHandling="passthrough"` を `Connector` エlement に追加します。

次の行を `<Host>` Element に追加します。

```
<Context path="/ebx" crossContext="true" docBase="ebx.war"/>
```

この例では、次のようになります。

```
<Host 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.DefaultJarScanner.jarsToSkip` 属性に以下の行を追加し、修正します。

```
ebx.jar,\
```

```
ebx-addons.jar,\
```

```
ebx-lz4.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%
-Dorg.apache.catalina.connector.CoyoteAdapter.ALLOW_BACKSLASH=true
%JAVA_OPTS%

set CLASSPATH=<$CATALINA_HOME_or_$CATALINA_BASE>\compress\ebx-lz4.jar;%CLASSPATH%
```

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 *Java* オプションフィールドで、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"
```

//US// パラメーターを使用してサービスを更新し、適切なクラスパス値を設定します。

```
C:\> tomcat9 //US//Tomcat9 --
Classpath="<$CATALINA_HOME_or_$CATALINA_BASE>\compress\ebx-lz4.jar;
%CLASSPATH%"
```

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}"
-Dorg.apache.catalina.connector.CoyoteAdapter.ALLOW_BACKSLASH=true $
```

```
{JAVA_OPTS}"
```

```
export CLASSPATH="<$CATALINA_HOME_or_$CATALINA_BASE>/compress/ebx-  
lz4.jar:${CLASSPATH}"
```

<\$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:

Tomcat オプション

`-Dorg.apache.catalina.connector.CoyoteAdapter.ALLOW_BACKSLASH=true` は、エンコードされたバックスラッシュを含む URL のブロックを回避します。

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。

注意:

データ圧縮ライブラリのネイティブインストールの場合は、`CLASSPATH` 環境変数で参照することだけを確認してください。

EBX® およびサードパーティライブラリの展開

1. `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。ただし、データ圧縮ライブラリは除いてください。今回の例では、以下のようになります。

`$CATALINA_HOME/lib/javax.mail-1.5.6.jar` は、`ebx.software/lib/lib-mail` ディレクトリから取得されます。

`$CATALINA_HOME/lib/h2-2.1.212.jar` (デフォルトの永続化ファクトリー、`ebx.software/lib/lib-h2` ディレクトリ下)

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. [データ圧縮ライブラリ](#)専用のディレクトリ (例: `$CATALINA_HOME/compress` または `$CATALINA_BASE/compress`) を作成し、そこにコピーします。

注意:

ライブラリが、以前に更新された `CLASSPATH` 環境変数によって示されるディレクトリにコピーされていることを確認します。

3. *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-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-manager.war` : マスターデータ管理 Web 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-ui.war` : ユーザーインターフェイス 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. 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。

WebLogic 14c のインストールに関する注意事項

- 概要
- 要件
- WebLogic Application Server のインストール
- EBX® ホームディレクトリの構成
- WebLogic 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
- データソースと JDBC プロバイダーの構成
- 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、WebLogic 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完全な説明は、[Java 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に、クラスローディングシステム内の `ebx.jar`、`ebx-lz4.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。
- 安全な導入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

- 認定された Oracle Java SE 11 ([アップデート 6 以降](#))
- WebLogic Server 14c
- データベースと JDBC ドライバー
- EBX® CD

以下も参照してください。:

- [サポートされる環境](#)

WebLogic Application Server のインストール

1. 次の場所から WebLogic 14c の最新バージョンをダウンロードします。

<https://www.oracle.com/middleware/technologies/fusionmiddleware-downloads.html>

2. 認定された Oracle JDK と `java -jar` コマンドラインを使用して、*Oracle Fusion Middleware Weblogic* インストールウィザードを実行します。
3. 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、適切なインストールディレクトリを選択します。
4. 次の手順を実行するには、オプションを有効のままにします。
 - a. ドメインの作成: を選択し、ドメインのホームディレクトリを指定して、をクリックします。
 - b. テンプレート: デフォルトのままにして、をクリックします。
 - c. 管理者アカウント: ドメイン管理者のユーザー名とパスワードを入力し、をクリックします。

- d. ドメインモードと JDK：実稼働モードと JDK インストールホームを選択し、をクリックします。
- e. 高度な構成：「管理サーバー」と「トポロジ」を確認します。このようにして、管理ノードとアプリケーションノードの 2 つの独立したドメインノードを作成します。

をクリックします。
- f. 管理サーバー：管理ノード名 (例：「AdminServer」) とリッスンポート (デフォルトでは「7001」) を入力し、をクリックします。
- g. 管理対象サーバー：アプリケーションノード名 (例：「EbxServer」) とリッスンポート (例：7003) を追加し、をクリックします。
- h. クラスター：デフォルトのままにして、をクリックします。
- i. サーバーテンプレート：デフォルトのままにして、をクリックします。
- j. マシン：デフォルトのままにして、をクリックします。
- k. 構成の概要：をクリックします。
- l. 構成プロセス：をクリックします。
- m. 構成の終了：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：`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. 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上書きします。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。

WebLogic アプリケーションサーバーと Java 仮想マシンの構成

1. 管理対象サーバーの起動プロパティを構成します (例：「EbxServer」)。

`<DOMAIN_HOME>/bin/startManagedWebLogic.sh` スクリプトファイルに次の行を追加します。

```
EBX_HOME="<path_to_the_directory_ebx_home>"

EBX_OPTIONS="-Debx.home=${EBX_HOME} -Debx.properties=${EBX_HOME}/
ebx.properties"

export JAVA_OPTIONS="${EBX_OPTIONS} ${JAVA_OPTIONS}"
```

2. `<DOMAIN_HOME>/bin/setDomainEnv.sh` スクリプトファイルに次の行を追加します。

```
PRE_CLASSPATH="<path_to_the_data_compression_library>"

# For our example

# PRE_CLASSPATH="${DOMAIN_HOME}/compress/ebx-lz4.jar"
```

EBX® およびサードパーティライブラリの展開

1. **データ圧縮ライブラリ**を除いて、サードパーティライブラリを *EBX® CD* から `<DOMAIN_HOME>/lib` ディレクトリにコピーします。この例では、H2 スタンドアロンデータベースの場合、次のようになります。

```
<DOMAIN_HOME>/lib/h2-2.1.212.jar (default persistence factory) coming from
ebx.software/lib/lib-h2 directory.
```

EBX® に必要なコンポーネントの完全な説明は、[Java EE 展開](#)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、それらを複製する必要はありません (例: `javax.mail-1.5.6.jar` がすでに WebLogic Server に存在する)。

2. **データ圧縮ライブラリ**専用のディレクトリ (例: `<DOMAIN_HOME>/compress`) を作成し、そこにコピーします。

注意:

ライブラリが、以前に更新された *PRE_CLASSPATH* 環境変数によって示されるディレクトリにコピーされていることを確認します。

データソースと JDBC プロバイダーの構成

1. 以下を使用して、「管理サーバー」(例:「AdminServer」)を起動します。

```
<DOMAIN_HOME>/bin/startWebLogic.sh
```

2. ブラウザーに次の URL を入力して、を起動します。

<http://localhost:7001/console>

ドメイン管理者のユーザー名とパスワードでログインします。

3. パネルの > をクリックしてから、> をクリックします。
 - a. 次を設定します。タイプ名: `EBX_REPOSITORY`、JNDI名: `EBX_REPOSITORY`、データベースタイプ: *使用しているデータベースタイプ* をクリックします。
 - b. データベースドライバの種類を選択し、 をクリックします。
 - c. のチェックをオフにし、 をクリックします。
 - d. データベースの を設定し、 をクリックします。
 - e. 、 を順にクリックします。
 - f. タブですべてのサーバーを選択し、 をクリックします。
 - g. 以下を使用して、管理サーバー (例:「AdminServer」) を再起動します。

```
<DOMAIN_HOME>/bin/stopWebLogic.sh
```

```
<DOMAIN_HOME>/bin/startWebLogic.sh
```

EBX® アプリケーションの展開

1. *EBX® CD* の `ebx.software/webapps/ear-packaging/EBXForWebLogic.ear` から *EBX_HOME* 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。

```
C:\EBX\home\EBXForWebLogic.ear または /home/ebx/EBXForWebLogic.ear
```

2. ブラウザーに次の URL を入力して、を起動します。

<http://localhost:7001/console>

3. パネルで をクリックします。
4. パネルの をクリックし、 をクリックします。
 - a. アプリケーションアシスタントのインストール： に、 `C:\EBX\home\` または `/home/ebx/` ディレクトリにある EBXForWebLogic.ear ファイルへのアプリケーションのフルパスを入力します。をクリックします。
 - b. インストールの種類とスコープを選択します。、 のデフォルトスコープをクリックし、 をクリックします。
 - c. 展開ターゲットを選択します。「サーバー」リストからノード (「EbxServer」など) を選択し、 をクリックします。
 - d. オプション設定：デフォルトのままにして、 をクリックします。
5. 左上隅にある をクリックします。展開ステータスが「準備済み」に変わります。
6. タブに切り替え、「EBXForWebLogic」エンタープライズアプリケーションを選択し、> をクリックします。
7. 以下を使用して、アプリケーションノード (例：「EbxServer」) を起動します。

```
<DOMAIN_HOME>/bin/startManagedWebLogic.sh EbxServer http://localhost:7001
```

EBX® アプリケーションの開始

1. WebLogic Application Server 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:7003/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。

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
- 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

- 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
- 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
- フェールオーバーの構成
- EBX® リポジトリの調整
- 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ソートす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
- 検証レポートページ
- 検証レポートログ

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、Java 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から追加の構成を実行することもできます。構成後、これらは 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。

この標準の Java 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、サブレット `FrontServlet` のメソッド `ServletConfig.getInitParameter ("ebx.properties")` を呼び出すことでこのパラメーターにアクセスします。

詳細については、Oracle [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。

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).

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.persistence.timeout=10000
```

```
#####
```

```
## The prefix to add to all table names of persistence system.
```

```
## This may be useful for supporting multiple repositories in the  
relational database.
```

```

## Default value is 'EBX_'.

#####

ebx.persistence.table.prefix=

#####

## Case EBX® persistence system is H2 'standalone'.

#####

ebx.persistence.factory=h2.standalone

ebx.persistence.user=sa

ebx.persistence.password=

#####

## Case EBX® persistence system is H2 'server mode',

#####

#ebx.persistence.factory=h2.server

## Specific properties to be set only if you want to ignore the standard

## deployment process of 'ebx' web application in the target operational

environment

## (see the deployment descriptor 'web.xml' of 'ebx' web application).

#ebx.persistence.url=jdbc:h2:tcp://127.0.0.1/ebxdb

#ebx.persistence.user=xxxxxxxxx

```

```

#ebx.persistence.password=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 SAP Hana

```

```
#####

#ebx.persistence.factory=hana

## 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:sap://127.0.0.1:39041

#ebx.persistence.driver=com.sap.db.jdbc.Driver

#ebx.persistence.user=xxxxxxxxx

#ebx.persistence.password=yyyyyyyyy

#####

## 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;datasname=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

このパラメーターは、Java ディレクトリのファクトリークラス名を指定します。デフォルトの EBX® ディレクトリを使用しない場合にのみ定義する必要があります。

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`

```

#####

## Specifies the Java directory factory class name.

```

```
## Value must be the fully qualified name of the Java class.

## The class must extend
com.orchestranetworks.service.directory.DirectoryFactory.

#####

#ebx.directory.factory=xxx.yyy.DirectoryFactoryImpl
```

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
```

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	対象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。

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.PatternLayout

#-----

ebx.log4j.category.log.monitoring=INFO, ebxFile:monitoring

ebx.log4j.category.log.dataServices=INFO, ebxFile:dataServices

ebx.log4j.category.log.d3=INFO, ebxFile:d3

ebx.log4j.category.log.request=INFO, ebxFile:request

ebx.log4j.category.log.restServices=INFO, ebxFile:dataServices

ebx.log4j.category.log.audit=INFO, ebxFile:audit

```

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 `xxxxxxx` 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).

## If SSL is activated, a SSL factory and a SSL provider are required.

#ebx.mail.smtp.ssl.activate=true

#ebx.mail.smtp.ssl.provider=com.sun.net.ssl.internal.ssl.Provider

#ebx.mail.smtp.ssl.factory=javax.net.ssl.SSLSocketFactory
```


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
```

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
```

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 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,

#     "importDomainDefinition": true

#   }

# }

# ]

#}

# [forceUpdateOnIdenticalComponent] boolean, when true, forces the update
of a component even

# if it is the same in the repository.

# [importDomainDefinition] 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ーからの 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 認証ヘッダーの上書き防止のため、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

##

## 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.useHttpsSettings=false


#ebx.servlet.https.host=

#ebx.servlet.https.port=

ebx.servlet.https.path=ebx/

#ebx.servlet.https.ui.path=ebx-ui/


#####

## 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。

受信リクエストで使用されない場合でも SSL セキュリティの使用を強制するには、

`ebx.servlet.http.useHttpsSettings` および/または

`ebx.externalResources.http.useHttpsSettings` プロパティを `true` に設定する必要があります。デフォルト値は `false` です。

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.useHttpsSettings=false


#ebx.servlet.https.host=

#ebx.servlet.https.port=

ebx.servlet.https.path=servletAlias
```

```

ebx.servlet.https.ui.path=uiServletAlias

#####

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.useHttpsSettings=false
```

```
ebx.servlet.https.host=reverseDomain
```

```
#ebx.servlet.https.port=
```

```
ebx.servlet.https.path=ebx/
```

```
#ebx.servlet.https.ui.path=ebx-ui/
```

```
#####
```

```
## 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。

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](#)

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 `true` 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ソートす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.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/

#####

## 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/*
```

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](#)

初期化と初回起動アシスタント

提供されるもの（ソフトウェアおよびドキュメント）は、[TIBCO eDelivery](#)にあります（eDeliveryにアクセスするにはアカウントが必須です。[サポートチーム](#)に連絡してリクエストしてください）。

TIBCO EBX® Configuration Assistantは、EBX®リポジトリの初期構成を支援します。起動時にEBX®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デプロイされていることを確認してください。[Java EEデプロイメント](#)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する必要があります。TIBCO EBX®メイン構成ファイルを参照してください。

- 構成手順
 - 使用許諾契約の検証
 - 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

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. 使用許諾契約の検証
2. リポジトリの構成
3. デフォルトのユーザーおよびロールディレクトリでのユーザー定義（カスタムディレクトリが定義されていない場合）
4. 入力した情報の検証
5. EBX®リポジトリのインストール

使用許諾契約の検証

構成を続行するには、製品使用許諾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いくつかのプロパティが表示されます。このステップでは、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ID (`repositoryId`)

（企業の範囲内で）リポジトリを一意に識別する必要があります。識別子は48ビット（6バイト）の長さで、通常は12桁の16進数で表されます。この情報は、リポジトリに作成されたエンティティのUniversally Unique

	Identifier (UUID) 、および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で指定されているように、「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**<戻る**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フィックス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である必要があります。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® アドオンのバージョン管理およびパッケージ化ガイド』*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。

新しい EBX® アドオンをリポジトリに登録するには、以下の手順に従います。

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、 ボタンをクリックして新しいエントリを作成します。
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ず、アドオンが再度登録されるまで UI で使用できます。

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。

EBX® Container Edition

- [イメージの構築](#)
- [イメージの実行](#)

- イメージのカスタマイズ

イメージの構築

- 概要
- 要件
- イメージの構築
 - 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
 - Intel または AMD 以外のアーキテクチャ

概要

TIBCO EBX® Container Edition イメージは Linux ベースであり、次のものが含まれます。

- Apache Tomcat® 9.0 Application Server
- Java JDK 17

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構築できます。

- **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
- **プロセッサの種類**：Intel または AMD 64 ビット
- **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

イメージの構築

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\6.1.2\addon\6.1.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\6.1.2\addon\6.1.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\6.1.2\addon\6.1.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。

指示に従い、イメージに追加するアドオンを選択します。

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:6.1.2-dama-dint-dmdv-dpra-mame-tese-6.1.X
```

```
This image includes the following addon(s):
```

```
Digital Asset Manager (DAMA)
```

```
Data Exchange (New) (DINT)
```

```
Data Model and Data Visualization (DMDV)
```

```
Insight (New) (DPRA)
```

```
Match and Merge (MAME)
```


Information Search (TESE)

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.

```
*****
*****
```

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\6.1.2\addon\6.1.X\container\edition.zip** を解凍します。
- **ebx-ce-installer-batch.sh** ファイルの格納先フォルダーでターミナルを開きます。
- コマンド **./ebx-ce-installer-batch.sh** を実行します。

Batch script to build image with optionally addon(s).

```
-a                add all addons

-t <tag_file>    create a file named tag_file including the image tag
built

--help          display this help and exit

[addon]...      add specific addons (separate by a space)
```

```
for sample: 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\6.1.2\addon\6.1.X\container\edition.zip** を解凍します。
- **ebx-ce-installer-batch.bat** ファイルの格納先フォルダーで Windows PowerShell を開きます。
- コマンド **ebx-ce-installer-batch.bat** を実行します。

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、タグ **latest** を使用しないことをお勧めします。インストーラーは、選択したアドオンに応じて、常に別のタグを生成します。

上記のサンプルの場合、次のアドオンが選択されているため、タグは **6.1.2-dama-dint-dmdv-dpra-mame-tese-6.1.X** です。

- Digital Asset Manager (DAMA)
- Data Exchange (New) (DINT)
- Data Model and Data Visualization (DMDV)
- Insight (New) (DPRA)
- Match and Merge (MAME)
- Information Search (TESE)

次に、以下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:6.1.2-dama-dint-dmdv-dpra-mame-tese-6.1.X
```

アドオンが選択されていない場合、タグは **6.1.2** です。

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:6.1.2
```

イメージの共有

イメージを共有する手順は、ユーザーの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:6.1.2 myregistry:5000/ebx:6.1.2
```

```
docker push myregistry:5000/ebx:6.1.2
```

Intel または AMD 以外のアーキテクチャ

インストーラーは、Intel または AMD 64 ビットシステムでテストされています。これらのプラットフォームでは、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。

別のプロセッサを使用するシステムで実行した場合でも、インストーラーが成功する場合があります。その場合、生成されたイメージアーキテクチャは、そのプロセッサタイプの 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アーキテクチャは **linux/arm64** になります。

注意

Intel または AMD 以外のシステムでのイメージの生成は**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
 - URL 構成
 - データベース接続
 - 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:6.1.2
```

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>" \  
  
ebx:6.1.2
```

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

サポートされているブラウザの詳細については、[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。

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>" \
```

ebx:6.1.2

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	着信要求が 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」が想定されます。「false」の場合、プロトコル「HTTP」が想定されます。
EBX_HOSTNAME	着信 HTTP(S) 要求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(S) 要求によって指定されたポート番号	EBX® サーバーのポート番号
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 <i>/</i> の場合を除き、値には先頭に <i>/</i> を付ける必要があります、末尾に <i>/</i> 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

名称	デフォルト	説明
		は https://host_name です。 EBX_URL_DEFAULT が指定されておらず、 EBX_HOSTNAME が指定されている場合、デフォルトは次の仮定で計算されます。 - EBX_IS_SECURED が指定されていない場合、「false」が想定されます。 - EBX_PORT が指定されていない場合、 EBX_IS_SECURED が true の場合は 443 が想定され、そうでない場合は 80 が想定されます。

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 jdbc:postgresql://<database_host>:<database_port_number>:<database_name> です。
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、および Microsoft SQL Server 専用の JDBC ドライバーが含まれています。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。

例

Postgres データベースに接続する 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_DB_FACTORY=postgresql" \
  -e "EBX_DB_URL=jdbc:postgresql://<server_name>:5432/<database_name>" \
  -e "EBX_DB_USER=<user_name>" \
  -e "EBX_DB_PASSWORD=<user_name>" \
  ebx:6.1.2
```

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

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_CONNECTION_TIMEOUT	ebx.mail.smtp.connectionTimeout	60000	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
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** の場合はアクティブ化され、値が **false** の場合はステージング機能が非アクティブ化されます。

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**
- **/my_custom/conf/ebx-container.properties**

ebx-default.properties ファイル

/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties ファイルは、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

/opt/ebx/conf/ebx-container.properties ファイル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、**/my_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:6.1.2
```

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
/ebx/data	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。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。

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

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:6.1.2

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 ファイルで次の手順を実行します。

- 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:6.1.2

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、標準の Java EE Web アプリケーションであり、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ます。

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-9.0-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:6.1.2

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

新しいロケールを追加するには、言語パックを含む jar ファイルが必要であり、ロケールを ebx 構成に追加する必要があります。

以下は、サンプルの Docker ファイルを示しています。

```
FROM ebx:6.1.2

USER root
```

```

COPY "<path_to_lib>" "/opt/ebx/lib"

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=es" >> "/opt/ebx/webapps/ebx/WEB-INF/ebx-
default.properties"

USER ebx

```

この例では、**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、lib がコピーされ、ロケールが「es」に設定されます。

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:6.1.2

USER root

ADD \

      https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc11/21.8.0.0/
      ojdbc11-21.8.0.0.jar \

```

```
"/opt/ebx/lib/"
```

```
RUN chmod +r "/opt/ebx/lib/ojdbc11-21.8.0.0.jar"
```

```
USER ebx
```

詳細については、データベースドライバ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
- ワークフロー管理
- タスクスケジューラ
- 監査証跡
- コラボレーション機能 (チームアップ)
- その他

リポジトリ管理

- テクニカルアーキテクチャ
 - 概要
 - データベースアクセスとユーザー権限のルール
 - リポジトリごとに 1 つの 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

- 自動インクリメント
- 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
- 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
 - データスペースのアクセス許可
 - リポジトリの履歴

テクニカルアーキテクチャ

概要

TIBCO EBX® テクニカル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ています。このリポジトリは、構成されたデータソースを介してアクセスされる、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永続化されます。
- リポジトリは、常に複数の 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もできます。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ための代替手段は、分散データ配信 (D3) 環境を展開することです。
- 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 (個別の JVM で使用) をサポートできます。次に、プロパティ `ebx.persistence.table.prefix` を使用して個別の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。

構成済みデータソースで指定されたデータベースユーザーは、テーブル、インデックス、およびシーケンスに対する「作成/変更」権限を持っている必要があります。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 1 つの JVM

リポジトリを複数の JVM で共有することはできません。このような状況が発生した場合、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、この制限を適用するための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、リポジトリは最初に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。リポジトリを開始した後、JVM は定期的にリポジトリの所有権を保持していることを確認します。

これらのチェックは、リレーショナルデータベースのテクニカルテーブルに繰り返しタグを付けることによって実行されます。アプリケーションサーバーのシャットダウンコマンドは、このテクニカルテーブルのタグが削除されていることを確認します。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っている可能性があります。これが発生した場合、サーバーは再起動時にさらに数秒待機して、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期間を回避するために、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内で常に 1 つのサーバーのみがアクティブであ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。これを確実に行うには、メインサーバーがプロパティ `ebx.repository.ownership.mode=failovermain` を宣言する必要があります。単一サーバーの場合と同様に、メインサーバーは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引き続き起動できますが、リポジトリにアクセスすることはできません。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。さらに、両方のサーバーが `ebx.repository.directory` に同じ値を定義し、この値で定義されたディレクトリを共有するために必要です。(これは特に、Lucene インデックスを共有できるようにするためです。つまり、フェールオーバーサーバーの起動時にオンデマンドで再構築されません。) 起動すると、バックアップサーバーが接続ログに登録されます。そのステータスは、以下の[リポジトリのステータス情報とログ](#)のセクションで説明されているように、JavaAPI または HTTP リクエストを介して取得できます。

バックアップサーバーをアクティブ化し、リポジトリの排他的所有権をサーバーに転送するには、HTTP リクエストによって、または Java API を使用して特定のリクエスト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` が含まれている必要があり、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` に対して宣言されている値と同じである必要があります。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する必要があります。

```
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、以下が適用されます。バックアップサーバーはデータベース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すべて終了します)。メインサーバーが所有権を返した後でのみ、バックアップサーバーはリポジトリの使用を開始できます。

リポジトリのステータス情報とログ

リポジトリへのすべての試行された Java プロセス接続のログは、[\[履歴とログ\]](#) > の下の エリアにあります。

リポジトリのステータスは、`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、リポジトリステータス情報を取得することもできます。

state	所有権登録に関するリポジトリの状態。 • D : Java プロセスは停止しています。 • O : Java プロセスはデータベースの独占的な所有権を持っています。 • 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すが、リポジト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ましたが、別のプロセスがデータベースを保持しているため失敗しました。

heart_beat_count	データベースに関連付けられてからリポジトリが接続した回数。
info	リポジト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、明示的な警告なしに将来変更される可能性があります。

自動インクリメント

EBX® ユーザーインターフェイスの エリアから、いくつかのテクニカル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しない限り、コンテンツを手動で編集しないでください。これらのテクニカルテーブルには次のものがあります。

自動インクリメント	リポジト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。

リポジトリ管理

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により、リポジトリのインストールまたはアップグレードが自動的に行われます。

データベース間の移行

EBX® は、リポジトリの全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。この移行を操作するには、次のガイドラインに従う必要があります。

- ソースリポジトリをシャットダウンする必要があります：EBX® サーバプロセスがアクセスすることはできません。この要件に厳密に準拠していないと、ターゲットリポジトリが破損する可能性があります。
- 新しい EBX® サーバプロセスをターゲットリポジトリで起動する必要があります。ターゲットリポジトリは空である必要があります。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 ebx.migration.source.properties

(ソースリポジトリを指定する EBX® プロパティの場所) も指定する必要があります (ターゲットとソースの間に個別の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、ターゲットリポジトリにアクセスする EBX® サーバープロセスを強制的に再起動します。

制限：

- 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。
- 結果として、データモデル内のレプリケート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合された名前と一致しません。このデータモデルの最初の再コンパイルでは、この不整合を修正する必要があります。
- 数値型の表現が異なるため、ターゲットデータベースエンジンがソースよりも精度が低い場合、`xs:decimal` 型の値が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。

リポジトリのバックアップ

EBX® リポジト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の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 (構成を参照)。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:

このディレクトリへのアクセスは、**セキュリティのベストプラクティス**で規定されているように慎重に保護する必要があります。また、手動でこのディレクトリを作成する場合は、

EBX® プロセスが読み書き可能なアクセス権を持ってい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ていないため、管理者はこのディレクトリをクリーンな状態に保つ責任があります。

注意:

2 つの EBX® 環境間でのファイルの転送は、FTP やネットワーク共有による単純なファイルコピーなどのツール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	会社の範囲内の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、リポジトリに作成されたエンティティの UUID (Universally Unique Identifiers)、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 で指定されているように、「UUID ノード」の部分として機能します。
repository label	リポジト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ーなラベルを提供します (例: 「実稼働環境」)。
store format	構造の現在のバージョンを含む、基盤となる永続化システムを識別します。

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

一部のエンティティは、EBX® の実行中に蓄積されます。

注意:

これらのエンティティを監視および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

リポジトリの永続性データソースは、RDBMS 監視を通じて監視する必要があります。

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の最新の統計を計算している必要があります。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、テーブルが短期間に大幅に変更された場合 (たとえば、インポートによって多くのレコードが作成された場合)、統計を明示的に更新する必要があります。

履歴テーブル：UI への影響

履歴テーブルの場合、一部の UI コンポーネントは統計を使用して動作を調整し、ユーザーがコストのかかるリクエストを不本意に実行するのを防ぎます。

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、実際にはそうではない場合でも、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

リポジトリからのデータスペース、スナップショット、および履歴の完全なクリーンアップには、いくつかの段階が含まれます。

1. キャッシュを最小サイズに保つために、未使用のデータスペースとスナップショットを閉じます。
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、および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すべて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 エリアを使用します。
- エリアの の テーブルから、ワークスペースの メニューを使用します。このアクションは、テーブルのフィルタリングされたビューで使用することができます。
- Java API を介して、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、データはリポジトリから安全に削除できます。

注意:

閉じたデータスペースとスナップショットは、 の下の エリア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、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、データスペースの削除は、必ずしもその履歴の削除を意味するわけではありません。2つの操作は独立しており、異なる時間に実行できます。

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削除が実行されるまで、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、リポジトリ全体の削除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削除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削除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。削除操作が実行されると、後続の履歴操作は含まれません。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:

今後、削除日を設定することはできなくなります。したがって、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、および履歴の削除は、さまざまな方法で実行できます。

- ワークスペースのメニューボタンを使用して、エリアの の下にある テーブルから。このアクションは、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、より具体的にはメソッド `Repository.deleteHome` および `RepositoryPurge.markHomeForHistoryPurge` を使用します。
- データサービスの 操作 (パラメーター `deleteDataOnClose` および `deleteHistoryOnClose` を使用)、または 操作 (パラメーター `deleteDataOnMerge` および `deleteHistoryOnMerge` を使用) の終了時。

データスペース、スナップショット、または履歴の削除後の残りのエンティティの削除

アイテムが削除されると、削除を実行して、その時点までに実行されたすべての削除から残りのデータをクリーンアップできます。削除は、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 エリアで > 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を使用します。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。

削除プロセスは、ディレクトリ `${ebx.repository.directory}/db.purge/` に記録されます。

他のリポジトリエンティティのクリーンアップ

次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の実行レポート

タスクスケジューラの実行レポートは、エリアの セクションの テーブルに保持されます。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。

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* 管理セクションで保持されます。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。

履歴をクリーンアップする手順は次のとおりです

- プロセスの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 エリアで > またはナビゲーションペインで > 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クリーンアップします (ワークフロー履歴の エリアで > またはナビゲーションペインで > を選択することで実行できます)。
- 「標準 EBX® ページ」を使用して、ワークフロー履歴の残りのエンティティを削除します。

以下も参照してください。:

- [標準 EBX® ページ](#)

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、次のディレクトリのディスク使用量とディスク可用性を管理者が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: インデックスには大量のディスクスペースが必要になる場合があります。これらは、EBX® が正しく機能するために重要です。名目上の使用法では、それらを直接削除または変更しないでください。ただし、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。

- リポジトリが最初から再作成され、ディレクトリ `${ebx.repository.directory}/` が保持されている場合。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矛盾するようになった場合 (これはまれに不具合が発生する可能性があります)。

削除後、インデックスのコンテンツは、リポジトリのコンテンツから派生して、テーブルごとに遅延して再計算されます。削除はインデックスのルートフォルダーで行う必要があります。単一のディレクトリが下位レベルで削除されると、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、通常は回避する必要があります。

- **XML 監査証拠:** `${ebx.repository.directory}/History/`
- **アーカイブ:** `${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、(デフォルト設定とは逆に) 完全な更新の詳細をアクティブにして大規模なトランザクションを実行すると、必要なディスク容量が増える可能性があります。

注意:

データサービスの `getChanges` 操作でのページ付けには、一時ディレクトリで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 エリアから を選択することにより、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、リポジトリ内の既存のすべてのデータスペースとスナップショットが一覧表示されます。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、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、開いたデータスペースまたは閉じたデータスペース、関連する履歴、スナップショットを削除および削除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

データスペースのアクセス許可

この表は、リポジトリ内のすべてのデータスペースで定義されているすべての既存の権限ルールを示しています。権限ルールを表示し、それらの情報を変更できます。

以下も参照してください。:

- データスペースのアクセス許可

リポジトリの履歴

テーブル「削除されたデータスペース/スナップショット」には、リポジトリからすでに削除されているすべてのデータスペースが一覧表示されます。

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。

[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
 - 表示権限
- ユーザーセッション管理
- ユーザー補助モード
 - 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。

「表示領域」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

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。

定義されている場合

- 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。

この構成方法は非推奨であり、[URL コンピューティング](#)に置き換える必要があります。EBX® メイン構成ファイルを構成した後、これらの構成を設定解除する必要があります。

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

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イス構成

アクティベーション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	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。
デフォルトの選択	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

可用性ステータス	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。
使用不可メッセージ	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。

独自のセッション制御	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ー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

表示するテーブル列の最大数	ネットワークとブラウザー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。
デフォルトの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 エディターの幅	フォーム内の HTML エディターの幅。
フォーム：HTML エディターの高さ	フォーム内の HTML エディターの高さ。
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要求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。
ヘッダー内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。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。
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。

アクティベーション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	この項目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 ピクセルである必要があります。 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	[ビュー] メニューからこのビューを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

権限	説明	デフォルト値
		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ビューを利用できるようにします。	ユーザーがビルトインの管理者である場合、デフォルト値はです。それ以外の場合はです。

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。

ユーザー補助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。

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、次の概念に精通している必要があります。

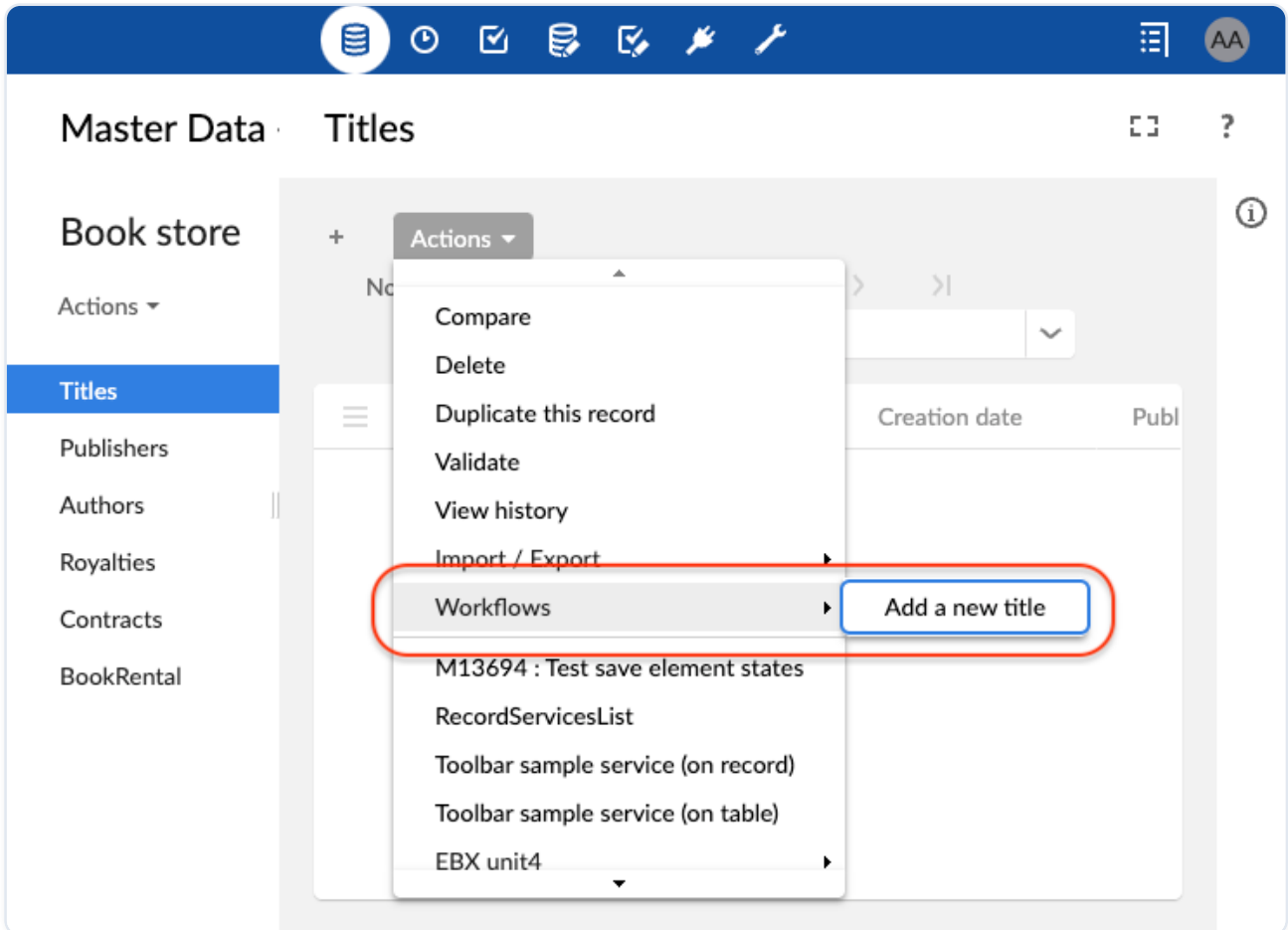
概念	説明/リンク
ワークフローモデル	ドキュメントを参照
データコンテキスト	ドキュメントを参照
ワークフローパブリケーション	ドキュメントを参照
パブリケーション名	ワークフローパブリケーションの一意の識別子
データワークフロー	ドキュメントを参照
データセクション	EBX® のデータセットとテーブルを表示する データユーザーインターフェイス です。メインヘッダーからアクセスできます。
ワークフローランチャーデータセット	これには、** Launchers** と Activations という 2 つのテーブルが含まれています。このデータセットを使用すると、ユーザーはワークフローパブリケ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

概念	説明/リンク
	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。
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、 <code>\$(dataspace)</code> 、 <code>\$(dataset)</code> 、 <code>\$(xpath(table))</code> 、および <code>\$(xpath(record))</code> です。たとえば、値 <code>\$(xpath(table))</code> 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。

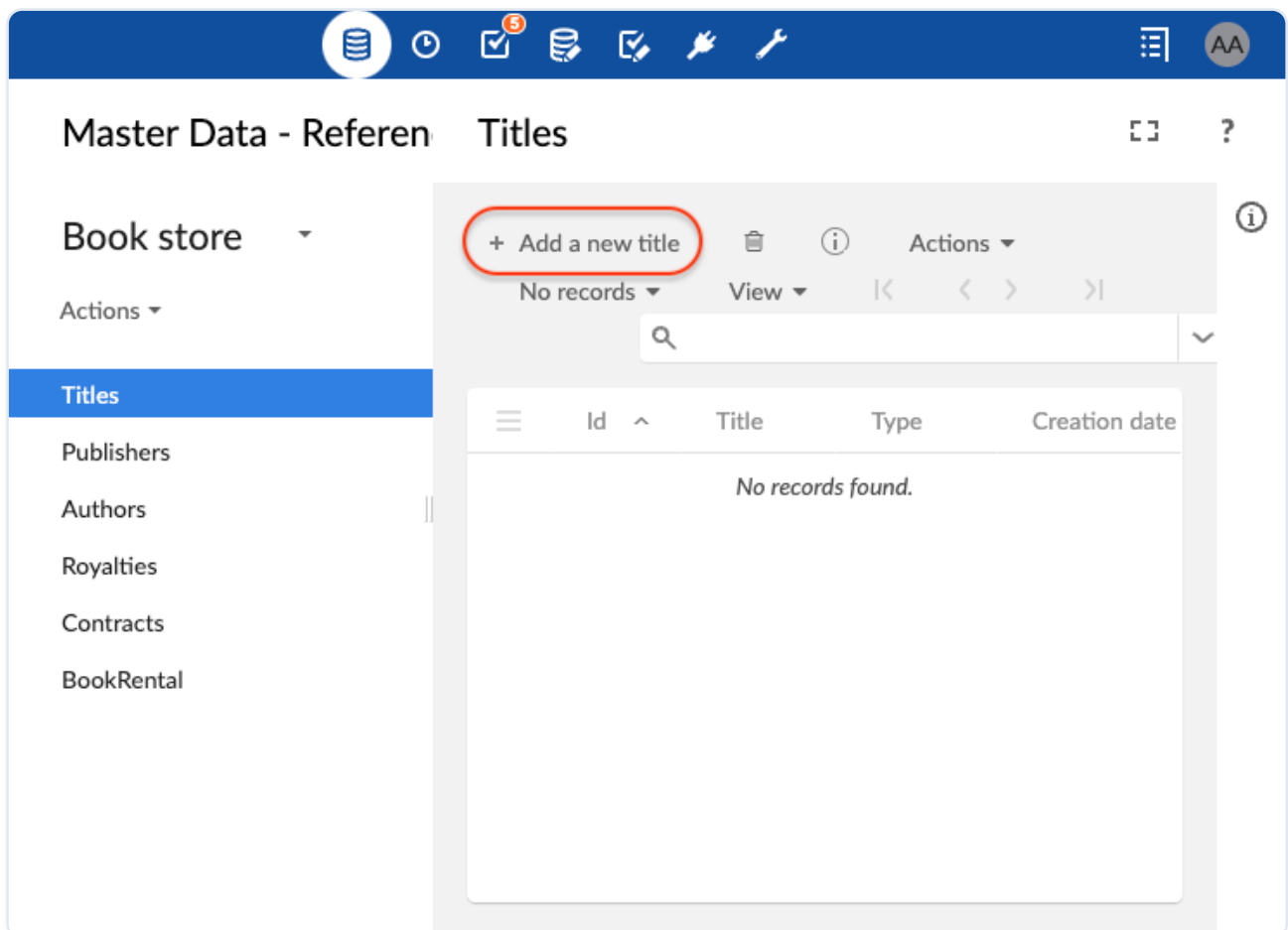
データセクションのワークフローランチャー

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

で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、次の優先順位で計算されます。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)}` にマッピング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 has the following fields:

- 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)}

Buttons at the bottom include '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

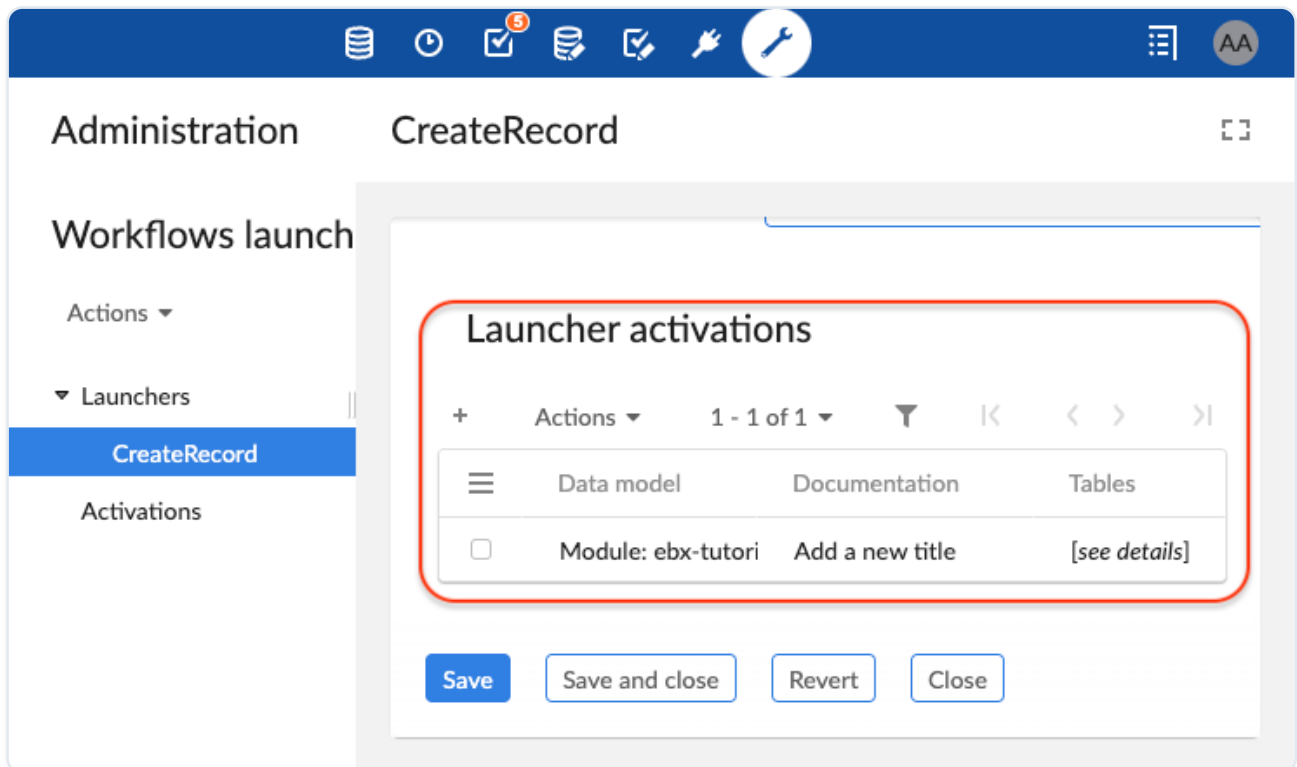
フィールド名	説明
	ローパブリケーションを定義します。リストで使えるようにするには、ワークフローモデルを最初に定義して公開する必要があります。
データコンテキスト変数	このフィールドには、ワークフローモデルのデータコンテキスト構成フェーズで定義された入力変数のリストが含まれます。各行は、ラベル、値、およびデフォルト値と上書き値を切り替えるためのトグルボタンで構成されます (ランチャーフォームのデータコンテキストに関するスクリーンショットを参照)。ラベルは、ワークフローモデルのデータコンテキストでユーザーによって設定された変数の名前です。 デフォルトでは、各変数の値はデータコンテキストで設定さ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、`\${dataspace}` 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、`\${dataspace}` 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]

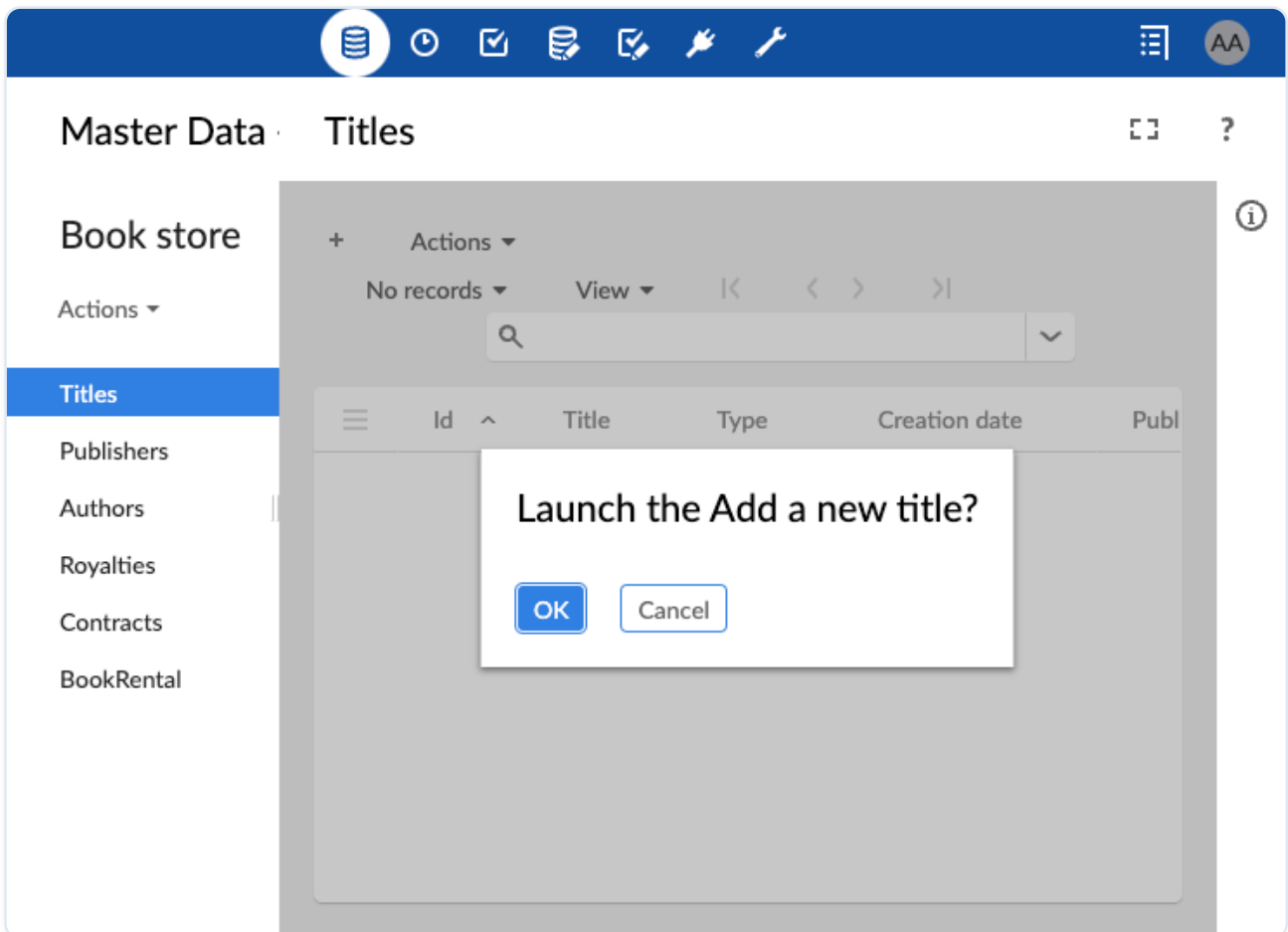
フィールド名	説明
	フィールドに含まれるすべてのテーブルを表示します。
プロファイル	以前に選択したテーブルでワークフローランチャーを表示および起動できるプロファイルのリスト。
確認	このフィールドでは、データワークフローの起動を確認するダイアログボックスを表示するかどうかを指定できます (起動確認メッセージに関するスクリーンショット を参照)。デフォルトでは、この機能は無効になっています。ダイアログボックスを表示するには、値のチェックをオンにする必要があります (以下のスクリーンショットを参照)。
ドキュメント	ドキュメントはラベルと説明で構成されます。ラベルと説明のデフォルト値は、ランチ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。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 TIBCO EBX Administration interface. On the left, under 'Workflows launchers', the 'Activations' section is expanded, and the 'New record' button is highlighted. The main area is titled 'Add new activation'. It contains several form fields: 'Launcher' is set to 'CreateRecord'; 'Data model' is 'Module: ebx-tutorial, path: /WEB-INF/ebx/scher'; 'Tables' has a list with 'Titles'; 'Profiles' has a list with '[all profiles]'; 'Confirmation' has radio buttons for 'Enable' (selected) and 'Disable'; 'Documentation' is set to 'English (United States)' with a sub-field 'Add a new title'. At the bottom 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。

- [1 番目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)。
- [1 番目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トレイ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' menu item is selected. The main area displays a form titled 'Create a record' with a sub-header 'Create a new book'. The form includes fields for 'Id' (A000001), 'Title', 'Type' (Unknown), 'Creation date' (date picker), 'Publisher' ([not defined]), and 'Author' ([not defined]). There are also buttons for 'Save' and 'Save and close'.

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[カスタムメニューグループ](#)の[アクションメニュー項目](#)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)。

Data models : |bars : □ Create a record [@creation]

BookStore ▾

Actions ▾ **Publish**

- Configuration
- Data structure
- Simple data types
- Complex data types
- ▾ Extensions
- Toolbars**
- Entity mappings
- User services
- Replications
- Ajax components

Toolbar item type* Action button

Parent element TitlesTop ↗

Service* Target * ☒ Current context ☐ Web component

Service Workflow Launcher [@workflowLauncher]

Launcher CreateRecord

Label and description ⓘ The default value is inherited from the workflow launcher. The value displayed on the toolbar is computed as follows : the label of the launcher is used if available, otherwise the label of the service is used. If the latter is not available, then the label of the launcher is used.

English (United States) ↕

Create a new record

Save Save and close Revert Close

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、>>> 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)。

Target * ☒ Current context ☐ Web component

Service Workflow Launcher [@workflowLauncher]

Launcher CreateRecord

Activations Table ↗

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構成フォームの「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、次の優先順位で計算されます。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

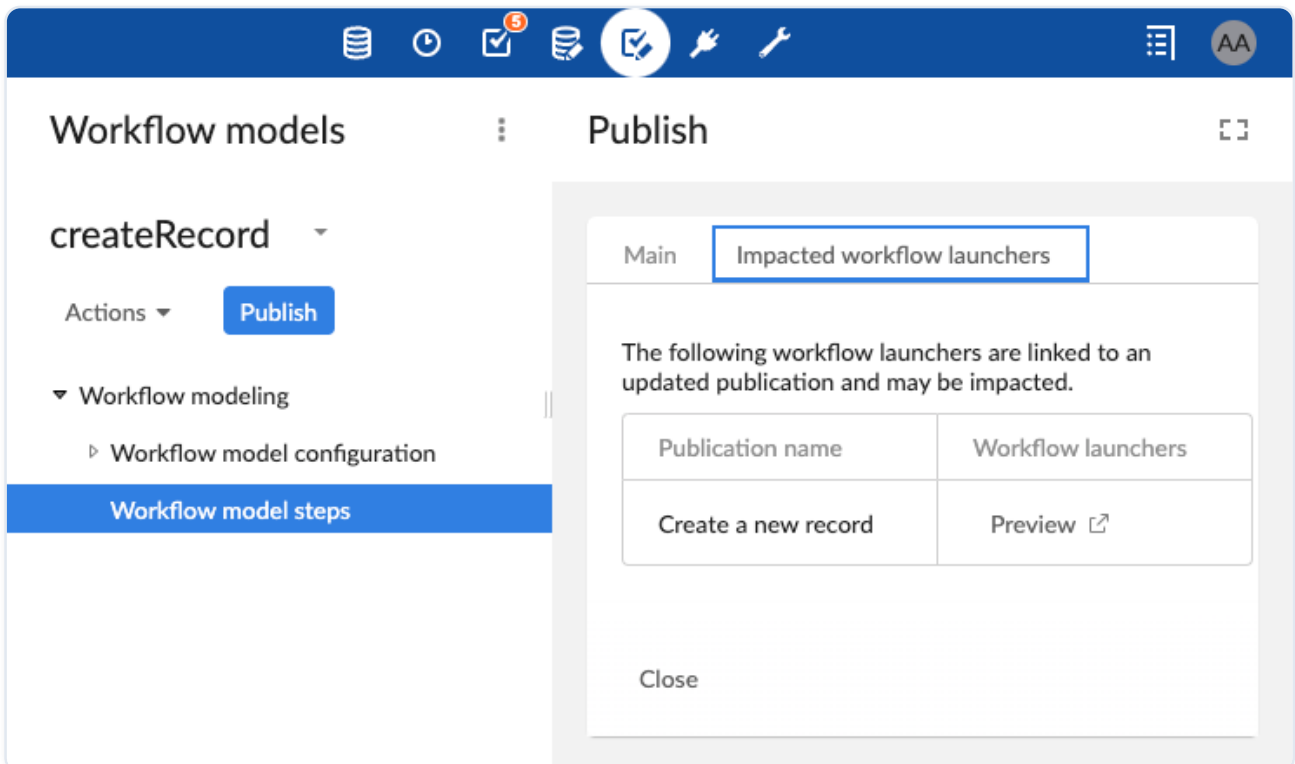
ワークフローランチャーサービスは、テーブルビュー上部、テーブル行、レコードフォームビュー上部、または階層ビュー上部に表示されるツールバーでのみ使用できます。現在まで、このサービスは、関連付けテーブルの上または関連付けテーブルの行に表示されるツールバーでは使用できません。

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。これらのランチャーで 사용되는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です。

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 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

Information message
 ⓘ 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	\${dataset}	
Dataspace	\${dataspace}	
Record	[not defined]	
Table	\${xpath(table)}	

The new input data context

ユーザーとロールのディレクトリ

- 概要
- 概念
 - ポリシー
 - ユーザー
 - ロール
- デフォルトディレクトリ
 - ディレクトリコンテンツ
 - パスワード回復手順
 - 管理者の回復手順
- カスタムディレクトリ

概要

TIBCO EBX®は、ユーザー認証とユーザーの役割（ロール）定義にディレクトリを使用します。

デフォルトのディレクトリが提供され、EBX®リポジトリに統合されます。「ディレクトリ」管理セクションでは、接続できるユーザーとその役割を定義できます。

別のタイプのエンタープライズディレクトリを統合することもできます。

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
- カスタムディレクトリ

概念

EBX®では、ユーザーは複数の役割のメンバーになることができ、役割は複数のユーザーによって共有されることがあります。さらに、ある役割を別の役割に含めることができます。プロファイルという一般的な用語は、ユーザーまたは役割のいずれかを表すために使用されます。

EBX®は、ディレクトリ定義の役割に加えて、次のビルトインの役割を提供します。

ROLE	定義
Profile.ADMINISTRATOR	ビルトインの管理者ロール。一般的な管理タスクを実行できます。
Profile.READ_ONLY	ビルトインの読み取り専用ロール。読み取り専用ロールに関連付けられているユーザーは、EBX®リポジトリのみを表示でき、リポジトリで変更を実行する権限はありません。
Profile.OWNER	動的なビルトインの所有者の役割。この役割は、現在のエレメントに応じて動的にチェックされます。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に属している場合にのみアクティブになります。
Profile.EVERYONE	すべてのユーザーがこの役割に属します。

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。

注意:

ユーザーとビルトインの役割OWNERおよびEVERYONEの間の関連付けは、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することはできません。

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](#)
- [役割](#)
- [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

これらのプロパティは、ユーザーとロールディレクトリのポリシーを構成します。たとえば、ユーザーが自分のプロフィールを編集できるかどうかなどです。

ユーザー

この表には、内部ディレクトリで定義されているすべてのユーザーが一覧表示されます。そこから新しいユーザーを追加できます。

役割（ロール）

この表には、内部ディレクトリで定義されているすべてのユーザーが一覧表示されます。このテーブルで新しい役割を作成できます。

デフォルトディレクトリ

ディレクトリコンテンツ

デフォルトのディレクトリは、「管理」領域のデータセット「ディレクトリ」で表されます。

このデータセットには、ユーザーとロールのテーブル、およびユーザーのロールテーブル、ロールの包含テーブル、敬称テーブルが含まれています。

注意:

役割の包含サイクルが検出された場合、権限の解決時に役割の包含は無視されます。サイクル検出のためにディレクトリ検証レポートを更新して確認します。

注意:

ユーザーの役割、役割の包含、および敬称の表は、**デフォルトで非表示になっています**。

定義されたポリシーに応じて、ユーザーは、ディレクトリデータセットで定義されたアクセス許可に関係なく、自分のアカウントに関連する情報を変更できます。

注意:

デフォルトのディレクトリを削除または複製することはできません。

パスワード回復手順

デフォルトのディレクトリでは、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。したがって、紛失したパスワードを取得することはできません。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。
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。

デフォルトでは、最初のオプションが使用されます。2番目のオプションをアクティブにするには、`TIBCO EBX®メイン構成ファイル` でプロパティ `ebx.password.remind.auto=true` を指定します。

注意:

セキュリティ上の理由から、管理者プロファイルではパスワード回復手順を使用できません。必要に応じて、代わりに管理者の回復手順を使用してください。

管理者の回復手順

管理者の「ログイン/パスワード」資格情報がすべて失われた場合は、特別な手順に従う必要があります。特定のディレクトリクラスは、ログイン「admin」とパスワード「admin」で管理者ユーザーを再定義します。

この手順をアクティブにするには：

- `TIBCO EBX®メイン構成ファイル` で次のプロパティを指定します。

```
ebx.directory.factory=
com.orchestranetworks.service.directory.DirectoryDefaultRecoverFactory
```

- EBX®を起動し、手順が完了するまで待ちます。
- `' ebx.directory.factory '` プロパティをリセットします。
- 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、ユーザーの認証は拒否されます。

カスタムディレクトリ

デフォルトのディレクトリの代わりに、特定の会社のディレクトリを統合することができます。たとえば、LDAP インスタンス、リレーショナル データベース、または EBX®にインスタンス化された特定のディレクトリ モデルです。デフォルトのログイン ページを特定の企業ページに置き換えることもできます。

以下も参照してください。:

- `DirectoryFactory`
- `Directory.getUserAuthenticationURI`

データモデルの管理

- 公開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
- スキームの進化
 - データの永続性への影響
 - 副次機能への影響

公開とバージョンの管理

データモデルの公開とバージョンに関連する技術データには、管理者が管理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。

- **公開**。リポジトリで利用可能な公開を保存します。
- **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格納します。

これらのテーブルは読み取り専用ですが、公開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：****公開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が使用できなくなります。したがって、この技術データは注意して削除する必要があります。

この技術データを他のTIBCO EBX®リポジトリに拡散して、EBX®リポジトリからアーカイブをエクスポートし、別のリポジトリにインポートすることができます。これは、データモデルの進化を他のリポジトリに伝播す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データモデルは、ファイルシステムのディレクトリ `${ebx.repository.directory}/schemas/` にデータモデル名（データモデル名が *Product*であれば *product.xsd* など）を付けて生成されていました。5.2.0バージョン以降、この種のデータモデルは、公開を通じてEBX®内で完全に管理されるようになりました。つまり、既存のデータモデルを再公開すると、それが公開として移行され、リンクされたデータセットが新しい埋め込みデータモデルにリダイレクトされます。また、`${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、接尾辞が`toDelete`になります。これは、そのドキュメントがもう使われないので安全に削除できるようになったことを意味します。

スキーマの進化

データモデルの変更が管理側に与える影響を評価することが重要です。以下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連付けることができます。次のリンクは、データモデルの進化が永続性レベルでどのように管理されるかを説明しています：
[データベース内のマスターテーブルのページ](#)。

副次的機能への影響

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。

「管理」セクションでは、これらのコンポーネント（ビューなど）の一部を管理する可能性があります。アーカイブや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です。

データベースマッピング管理

- 概要
- データベース列の名前の変更
- データベース列の削除
- データベース内マスターテーブルの名前の変更
- データベース内補助テーブルの名前の変更
- データベース内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、**管理領域**にあります。

以下も参照してください。:

- マップされたモード
- `DatabaseMapping`

データベース列の名前の変更

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。データベース内の列の名前を変更できます。

管理者は、マップされた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。

レコードでサービスが選択されると、概要画面に選択され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。

さらに、新しい名前は有効な列識別子である必要があります。これは、基盤となるRDBMSの命名規則によって異なります。

以下も参照してください。:

- DatabaseMapping

データベース列の削除

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。マップされた構造の列をパージできます。

マップされたモードで列が無効になっている場合は、列をパージできます。

次の場合、マップされた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で互換性のない方法で変更されている（たとえば、データ型が変更されている）、または
- 定義されたマップモードが、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になっている。

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を無効にする
- 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

この動作は、集約リストでは変更され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「Tables」テーブルで確認できます。
- 一方、非アクティブ化がリストの内部ノードのみの場合、リストは `LIVING` のままになり、その子は `DISABLED IN MODEL` になります。

列は、それ自体の状態が `DISABLED IN MODEL` である場合、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パージできます。

データベース内マスターテーブルの名前の変更

この機能により、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。名前がデータモデルで指定されているため、レプリケートされたテーブルでは使用できません。

どちらの機能も、メニューのテーブルレコードで使用できます。

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。

レコードでサービスが選択されると、概要画面に選択され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に求められます。

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である必要があります。これは、基盤となるRDBMSの命名規則によって異なります。

データベース内補助テーブルの名前の変更

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である必要があります。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、テーブルレコードのメニューで使用でき、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。

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管理 > ワークフロー管理 > ワークフロー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

「Workflows」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。

「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、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

「Tokens」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

「Work items」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、「データワークフロー」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

「Waitingworkflows」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

「Comments」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

「Workflow publications」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を関連付けます。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、出版物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

メール設定

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。

- URL定義フィールドは、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、電子メール通知の送信に使用される電子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

デフォルト値のモデリング

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、「ワークフローモデリング」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、受信ボックス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。

優先順位の設定

プロパティ「デフォルトの優先度」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「通常」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「通常」の優先度を持っているように見えます。

「プロパティ」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。

一時的なタスク

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の設定

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）	受信ボックス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
- 作業項目の許諾、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、>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、対話は、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。

履歴のクリーニング

「ワークフロー」テーブルのメニューから、「日付からクリア」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「日付からクリア」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「日付からクリア」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®ページ」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ル機能を提供します。

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ルされます。

EBX® からの構成

スケジュールとタスクの宣言は、領域で を選択することによって行われます。

- **スケジュール**：「cron 式」を使用してスケジュールを定義します。
- **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。
- **スケジュールされたタスク**：タスクスケジュールのアクティブ化/非アクティブ化を含む現在のスケジュール。
- **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、スケジュール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。

フォーマット

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 日の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。
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。

式	定義
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、いくつかの事前定義されたタスクが付属しています。

スケジューラ Java API を使用して、カスタムのスケジュールされたタスクを追加できます。

スケジュールとタスクの宣言は、領域でを選択することによって行われます。

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。

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。

データスペース	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>structure-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 <pre>filters={"structured-search":["starts-with","/fi</pre>

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。
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、<code>[管理] > [メタデータ管理] > [チームアップ管理] > [タグ] > [タグ]</code> からタグを作成します。
カタログへのタグの割り当て	<code>[管理] > [メタデータ管理] > [チームアップ管理] > [タグ] > [タグ]</code> 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。

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ために、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の生成に使用するには、プロファイルをこのテーブルに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と情報を受信する機能を提供します。この機能は、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ことで構成されます。

用語

イベントブローカー	緩く結合されたイベント処理のため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で構成されます。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ているサブスクライバーの数は無制限です。
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でのクライアントの実装。

トピック

データスペースとスナップショット	データスペースおよびスナップショットでの操作に対応します。たとえば、作成、閉じる、再度開く、削除、アーカイブエクスポート、アーカイブインポート（データスペースのマージのみ）などがあります。
リポジトリ	起動やパージなど、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、「管理」領域の「イベントブローカー」の下にあります。これには、「トピック」、「サブスクライバー」、および「サブスクリプション」の3つのテーブルが含まれています。

次の操作を除いて、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、プロパティファイルで次のように定義できます。

```
# 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](#)
 - [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](#)

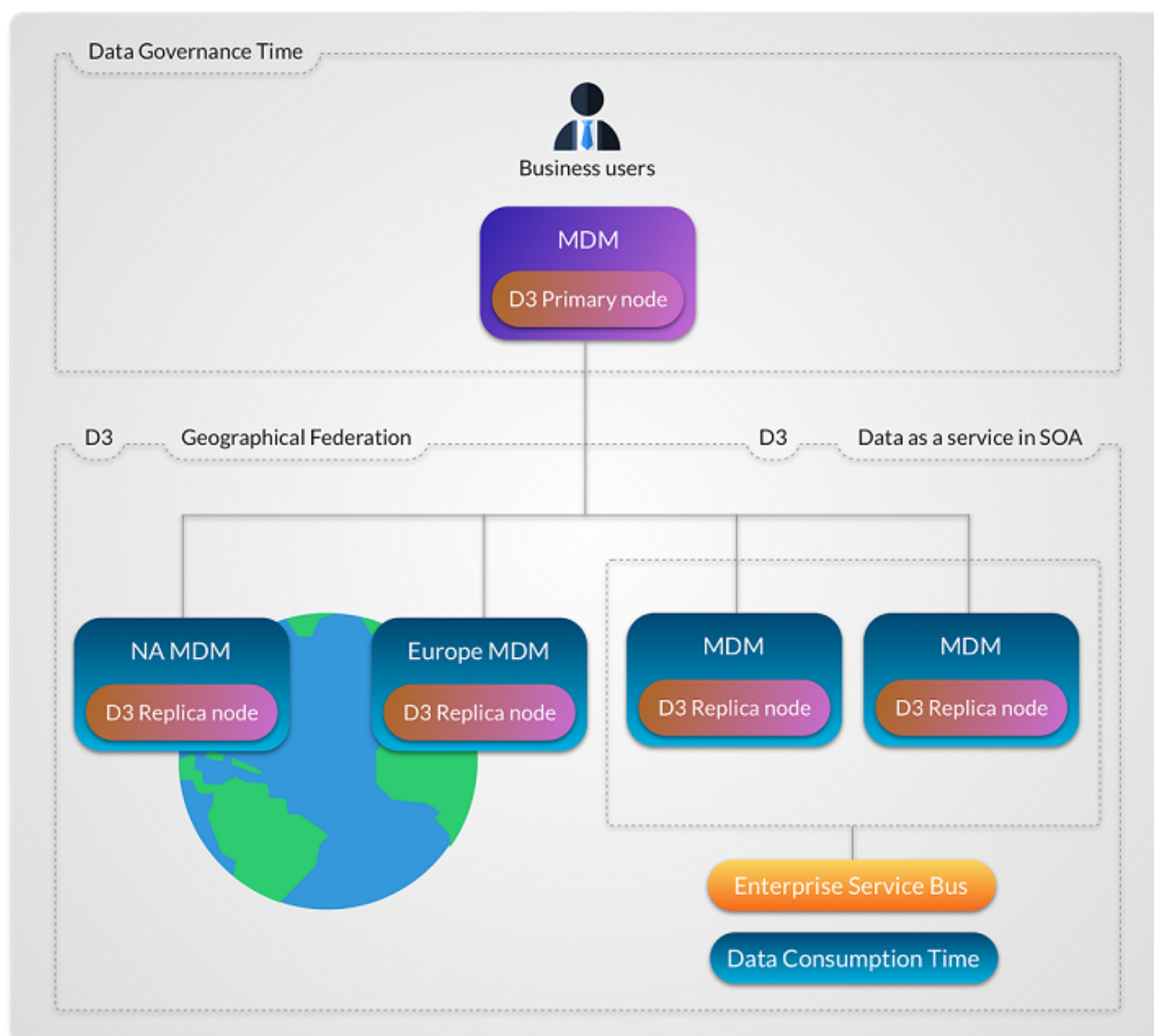
概要

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は不可能です。

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的に保証します。
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

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すが、データセットのアクセス権はブロードキャストされません。
- 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、その後同名のいかなるスナップショットがブロードキャストされた場合、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パージされ、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るのです。
- 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である必要があります。
- 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。
- レプリカノードでの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JavaAPI

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は最初に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。

ブロードキャストは、状況に応じて次のいずれかの手順を実行します。

- 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で、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。:

- 。プライマリノードのサービス

ブロードキャストの実行

ブロードキャストは、次のように実行できます。

- 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使用可能な** Broadcast**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可能）。
- `D3NodeAsMaster` を使用するカスタムJavaコードを使用する。

条件

ブロードキャストできるようにするには、次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。
- ブロードキャストされ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:****推奨されていませんが、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、「管理」領域の「プライマリノード構成」の下にある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、関連する配信データスペースとそのすべての依存関係（依存する配信マッピング、配信プロファイル、および登録されたレプリカノード）の技術レコードの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、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の違いが含まれています。

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

配信先

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、プライマリノード、レプリカノード、またはハブノードで同じ識別子を持っています。
- 配信データスペースが欠落している場合は、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で作成されます。
- 配信データスペースが最初のブロードキャストまたは完全同期ですでに存在する場合、それはオーバーライドされます。
- 配信データ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

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。それでも、「プライマリノード構成」内のブロードキャスト操作を監視することは可能です。 [監督](#)を参照してください。

レプリカノードの登録

範囲と内容

初期化は、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てから、登録されているすべての配信データスペースの最後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:

登録済みレプリカノードリポジトリIDまたは通信レイヤがすでに存在する場合は、「登録済みレプリカノード」テクニカルテーブルのレプリカノードエントリが更新されます。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。
- 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。

条件

登録するには、次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」である必要があります。
- 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である必要があります。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ている必要があります、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である必要があります。

- 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レコードとそのすべての依存関係（依存する配信プロファイル、配信マッピング、および配信データスペース）の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](#)、および[レプリカノードの構成](#)のレプリカまたはハブの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、最後にブロードキャストされたスナップショットである現在の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、要求レベルまたはグローバル構成のいずれかで変更できます。

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したり閉じたりすることはできません。権限に応じて、他の操作を利用できます。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します。

レプリカノードでは、ブロードキャストプロセスを除いて、エンドユーザー、データサービス、またはJavaプログラムのいずれによっても、配信データスペースにいかなる種類の変更も加えることができません。さらに、レプリカノードでは、マージ、クローズなどのデータスペース関連の操作は禁止されています。

D3ブロードキャストJava API

その間にブロードキャストが行われた場合、最後のブロードキャストスナップショットは2つの呼び出し間で変更される可能性があります。連続する複数の呼び出しで完全に安定したビューが必要な場合、これらの呼び出しは同じスナップショットを具体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での共通の宣言（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の場合）
 - JMS構成の例

分散データ配信用のJMS (D3)

D3がデフォルトのHTTPおよびTCPプロトコルの代わりにJMSを使用するように設定するには、**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` 配置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。

注意:

TIBCO EBX®のメイン構成が、`ebx.d3.mode`、`ebx.jms.activate` および `ebx.jms.d3.activate` プロパティによってJMSおよびD3（「スレーブ」「ハブ」「マスター」ノード）を有効化しない場合、以下の環境項目は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®の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の宣言（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>	Weblogic : <code>EBX_D3MasterQueue</code>	D3プライマリJMSキュー（D3モードの「スレーブ」または「ハブ」の場合のみ）。これは、SOAP要求をD3プラ

予約済みリソース名	デフォルトのJNDI名	説明
	JBoss : <code>java:/jms/EBX_D3MasterQueue</code>	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> の値として設定します。Javaタイプ : <code>javax.jms.Queue</code>
<code>jms/EBX_D3ReplyQueue</code>	Weblogic : <code>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）。これは、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。データ消費は、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>javax.jms.Queue</code>
<code>jms/EBX_D3ArchiveQueue</code>	Weblogic : <code>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。データ消費は、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。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大きい場合、アーカイブはいくつかのシーケンスに分割されます。したがって、データ消費は、代わりにヘッダーフィールド <code>JMSXGroupID</code> および <code>JMSXGroupSeq</code> を使

予約済みリソース名	デフォルトのJNDI名	説明
		用してフィルタリングされま す。Javaタイプ： <code>javax.jms.Queue</code>
<code>java:/</code> <code>EBX_D3CommunicationQueue</code>	WebLogic： <code>EBX_D3CommunicationQueue</code> JBoss： <code>java:/java:/</code> <code>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）。要求を受信 する通信キューの名前を指定 します。データ消費は、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>javax.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、`EBXForWebLogic.ear`（Weblogic
使用時）または `EBX.war`（JBoss、Websphere、その他のアプリケーションサーバー使用
時）に含まれるWebアプリケーションアーカイブ `ebx.ear` 内で変更することが可能です。

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の場合）

注意:

5.5.0より前の昇順の互換性、または単方向キューポートに使用されます。

プライマリノードのデプロイメント記述子は、レプリカノードごとに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で構成されます。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、関連す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。

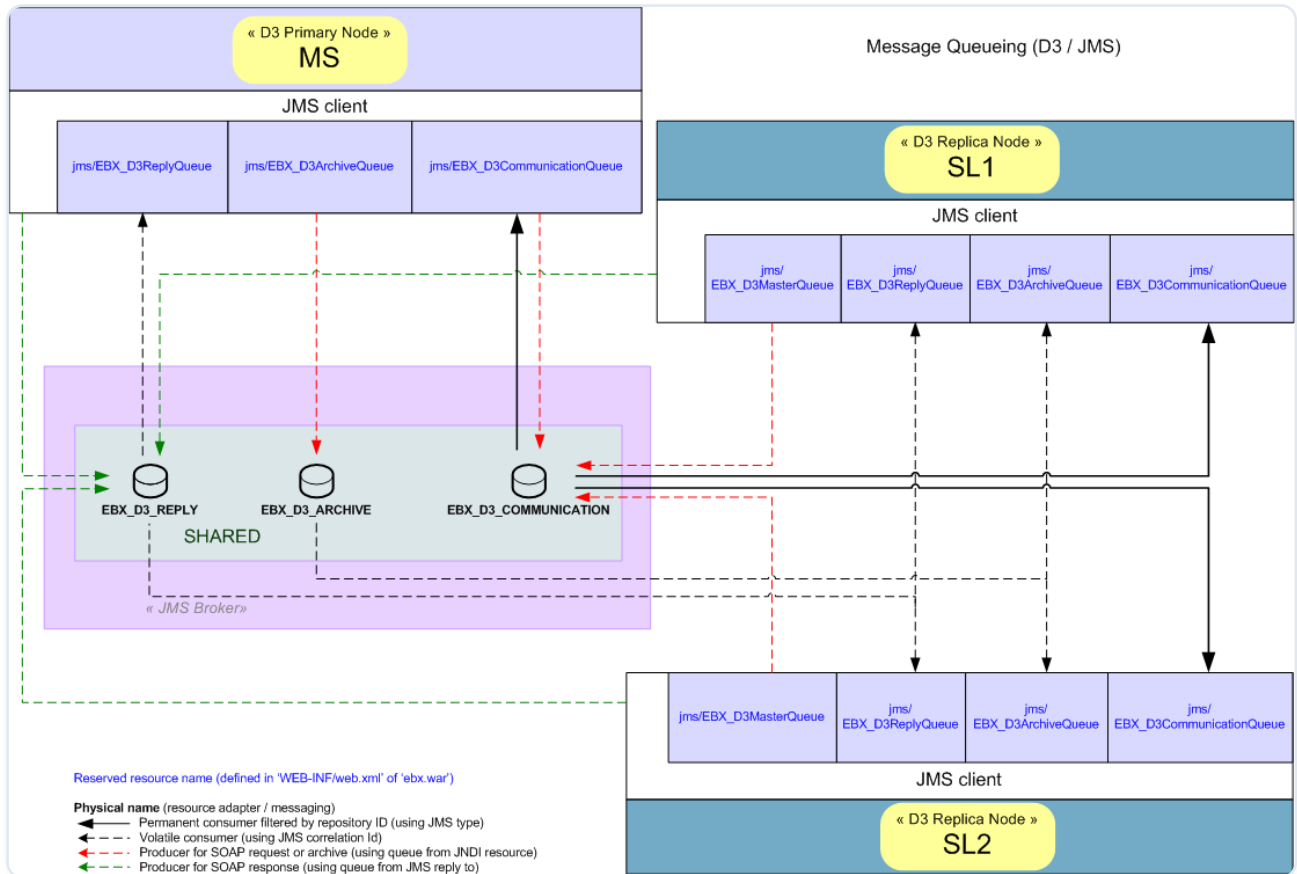
注意:

一致する物理キュー名：登録時に、レプリカノードは通信を送信し、物理キュー名をアーカイブします。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の中で物理キュー名と照合されます。一致しない場合、登録は失敗します。

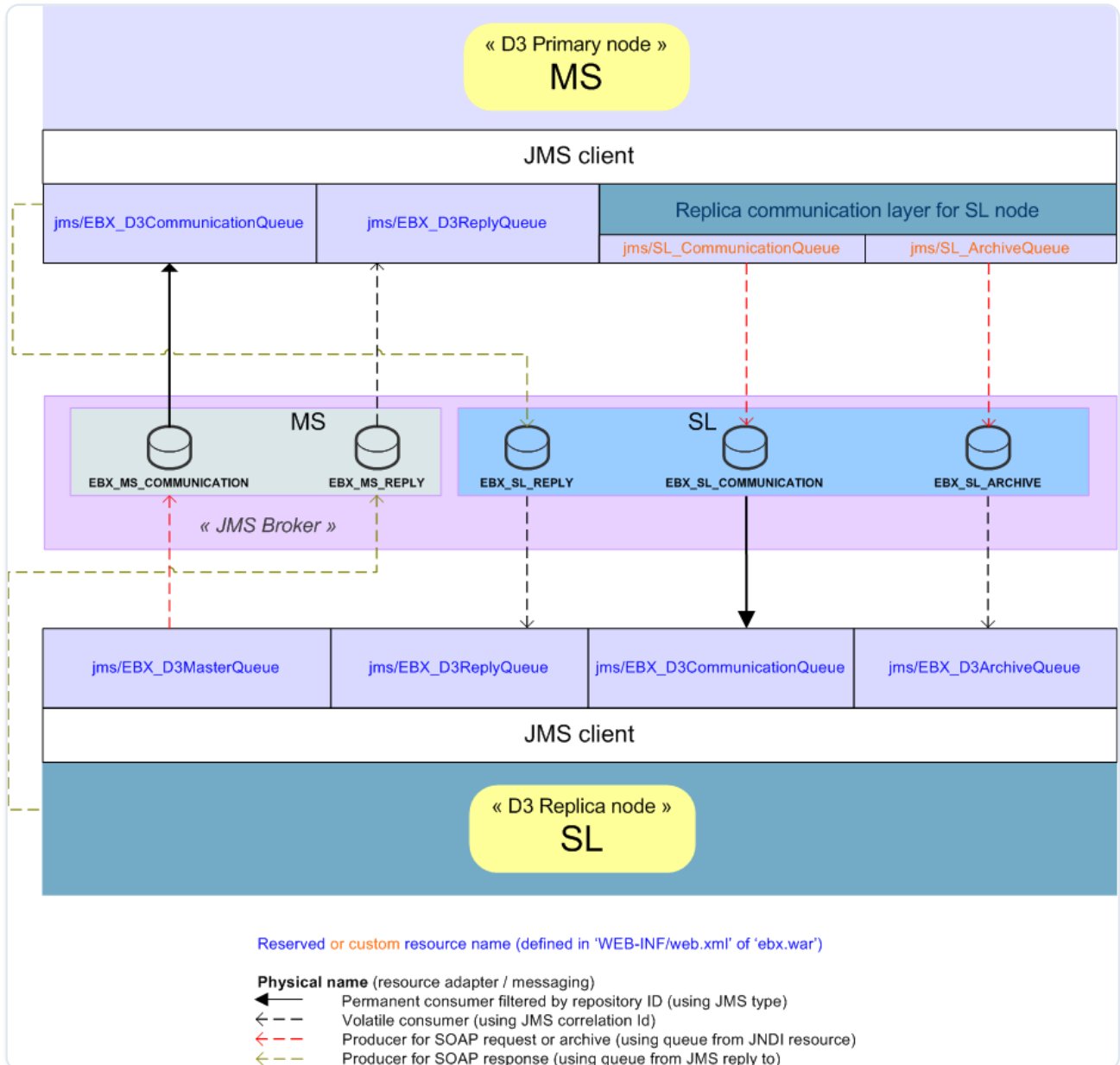
JMS構成の例

	共有キュー	特定のキュー
プライマリ-レプリカノード 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 ノードの間	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
ハブハブアーキテクチャ	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

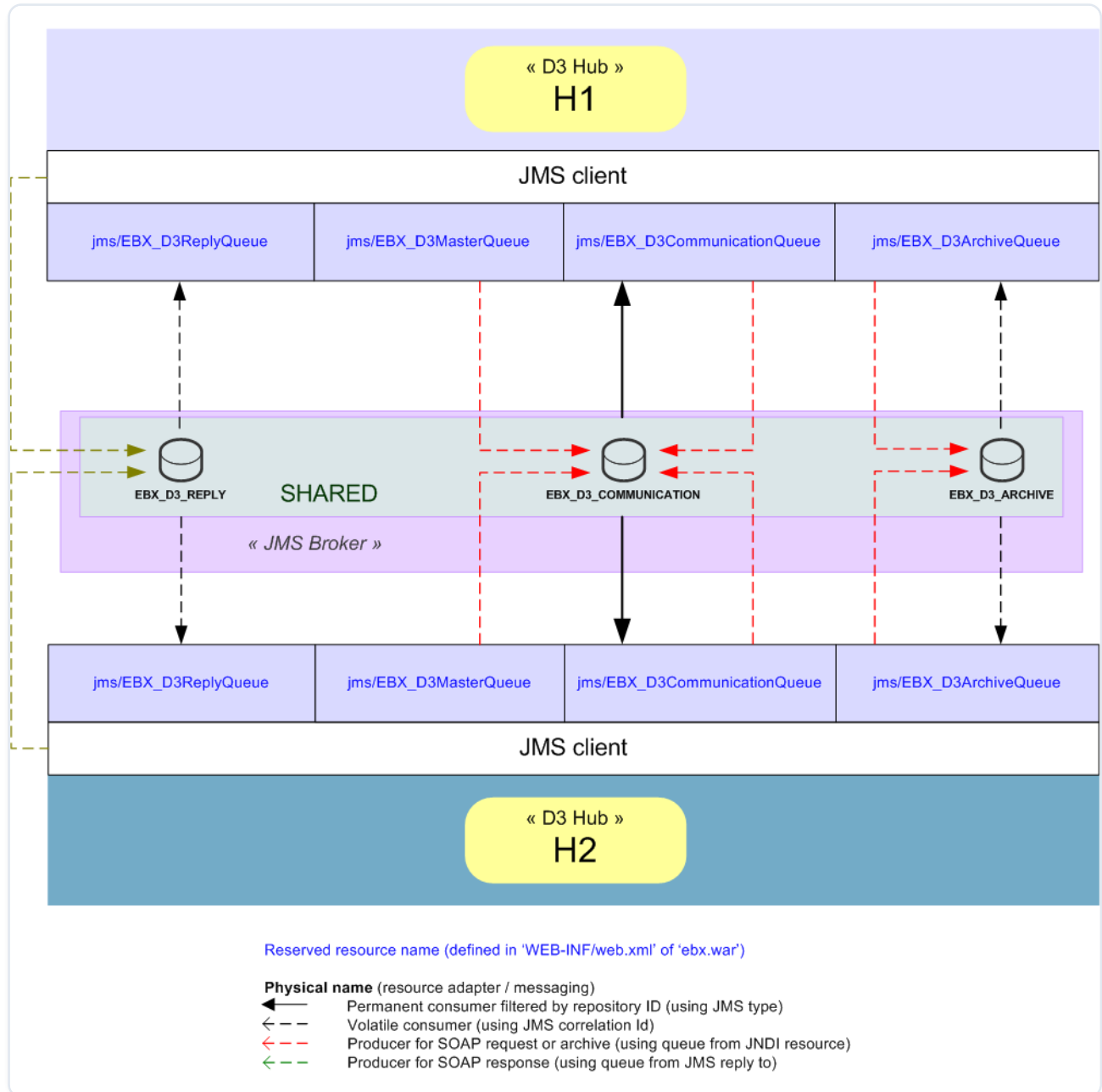
プライマリノードと共有キューを持つ2つのレプリカノード間



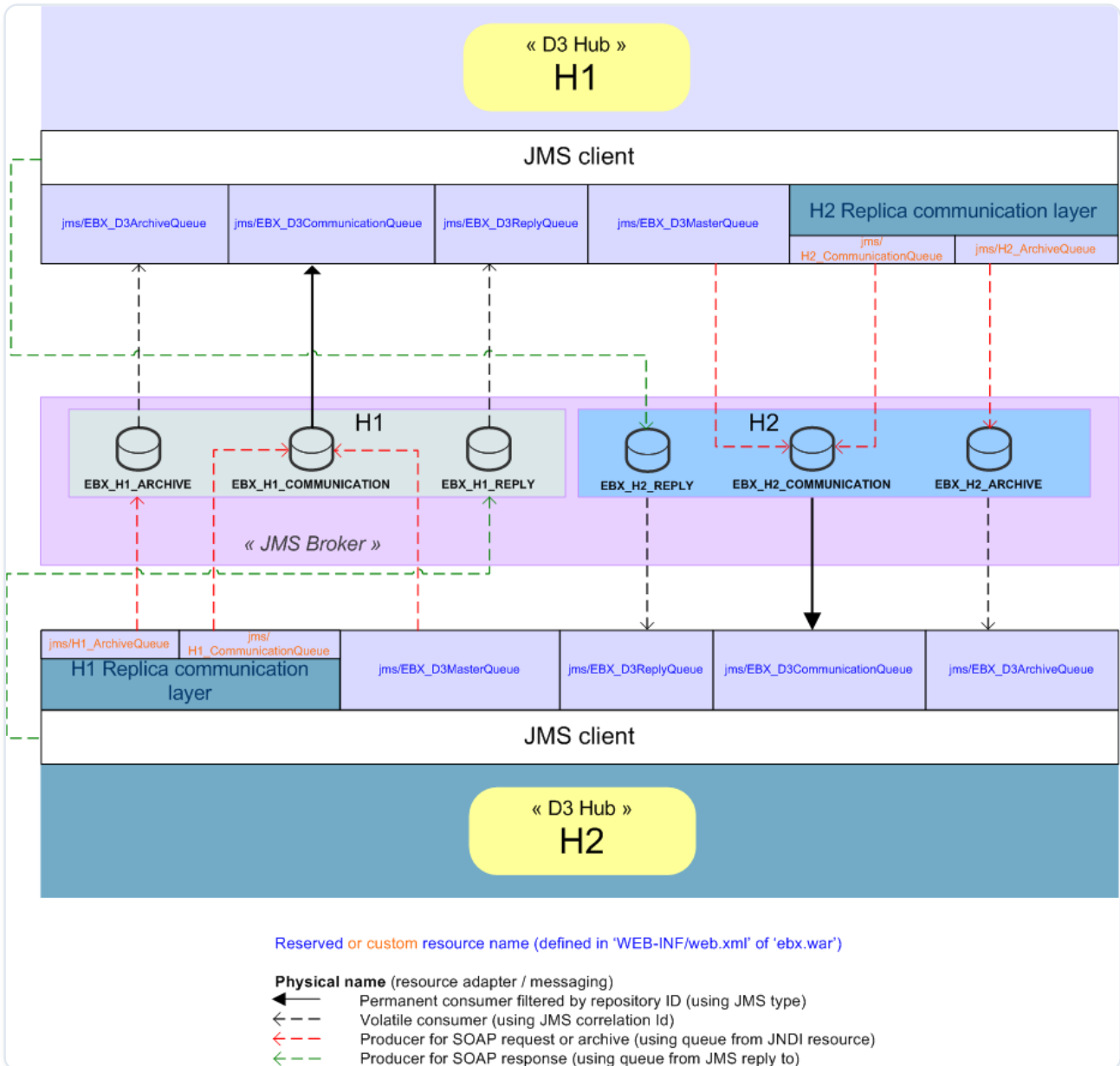
プライマリノードとレプリカ固有のキューを持つレプリカノード間



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

- プライマリノードのサービス
- レプリカノードのサービス
- 監督
 - 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デプロイします。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 `プライマリ` 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（メニューでを使用し、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 `レプリカ` 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、タブ内に移動します。

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てください (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、次の表がある領域から[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。

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、それらの状態を「使用不可」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。
レプリカノード登録の解除	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:

上記のプライマリノードサービスは、一部のブロードキャストが保留中または進行中の間は非表示になります。

レプリカノードのサービス

D3レプリカの構成ノードの下領域でサービスを利用して、プライマリノードへのサブスクリプションを管理し、その他のアクションを実行できます。

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。
レプリカノード登録の解除	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: 最後のブロードキャストスナップショットは自動的に選択から除外されます。

監督

最後のブロードキャスト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

いくつかのテーブルは、プライマリノードの「管理」領域の「D3プライマリノード構成」の下にあるプライマリノードの管理コンソールを構成します。そ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ているレプリカノード。この表から、各レコードでいくつかのサービスを利用できます。
公開履歴	行われた公開操作の履歴。
レプリカノード登録ログ	行われた初期化操作の履歴。
詳細な履歴	行わ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、テーブル「ブロードキャスト履歴」および「初期化履歴」からアクセスできます。

プライマリノード監視サービス

「プライマリノード構成」の下でのプライマリノードの「管理」領域で使用できます。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、関連す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。
明確な履歴コンテンツ	「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のすべてのレコードを削除します。

Java APIを介したレプリカノードの監視

レプリカノード監視クラスを作成して、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「D3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ように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、詳細へのオプションのリンクを含むイベントの表が含まれています。

通知を有効にするには、「管理」領域から「プライマリノード構成」データスペースを開き、「グローバル構成」の下で「通知」グループを構成します。

「電子メールから」および「URL定義」オプションも、「電子メール構成」リンクを使用して構成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ているログカテゴリ「ebx.d3」を介して実行できます。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
- REST API
- 一時フォルダの処理
- 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。

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. 目的の要素を選択し、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、ステージングする要素を定義します。選択リストは、変更時に自動的に保存されます。

注意:

エラー状態の要素は非表示になり、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 個の要素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

2. 画面上部またはドメインテーブルの各行にある [ステージングアーカイブ](#) 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。
3. ステージングアーカイブをインポートします。
 - a. ターゲットリポジトリで、[>](#) に移動し、[ステージングアーカイブ](#) を選択します。
 - b. 画面上部の [ステージングアーカイブ](#) をクリックします。
 - c. ステージングアーカイブを参照して選択します。
 - d. 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。
 - e. [インポート](#) 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。

Administration **Import a staging archive**

Staging

Actions

Domains

▼ Reports

Import operations

Export operations

► Configuration

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上書き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

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。

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。

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` プロパティを設定します。
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,
        "importDomainDefinition": 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みなされ、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。

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、メインプロトコルである必要があります。

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 は遅くて耐性が低いです。

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。

以下の設定は、EBX® でサポートされているブラウザと互換性があります。

- 暗号スイート：ECDHE-ECDSA-AES256-GCM-SHA384：ECDHE-RSA-AES256-GCM-SHA384：ECDHE-ECDSA-CHACHA20-POLY1305：ECDHE-RSA-CHACHA20-POLY1305：ECDHE-ECDSA-AES128-GCM-SHA256：E-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特権のないユーザーを使用して、可能な限り **最小特権の原則** 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

```
header always unset x-xss-protection

header always set x-xss-protection "1; mode=block"
```

Apache2 での有効化

```
proxy_hide_header x-xss-protection;
```

```
add_header x-xss-protection "1; mode=block" always;
```

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 sets 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 sets 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
```

Ngix での有効化

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains"
always;
```

Apache2 での有効化

```
header always sets 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 sets 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';"
```

リファラーポリシー

Referrer-Policy HTTP ヘッダーは、行われた要求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 sets Referrer-Policy "strict-origin"
```

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" 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。暗号化用の秘密鍵がある場合は、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

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。これを担当する人は、**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している必要があります。

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
- 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
 - 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
- EBX®スクリプティング
 - レコードの権限
 - 関数フィールド
 - 関数フィールドAPI
 - ユニットの概要
 - ユニットのデフォルト

- ユニット core.complex
 - ユニット core.date
 - ユニット core.datetime
 - 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.resource
 - ユニット core.string
 - ユニット core.time
 - ユニット core.uri
- Java API（英語）ポップアップでJavaAPI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、標準のJava EE Webアプリケーションであり、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ます。

EBX®モジュールはWebアプリケーションであるため、クラスローディングの分離、WARまたはEARパッケージング、Webリソースの公開などの機能の恩恵を受けます。

- モジュール構造
- モジュール宣言
- モジュール登録
 - デプロイメント記述子の例
 - 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JavaEEデプロイメント記述子です。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行うことができます。 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「Implementation-Title」と「Implementation-Version」の値を 管理>技術構成>モジュールとデータモデルに報告 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を介して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ています。 パッケージ化されたリソース を参照してください。

Oracle WebLogicサーバーに必要なファイル：

<code>/WEB-INF/weblogic.xml</code>	次のような <code>prefer-web-inf-classes</code> ポリシーをアクティブ化するWebLogicデプロイメント記述子ファイル。 <pre><?xml version="1.0" encoding="UTF-8"?> <weblogic-web-app xmlns="http://xmlns.oracle.com</pre>

```
<container-descriptor>
```

```
<prefer-web-inf-classes>true</prefer-web-inf
```

```
</container-descriptor>
```

```
</weblogic-web-app>
```

詳細については、[weblogic.xmlデプロイメント記述子エレメント](#)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。例を次に示します。

```
<?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 http://
schema.orchestranetworks.com/module_2.4.xsd">

  <name>moduleTest</name>

</module>
```

各プロパティの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の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（そのディレクトリの名前）に対応します。	はい

エレメント	説明	必須
<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>name</code> です。	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以降では、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
<code>beans</code>	再利用可能なJavaBean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
<code>ajaxComponents</code>	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®で識別できるようにするには、アプリケーションサーバーの起動時にモジュールを実行時に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®モジュールは次のことを行う必要があります。

- クラス `ModuleRegistrationListener` を拡張するアノテーション `@WebListener` を持つ Javaクラスが含まれています。

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化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 TMサーブレット3.0仕様](#)を参照してください。

また、以下にも注意ください。

- クラス `ModuleRegistrationServlet` を拡張するサーブレットが含まれています。
- デプロイメント記述子 `/WEB-INF/web.xml` でこのサーブレットの標準宣言を行います。
- `<load-on-startup>1</load-on-startup>` の標準エレメントをデプロイメント記述子に追加して、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。

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、一般的なJavaScriptやCSSリソースなどの追加の動作を定義できます。
- `ModuleRegistrationServlet` を拡張する特定のクラスは、Webアプリケーション（`/WEB-INF/classes` または `/WEB-INF/lib` の下）に配置する必要があります。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によって使用される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ます。
- アプリケーションサーバーの起動プロセスは非同期であり、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、使用されているすべてのモジュールの登録を無制限の時間待機します。結果として、使用済みモジュールが何らかの理由でデプロイ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を未デプロイとして宣言する](#)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、原因はアプリケーションサーバーのログに記録されます。
- 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[サポートされていない](#)ことに注意してください。

デプロイメント記述子の例

次に、Java EEデプロイメント記述子（`/WEB-INF/web.xml`）の例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

        https://www.oracle.com/webfolder/technetwork/
jsc/xml/ns/javaee/web-app_3_0.xsd"

    version="3.0">

    <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

`ModuleRegistrationServlet` の実装例を次に示します。

```
package com.foo;

import javax.servlet.*;

import javax.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るファイルとドキュメントであり、`osd:resource` フィールドまたはJava APIを介して管理および指定できます。それらには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、ディレクトリでローカリゼーションを指定する必要があります。以下のとおりです。
 - `common/` には、ロケールに依存しないか、デフォルトのローカリゼーションとして使用されるため、デフォルトで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`（英語）です。）
 - `{lang}/` は、下にあるリソースのローカリゼーションが必要な場合、`{lang}` は実際のロケールコードに置き換えられます。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、ディレクトリでリソースタイプを指定する必要があります。以下のとおりです。
 - JavaScriptリソースの場合は、`jscripts/` です。
 - カスケードスタイルシート（CSS）リソースの場合は、`stylesheets/` です。
 - HTMLリソースの場合は、`html/` です。
 - アイコンタイプのリソースの場合は、`icons/` です。
 - 画像タイプのリソースの場合は、`images/` です。

例

この例では、画像 `logoWithText.jpg` がローカライズされている唯一のリソースです。

```

/www
|
├─ common
|
|   ── images
|       ── myCompanyLogo.jpg
|       ── logoWithText.jpg

```

```

|   └─ jscripts
|
|   └─ ┬─ myCompanyCommon.js
|
|   └─ ┬─ stylesheets
|
|       └─ myCompanyCommon.css
|
└─ de
|
|   └─ images
|
|       └─ logoWithText.jpg
|
└─ fr
|
|   └─ images
|
|       └─ logoWithText.jpg

```

Java へのマッピング

- Java からデータにアクセスする方法
 - 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
 - 適応オブジェクト
- データ型のマッピング
 - 単純なデータ型
 - 複合データ型
- Java バインディング

- 利点
- XML 宣言

Java からデータにアクセスする方法

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、または
- 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 `Procedure` と、複数の同時実行可能な `ReadOnlyProcedure` の実行をサポートしています。任意の `Procedure` 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）： - データモデルの公開では、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、クエリの実行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。 <code>Procedure</code> 外部のクエリは、自己完結型の <code>ReadOnlyProcedure</code> と見なすことができます。
<code>Procedure</code> の基になるデータスペースと同じデータスペース内の <code>Procedure</code> 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 <code>プロシージャ</code> で発生した変更を反映します。 <code>プロシージャ</code> 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 <code>Procedure</code> 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 <code>プロシージャ</code> 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。

適応オブジェクト

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	<code>Adaptation</code> 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、<code>Adaptation</code> 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 <code>Adaptation.getUpToDateInstance</code> を呼び出すことは可能です。
フェッチ	<code>Adaptation</code> が <code>QueryResult</code> 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 <code>Procedure</code> から <code>Adaptation</code> がフェッチされると、<code>Procedure</code> 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、<code>QueryResult</code> または実行中の <code>Procedure</code> の外部では、<code>Adaptation</code> 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。: <code>AdaptationHome.findAdaptationOrNull</code> <code>AdaptationTable.lookupAdaptationByPrimaryKey</code>

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。

単純なデータ型

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 スキーマに組み込まれた単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雑な型の定義

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。

メソッド `SchemaNode.createNewOccurrence`、`SchemaNode.executeRead`、および `SchemaNode.executeWrite` を呼び出すことにより、属性 `osd:class` の有無にかかわらず、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である必要があります、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。

バイディング XML スキーマを参照してください。

XML バイディングの例

```
<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>

      </ebxbnd: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` を持つエレメント `javaPathConstants` によって記述されます。関連するノードは、属性 `root` を持つエレメント `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。

- 開発ツールのアクティブ化
- 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

開発ツールのアクティブ化

開発ツールをアクティブ化するには、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、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、を選択します。

注意:

データの整合性に関して操作が重要であるため、データモデルを更新すると、リポジトリのグローバル排他ロックが取得されます。これは、他のほとんどの操作（データアクセスと更新、検証など）がデータモデルの更新が完了するまで待機することを意味します。

Javaバインディングの生成

Javaバインディングで指定されたJavaタイプは、ナビゲーションペインで**アクション> Javaの生成**を選択することにより、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

「データパス」フィールドが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ます。これは、XPath式を作成するときに役立ちます。

注意:

このフィールドは、管理者がいつでも利用できます。

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」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**アクション> Webコンポーネントリンクジェネレーター**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	適応インスタンス
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
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

新しい用語	バージョン 5.0.0 より前の用語
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](#)

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的な定義です。データモデルは、最高レベルのデータ整合性を保証し、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）に準拠するドキュメントです。その主な機能は次のとおりです。

- 整数、ブール値、小数、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](#)および[複雑型](#)を定義する機能
- [集約リスト](#)と呼ばれるアイテム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。

- 事前定義されたタイプ 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® データのJavaBeanへのマッピング
- 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:

一部の機能はマスターデータ管理に関連しないため、EBX®はW3C推奨のサブセットをサポートします。

- データモデルの編集
- 参考資料
- 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
- 慣例
- 予約名のあるスキーマ

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、DMA の外部での使用がドキュメント化されていない機能 (例：ツールバーとウィジェット) を使用できます。

- 外部XMLスキーマドキュメントエディターを使用します。

参考資料

XMLスキーマの概要については、W3Schools [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: 入門書
- 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、データセットの基礎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®によって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="http://schema.orchestranetworks.com/
common_1.0.xsd"/>

    <xs:element name="root" osd:access="--">

        ...

    </xs:element>
```

```
</xs:schema>
```

慣例

慣例により、ネームスペースは常に次のように定義されます。

プレフィックス	ネームスペース
<code>xs:</code>	http://www.w3.org/2001/XMLSchema
<code>osd:</code>	<code>urn:ebx-schemas:common_1.0</code>
<code>fmt:</code>	<code>urn:ebx-schemas:format_1.0</code>
<code>usd:</code>	<code>urn:ebx-schemas:userServices_1.0</code>
<code>emd:</code>	<code>urn:ebx-schemas:entityMappings_1.0</code>

予約名のあるスキーマ

EBX®のいくつかのデータモデルには予約名があります。

次の文字列のいずれか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、この場所にあるファイル、またはファイル名が同じ場合は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役立ちます。

セキュリティ上の理由から、EBX®はこれらのスキーマの内部定義を使用して変更を防ぎます。

データ型

この章では、TIBCO EBX®でサポートされ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スキーマに組み込まれた単純型
- 単純型という名前のXMLスキーマ
- 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
 - ユーザー定義の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スキーマに組み込まれた単純型

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	メモ
<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

XMLスキーマタイプ	JAVAクラス	メモ
		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> 制限項)	<code>java.lang.String</code>	
<code>xs:int</code> (<code>xs:decimal</code> 制限項)	<code>java.lang.Integer</code>	
<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。

単純型という名前のXMLスキーマ

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 `striction` 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、要素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オーバーライドされます。

ただし、この制限は要素 `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、要素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、要素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、要素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、要素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> 制限項)	<code>java.lang.String</code>
<code>osd:html</code> (<code>xs:string</code> 制限項)	<code>java.lang.String</code>
<code>osd:email</code> (<code>xs:string</code> 制限項)	<code>java.lang.String</code>
<code>osd:password</code> (<code>xs:string</code> 制限項)	<code>java.lang.String</code>
<code>osd:color</code> (<code>xs:string</code> 制限項)	<code>java.lang.String</code>
<code>osd:resource</code> (<code>xs:anyURI</code> 制限項)	内部クラス
<code>osd:locale</code> (<code>xs:string</code> 制限項)	<code>java.util.Locale</code>
<code>osd:dataspaceKey</code> (<code>xs:string</code> 制限項)	<code>java.lang.String</code>
<code>osd:datasetName</code> (<code>xs:string</code> 制限項)	<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>
osd:html	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>
osd:email	これは、RFC822標準で指定されている電子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>
osd:password	これは、ハッシュまたは暗号化されたパスワードを表します。EBX®には特定のエディターが用意されています。 <pre><xs:element name="password" type="osd:password"></pre> 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。この暗号化関数は、メソッド <code>DirectoryDefault.encryptString</code> を使用してJavaクライアントからも使用できます。 インターフェイス <code>暗号化</code> 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。

```
<xs:element name="password" type="osd:password">

    <xs:annotation>

        <xs:appinfo>

            <osd:uiBean class="com.orchestranetworks.ui.password.PasswordBean">

                <encryptionClass>package.Encryptor</encryptionClass>

            </osd:uiBean>

        </xs:appinfo>

    </xs:annotation>

</xs:element>
```

別のフィールドへのパスを参照することでソルトを指定することができ、クラスを使用することで、インターフェイス

`HashComputation` を実装します。

```
<xs:element name="password" type="osd:password">

    <xs:annotation>

        <xs:appinfo>

            <osd:uiBean class="com.orchestranetworks.ui.password.PasswordBean">

                <encryptionClass>package.HashComputation</encryptionClass>

                <saltPath>../login</saltPath>

            </osd:uiBean>

        </xs:appinfo>

    </xs:annotation>
```


	<pre></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> <xs:enumeration value="ar_AE" osd:label="Ar_AE"> ... </xs:enumeration> <xs:enumeration value="ar_BH" osd:label="Ar_BH"> ... </xs:enumeration> <xs:enumeration value="ar_DZ" osd:label="Ar_DZ"> ... </xs:enumeration> <xs:enumeration value="ar_EG" osd:label="Ar_EG"> ... </xs:enumeration> <xs:enumeration value="ar_IQ" osd:label="Ar_IQ"> ... </xs:enumeration> ... <xs:enumeration value="vi_VN" osd:label="Vi_VN"> ... </xs:enumeration> <xs:enumeration value="zh" osd:label="Ch"> ... </xs:enumeration> <xs:enumeration value="zh_CN" osd:label="Ch_CN"> ... </xs:enumeration> <xs:enumeration value="zh_HK" osd:label="Ch_HK"> ... </xs:enumeration> <xs:enumeration value="zh_TW" osd:label="Ch_TW"> ... </xs:enumeration> </xs:restriction> </xs:simpleType></pre>

osd:color	これは、16進数のRGBカラーフォーマット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>
osd:resource	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	このタイプは、データスペースへの参照を表します。 <pre><xs:element name="dataspaceField" type="osd:dataspaceKey"/></pre> EBX®には、参照可能なデータスペースを表示する特定のエディターが用意されています。 <code>xs:annotation/xs:appInfo</code> の下のエレメント <code>osd:dataspaceSet</code> を使用して参照できるデータスペースを指定することができます。エレメント <code>osd:dataspaceSet</code> が定義されていない場合、デフォルトでは、開いているブランチのみを参照できます。

```

<xs:element name="dataspaceField" type="osd:dataspaceField">
  <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>
        </include>
        <filter osd:class="com.foo.MyData">
          <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。 `include` を少なくとも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。 `スナップショット` 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。Includes（含む）が定義されていない場合、Excludes（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 `osd:class` は、インターフェイス `DataspaceSetFilter` を実装する JavaBean を指定します。

`xs:annotation/xs:appInfo/`
`osd:dataspaceSet` 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> <pre><xs:annotation> <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></pre>

```
</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。Includes（含む）が定義されていない場合、Excludes（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 **JavaBean** 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="1"> <xs:attribute name="osd:UDACatalogPath" type="string" value="//userCatalog" /> </xs:element></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応します。
<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" minOccurs="1" maxOccurs="1"> <xs:annotation base="xsd:documentation"> <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> <pre><xs:element name="userCatalog" type="osd:UDACatalog" minOccurs="1" maxOccurs="1"> <xs:annotation base="xsd:documentation"> <xs:documentation xml:lang="en-US">User Catalog</xs:documentation> <xs:documentation xml:lang="fr-FR">Catalogue utilisateur</xs:documentation> </xs:annotation> </xs:element></pre> 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エレメント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UIBean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` エレメントにも存在します。

集約リスト

XML スキーマでは、要素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 より大きいか、`制限なし` 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、この要素は **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（1～2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>
```

```
</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、[集約リスト](#)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](#)
- [単純表形式ビュー](#)
- [階層ビュー](#)

- 履歴

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。 <pre><defaultLabel>Product: \${./productCode}</ defaultLabel></pre>	いいえ

エレメント	説明	必須
	- ローカライズされた静的な式は、属性 <code>xml:lang</code> を持つ <code>defaultLabel</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> <defaultLabel xml:lang="fr- FR">Produit : \${./ productCode}</ defaultLabel> <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 <code>JavaBean</code>。 <code>JavaBean</code> 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> <defaultLabel osd:class="com.wombat.MyLabel"></ defaultLabel> </pre> 注意:	

エレメント	説明	必須
	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. ローカライズされていない静的な式を持つ <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いるフィールドは、ラベルに表示されます。	

エレメント	説明	必須
<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:element>
```



```

</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>

          <primaryKey>/catalogId</primaryKey>

        </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り、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

[テーブル](#)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す。

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> <pattern xml:lang="fr- FR">Produit : \${./ productCode}</ pattern> <pattern xml:lang="en- US">Product: \${./ productCode}</ pattern> </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ーを満たさない場合、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。データサービスで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」カーディナリティ制約 (`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 > 1</code>) 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 • 一意性制約	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」。
<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。	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
	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。 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。	

エレメント	説明	必須
<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、現在のテーブルに対して解決されます。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。	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
	<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。 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。	
<code>filter</code>	<code>predicate</code> 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。	いいえ

エレメント	説明	必須
	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。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。	いいえ

エレメント	説明	必須
	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。	
<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>	いいえ

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、	はい

エレメント	説明	必須
	<code>SchemaLocation</code> を参照してください。	
<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> で指定されたデータモデルを使用する必要があります。	はい
<code>dataSpace</code>	関連付けで使用され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。
 - 許可の観点からは非表示と見なされます。つまり、関連付けはユーザーインターフェイスとデータサービスで非表示になります。
 - 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。
 - 関連付けの結果を計算する既存の API で例外が発生します。

`AssociationLink.getAssociationResult` と `AssociationLink.getTargetTable` を参照してください。

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
<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>

```

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である必要があり、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
<code>sort</code>	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 <code>sort</code> 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 <code>nodePath</code> 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 <code>isAscending</code> 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実体化されます。
- **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:

関連付けと切断のアクションは、** 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	いいえ

エレメント	説明	必須
	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。	
<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、正の整数である必要があります。	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未加工の文字列 <code>unbounded</code> のいずれかである必要があります。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	いいえ。デフォルトでは最大値は制限されていません。

エレメント	説明	必須
	最小数よりも大きくする必要があります。	
<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。	いいえ
<code>validation</code>	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	いいえ

エレメント	説明	必須
	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。	

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、[前述した例](#)で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:annotation>

</xs:element>

```

リンクテーブルを介した関連付けの場合、前述した例を検討して、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を削除し、製品とカタログの関係を、2つの外部キー（「Products」テーブル（「productRef」という名前）と「Catalogs」テーブル（「catalogRef」という名前）を参照）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）によって再定義します。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>

                <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="0" type="xs:string">

    <xs:annotation>

        <xs:appinfo>

            <osd:association>

                <tableRefInverse>

                    <fieldToSource>/root/Products/catalog_ref</
fieldToSource>

                </tableRefInverse>

                <maxOccurs>

                    <value>10</value>

                <validation>

                    <severity>warning</severity>

                    <message xml:lang="en-US">Too much products for
this catalog.</message>

```

```

        <message xml:lang="fr-FR">Ce catalogue a trop de
produits.</message>

    </validation>

</maxOccurs>

</osd:association>

</xs:appinfo>

</xs:annotation>

</xs:element>

```

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」である必要があります。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: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。

前述した例で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 `Products` 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
<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
<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
<code>osd:resource</code> 3						
<code>osd:dataspaceKey</code> 4						
<code>osd:datasetName</code> 4						
<code>osd:color</code> 4						

例：

```
<xs:element name="loanRate">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="4.5" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



```

        <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。

例：

以下の例では、ターゲットフィールド「name」の「publisher」テーブルに一意性制約が定義されています。これは、「publisher」テーブルの2つのレコードが同じ名前を持つことはできないことを意味します。

```

<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>

            <severity>error</severity>

            <message>Name must be unique in table.</message>

            <message xml:lang="en-US">Name must be unique in
table.</message>

            <message xml:lang="fr-FR">Le nom doit être unique dans
la table.</message>

        </osd:validation>

    </xs:appinfo>

</xs:annotation>

<xs:selector xpath="." />

<xs:field xpath="name" />

</xs:unique>

</xs:element>

```

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

プロパティ	説明	必須
<code>xs:field</code> エlement	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> Element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、Element `annotation/appinfo` の下のElement `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、テーブルのスコップではなく、リストのスコップ内で一意である必要があります。

例:

以下の例では、ターゲットフィールド「printedEditions」の「title」テーブルに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。

```
<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>

                <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。

- 一意性の制約は、計算フィールドには適用されません。
- 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。
- 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>
```

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` のタイプ `integer`。
 - 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互換性があります。
- 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タイプ](#)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「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1つである必要があります。
<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、「イメージ」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www/common/images/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。これ。さらに、リソースがローカライズされたディレクトリ（www/fr/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www/common/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り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（Java 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、フィールドの生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` から供給されています。

```
<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` :

- 集約リスト、つまり `maxOccurs > 1` である必要があります。
- 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である必要があります。
- 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である必要があります。
- 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: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エレメント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エレメント `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 `制約` 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生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できるように。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、[検証パフォーマンス](#)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いずれも拒否され、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ますなど）。
never	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">

      <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>

        <blocksCommit>onInsertUpdateOrDelete</blocksCommit>

      </osd:enumerationValidation>

    </xs:appinfo>

  </xs:annotation>

  <xs:simpleType>

    <xs:restriction base="xs:string">

      <xs:enumeration value="0" osd:label="male" />

      <xs:enumeration value="1" osd:label="female" />

    </xs:restriction>

  </xs:simpleType>

</xs:element>
```

EBX®ファセット

制御ポリシーは、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。

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。

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によると、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。むしろ、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、エレメントは属性

`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。非終端エレメントの場合、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」に対する制約
- 空白の管理
- 空の文字列管理

データ型のEBX® 空白管理

XML Schema (<https://www.w3.org/TR/xmlschema-2/#rf-whiteSpace>) 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わなければなりません。

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。ス)。

collapse

replaceプロシーダの後、連続する

#x20 の列は1つの #x20 に折りたたまれ、先頭または末尾の #x20 は削除されます。

一般的な空白の処理

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、空の文字列はユーザー入力として `null` として解釈されますEBX®による。

ユーザー入力時の空白の処理

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「1234」と「1234」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以上である必要があります。

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されたフィールドを提供します。

- 計算値
 - 検証の無効化
- トリガー
 - データセットでのトリガー
 - テーブルでのトリガー

- 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リポジトリで読み取られ、永続化されます。それにもかかわらず、データは、計算および/または外部データベースアクセス、たとえばRDBMSまたは中央システムの結果である可能性があります。

EBX®では、現在のデータセットコンテキストの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*を定義することで可能になります。

計算ルールは、`osd:function` エlement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クラスの修飾名である必要があります。
- 追加のパラメーターをデータモデルレベルで指定できます。その場合、JavaBean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は、計算値の検証を無効にすると便利な場合があります。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、削除などの一部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の下で宣言する必要があります。

データセットでのトリガー

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クラスをエレメント `osd:trigger` 内で宣言する必要があります。

データセットトリガーの場合、データモデルの `root` エレメントのすぐ下に `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>
```

テーブルでのトリガー

テーブルレコードのトリガーを定義するためには、抽象クラス `TableTrigger` を継承したJavaクラスを `osd:trigger` 内に定義する必要があります。 `annotation/appinfo/osd:trigger` エレメントは、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定義することをお勧めします。

例：

テーブルエレメント：

```
<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>
```

テーブルタイプエレメントの場合：

```
<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。上記の例では、Javaクラスでメソッドを定義する必要があります：

`getParam1()`、`setParam1(String)`、`getParamX()`、`setParamX(String)` など。

自動インクリメントされた値

自動インクリメントされた値を定義することが可能です。自動インクリメントされた値はテーブル内でのみ許可され、タイプは `xs:int` または `xs:integer` である必要があります。

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で指定されます。

例：

```

<xs:element name="autoIncrementedValue" type="xs:int">

    <xs:annotation>

        <xs:appinfo>

            <osd:autoIncrement />

        </xs:appinfo>

    </xs:annotation>

</xs:element>

```

また、`start` と `step` の2つのオプションエレメントがあります。

- `start` 属性は、この自動インクリメント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、値 `1` がデフォルトで設定されます。
- `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、値 `1` がデフォルトで設定されます。

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>
```

`osd:autoIncrement` を指定するフィールドの動作は、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、フィールド値が未定義の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。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、その値は保持されます。

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に対しては、割り当ては実行されません。

- 新しく割り当てられた値は、可能な場合は常に、リポジトリの範囲内で一意です。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またがり、すべてのデータスペースにもまたがります。後者の場合、`osd:autoIncrement` がレコードの主キーの一部である場合に競合が発生しないことが合理的に保証された状態で、データスペースをその親にマージできます。

この原則には、非常に特殊な制限があります。値を指定する一括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場合、後者のトランザク

ションは、によって設定される値を割り当てる可能性があります。最初のトランザクション（異なるデータスペース間の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」テーブルに格納されます。ユーザーインターフェイスでは、管理者が領域からアクセスできます。このフィールドは自動的に更新されるため、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、関連する `osd:autoIncrement` フィールドに設定された最大の値が定義されます。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、それぞれが異なる自動インクリメント範囲を持つ複数の環境（開発、テスト、実稼働）を管理する必要がある場合、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現できます。どうしても必要な場合を除いて、このプロパティを有効にすることは通常お勧めしません。これにより、自動インクリメント値に競合が発生する可能性があります。ただし、このプロパティは次の方法で設定できます。

- ローカルでは、自動インクリメント宣言でパラメーターエレメントを設定することにより：
`<disableMaxTableCheck>true</disableMaxTableCheck>`、
- データモデル全体の場合、データモデル宣言のエレメント `xs:appinfo` で
`<osd:autoIncrement disableMaxTableCheck="true"/>` を設定するか、または
- グローバルに、EBX®メイン構成ファイルでプロパティ
`ebx.autoIncrement.disableMaxTableCheck=true` を設定します。

TIBCO EBX®メイン構成ファイルを参照してください。

注意:

このオプションをグローバルに有効にすると、たとえばXMLまたはCSVからインポートすることにより、自動インクリメントのテーブルにレコードを作成できるようになります。このオプションが選択されていない場合、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のテーブルにレコードを作成することは禁止されています。

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。

- ラベルと説明

- 動的ラベルと説明
- 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)

ラベルと説明

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。

Full (フル)	ラベルと説明は、それぞれエレメント <code><osd:label></code> と <code><osd:description></code> によって定義されます。
Simple (シンプル)	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文字です。説明では、テキストの残りの部分を使用します。

説明には、外部ドキュメントへの標準HTML `href`、またはEBX®内の適応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があります。

- `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ないため、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">で適応内</osd:link>の他のノードへリンクします。
```

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わります。同じメカニズムがノードの説明にも適用されます（エレメント `osd:description`）。

注意:

空白の管理に関しては、ノードのラベルは表示時に常に折りたたまれています。つまり、連続する一連の空白が1つの空白に折りたたまれ、先頭と末尾の空白が削除されます。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ができます。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ができます。

例：

```

<xs:schema ...>

    <xs:annotation>

        <xs:appinfo>

            <osd:documentation class="com.foo.MySchemaDocumentation">

                <param1>...</param1>

                <param2>...</param2>

            </osd:documentation>

        </xs:appinfo>
    </xs:annotation>

```

```

    </xs:annotation>

    ...

</xs:schema ...>

```

プログラムで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ているものよりも優先されます。

以下も参照してください。:

- `SchemeDocumentation`

列挙ラベル

列挙型では、属性 `osd:label` を使用して、ローカライズされていない単純なラベルを各列挙型エレメントに追加できます。

注意:

列挙型エレメントに定義されたラベルは、表示時に常に `折りたたみ` されます。

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>

```

```

    </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、ローカライズされていないラベルは、ラベルが定義されていないロケールで表示されます。

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>
```


必須のエラーメッセージ（osd:mandatoryErrorMessage）

ノードが属性 `minOccurs="1"`（デフォルトの動作）を指定している場合、ユーザーがフィールドに入力しないと、提供する必要の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、ノードごとに具体的に定義できます。

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、列挙ラベルは表示時に常に折りたたまれます。

変換エラーメッセージ

事前定義されたXMLスキーマエレメントごとに、ユーザーエントリ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することができます。

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は、カスタムの重大度、デフォルトの非ローカライズされたメッセージ、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。ブロッキング制約は必須の重大度 **エラー** を持っている必要があります。

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>
```

```

        </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` で定義されている）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
 - ビューでのグループとフィールドの非表示
 - 構造化検索ツールでフィールドを非表示にする
 - データサービスでのフィールドの非表示
 - 外部キーのコンボボックスセクタービューの定義
 - 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
 - データのフィルター
 - 事前定義されたカテゴリ
 - 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オカレンスを追加するときに、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、すなわち、特定のデータモデルノードのデータが読み取りおよび／または書き込み可能であるかどうかを定義します。この属性は、`RW`、`R-`、`CC` または `--` の値の一つを持たなければなりません。

XMLスキーマノードごとに、次の3種類の適応が可能です。

1. 適応ターミナルノード

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、適応検索アルゴリズムが使用されます。

2. 適応非ターミナルノード

このノードは、適応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. 非適応ノード

このノードは適応ターミナルノードではなく、子適応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>	適応ターミナルノード：値はEBX®で読み書きできます。
<code>R-</code>	適応ターミナルノード：値はEBX®でのみ読み取ることができます。
<code>CC</code>	<i>Cut</i> ：これは適応ターミナルノードではなく、その子はいずれも適応ターミナルノードではありません。この「命令」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。
<code>-</code>	ノードが単純なタイプの場合、適応できません。ノードが複合タイプの場合、それは適応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 <code>root</code> ノードは、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： ノードが計算値の場合、<code>R-</code> と見なされます。

アクセスモード	動作
	• ノードが単純型であり、その値が計算されていない場合、ノードは RW と見なされます。 • ノードが集約リストである場合、それは最終値であり、RW と見なされます。 • それ以外の場合、それは適応ターミナルノードではなく、その子ノードについて何も定義しません。

例：

この例では、エレメントは適応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` を使用して、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、特定のアクセス許可を定義せずに、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、関連するデータモデルの構造から生成された、表形式か階層形式かに関係なく、デフォルトのフォームとビューには表示されません。

注意:

- エレメント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構成されている場合、このフィールドに関連付けられているアクセス許可は評価されません。
- ビューを定義することにより、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することができます。この場合にのみ、このフィールドに関連付けられたアクセス許可が評価され、フィールドが表示されるかどうかが決まります。
- 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、エレメント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>

```

ビュー内で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、特定のアクセス許可を定義せずに、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、表形式か階層かを問わず、どのビューにも表示されません。ただし、ビューの非表示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を選択できません。結果として、エレメントはデータセット内のテーブルのすべてのビューに表示されるわけではありません。

グループがビューで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が、テーブルのビューにそれぞれ表示されるわけではありません。

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および入力された検索ツールでフィールドを選択することはできません。

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、型付き検索で選択可能になります。

注意:

グループが検索ツールまたはテキスト検索のみで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、検索ツールまたはテキスト検索のみにそれぞれ表示されません。

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により、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された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っています。

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Select操作で非表示になっています。

外部キーのコンボボックスセレクトビューの定義

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することができます。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、外部キーフィールドで詳細選択を使用することはできません。

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](#)を介してRESTデータサービスで管理されます。

比較モード

比較モードを設定するために、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差異を検出する方法を制御します。属性に使用できる値は次のとおりです。

default	エレメントは、その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、値が異なっても、コンテンツが更新され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がマージされます。 アーカイブのインポート中に、コンテンツが更新されたときに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が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も無視されます。
- 端末フィールドに属性 `osd:comparison` が含まれていない場合、比較時にデフォルトで含まれます。

制限事項:

- このプロパティは、非終端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、最初の要素は比較中に明示的に無視され、2番目の要素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` は、データセットのテーブルで実行できる「最後の変更を適用」サービスにこの要素を含める必要があるかどうかを指定するために、ターミナルノード要素で定義できます。

属性に使用できる値は次のとおりです。

default	最後の変更をこの要素に適用できます。
ignored	この要素は、最終変更の適用サービスから無視されます。つまり、この要素に対して実行された最後の変更を他のレコードに適用することはできません。

注意:

- グループが「最後の変更を適用」サービスによって無視されるように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も無視されます。
- 端末フィールドに属性 `osd:applyLastModification` が含まれていない場合、デフォルトで「最終変更の適用」サービスに含まれます。

制限事項:

- このプロパティは、非終端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まれています。

```
<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により、「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、属性 `osd:category` をデータモデルXSDのテーブルノードに追加します。

データのフィルター

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のデータセットで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**アクション>カテゴリ> *category name>***を選択します。

事前定義されたカテゴリ

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**アクション>カテゴリ> ****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っています。

- Constraint（制約）：非推奨

制限事項

カテゴリ「Hidden」を除いて、カテゴリは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** である必要があります。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、ツールバードキュメントは 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。

関連する概念:

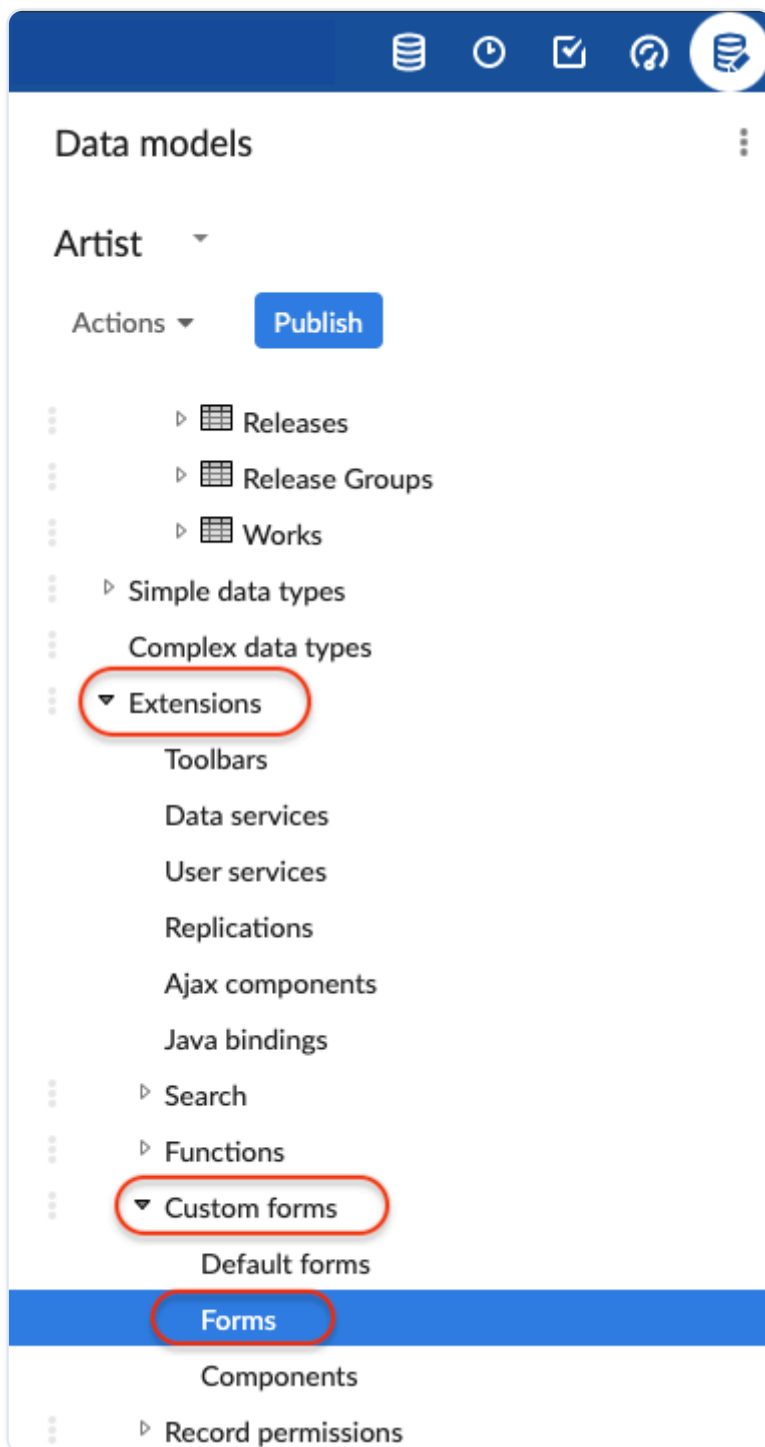
- インターフェイスのカスタマイズ
- アクセス
- フォームとコンポーネント
- エディター
- ブロック
 - 算術演算子
 - Array（配列）
 - 配列アイテム
 - ブール演算子
 - Check box（チェックボックス）
 - Check box group（チェックボックスグループ）
 - Children（子）
 - Choose（選択）
 - Close button（閉じるボタン）
 - Color picker（カラーピッカー）
 - Combo box（コンボボックス）

- Comparator（コンパレータ、比較）
- 比較
- 配列の連結
- 定数
- コンテンツタイトル
- Creating record?（レコードを作成しますか？）
- Custom action bar（カスタムアクションバー）
- Date time input（日時入力）
- Declare（宣言）
- Default action bar（デフォルトアクションバー）
- Default toolbar（デフォルトツールバー）
- Duplicating record?（レコードを複製しますか？）
- Expand/Collapse（展開/折りたたみ）
- Flow（フロー）
- For each
- Form（フォーム）
- Form definition（フォーム定義）
- Form group（フォームグループ）
- Form row（フォーム行）
- Form parameter（フォームパラメーター）
- From...get（から...取得）
- Function（関数）
- Function call（関数呼び出し）
- Function call argument（関数呼び出し引数）
- Function parameter（関数パラメーター）
- Grid（グリッド）
- Grid element（グリッドエレメント）
- If
- Input parameter（入力パラメーター）
- In role?（役割は？）

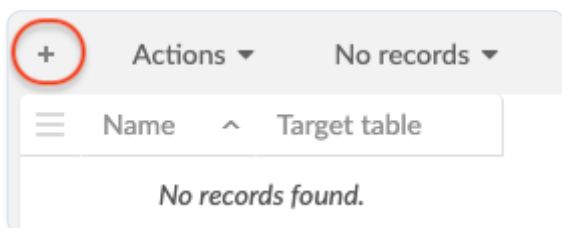
- Label（ラベル）
- Localized（ローカライズ）
- Message（メッセージ）
- Next button（次へボタン）
- Node（ノード）
- Nomenclature item（命名項目）
- Not
- Object（オブジェクト）
- Otherwise（それ以外の場合）
- Paragraph title（段落タイトル）
- パスワード
- Previous button（前のボタン）
- Property（プロパティ）
- Radio button（ラジオボタン）
- Radio button group（ラジオボタングループ）
- Revert button（元に戻すボタン）
- Save button（保存ボタン）
- Save and close button（保存して閉じるボタン）
- Tab（タブ）
- Tab container（タブコンテナ）
- Text input（テキスト入力）
- Toolbar named（名前の付いたツールバー）
- Variable（変数）
- When
- Widget（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、使用できるテーブルを示します。

Name AlbumForm

Target table * /root/artist

「保存」 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「カスタムフォーム」データモデル拡張で作成できます。

- **フォームはレコードのレイアウトを記述します。*フォームは、レコードや入力パラメータ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とき、ユーザーはそのターゲットテーブルを提供するように求められます。作成されると、**テーブルのデフォルトフォームとして宣言**できます。拡張機能の「デフォルトフォーム」テーブル、またはターゲットテーブルの「拡張機能」タブのいずれかで宣言できます。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。

- **コンポーネントは再利用可能なフラグメントです。**フォームと他のコンポーネント間で共有できます。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。このような情報が必要な場合は、呼び出し元から明示的に提供する必要があります。

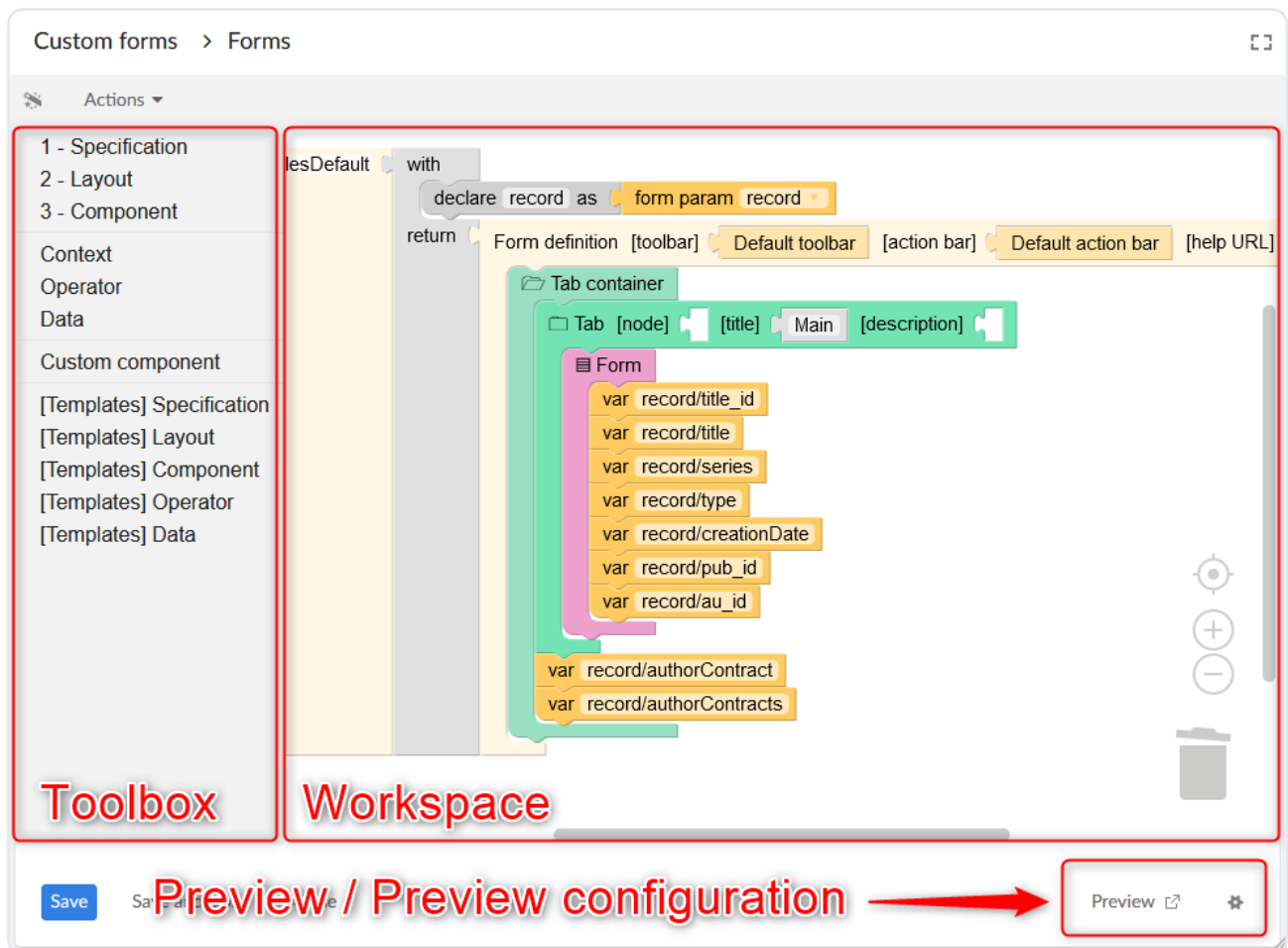
エディター

フォームまたはコンポーネントは、ブロックの積み重ねに過ぎません。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、言語などの条件に基づいて異なるレイアウトを持つことができます。

ワークスペースには、事前定義された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、非アクティブとしてマークするためにグレー表示されます。 **ブロックを移動するには**、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。ブロックを1つだけ移動する必要がある場合は、Ctrlキーを押しながらクリックします。ワークスペースのブロックを**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リストも表示されます。

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「テンプレート」セクションがあります。このセクションは、セクションのブロックのいくつかの組み合わせを提供し、便利なショートカットまたはサンプルと見なすことができます。

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は次のとおりです。

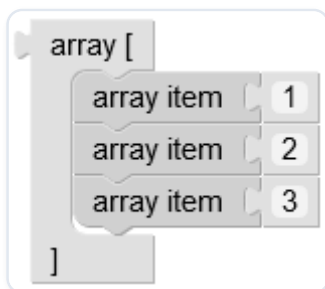
算術演算子

2つの整数の演算。



Array(配列)

アイテムのリスト。



以下も参照してください。:

- 配列アイテム

配列アイテム

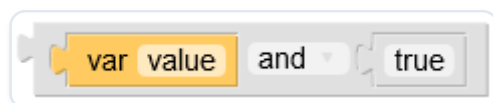
式ブロックを配列にするためのアダプター。

以下も参照してください。:

- Array (配列)

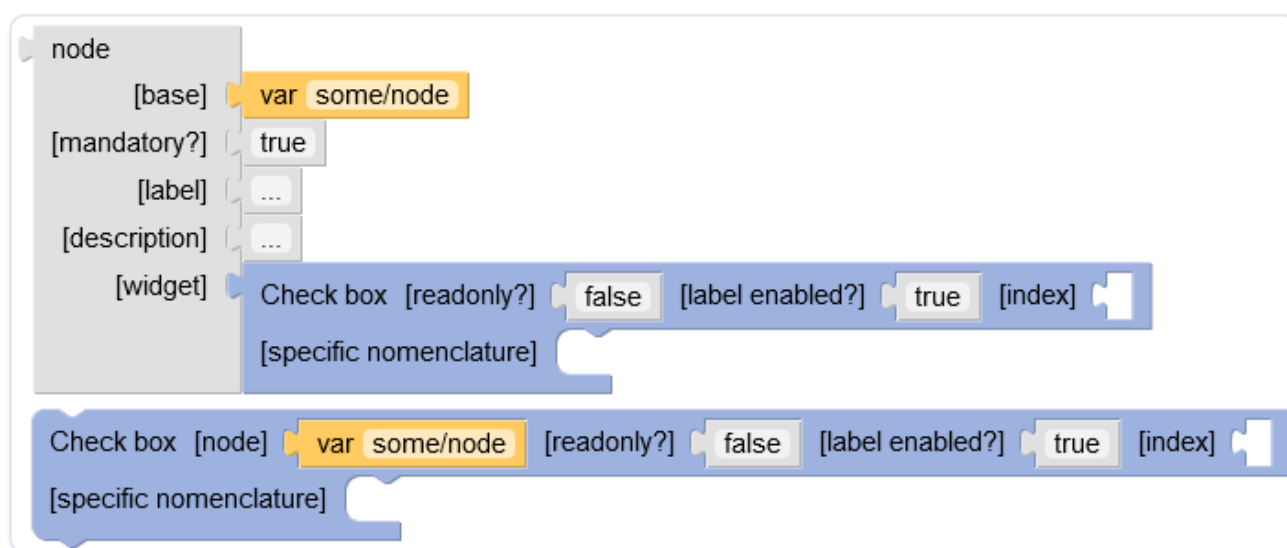
ブール演算子

2つのブール値の操作。



Check box (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。

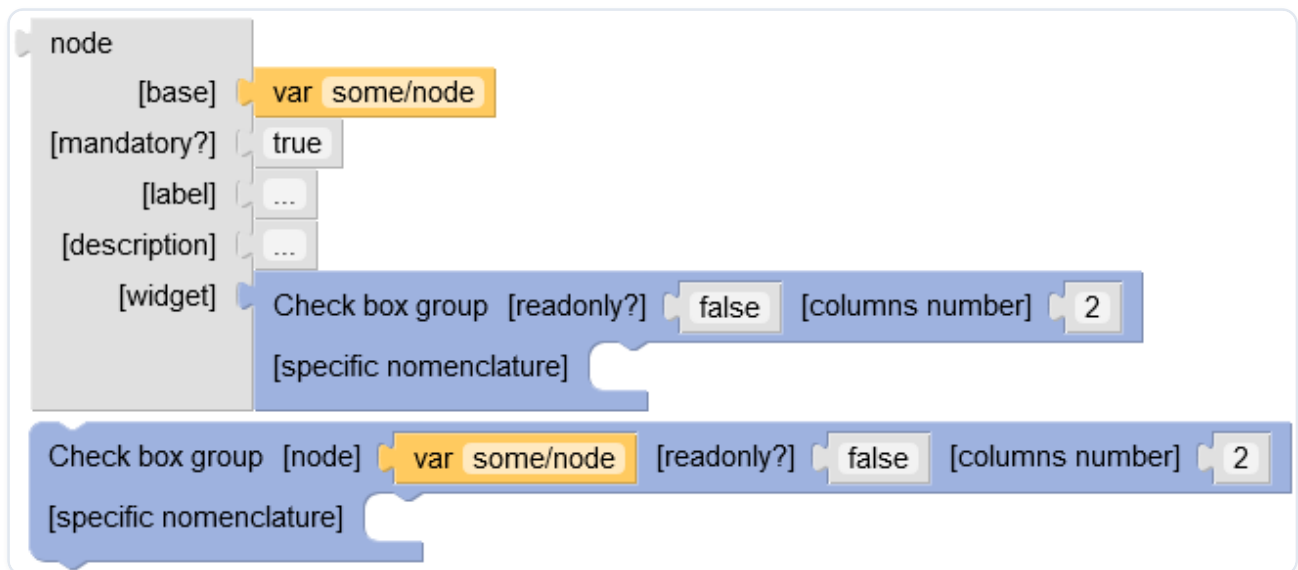


パラメーター

入力	説明
node	ウィジェットを表示するノード。
readonly?	ウィジェットを読み取り専用モードにする必要があるかどうかを示します。
label enabled?	設定されている場合、ウィジェットの横にアイテムラベルを追加するかどうかを示します。
index	この列挙項目のインデックス。
specific nomenclature	設定されている場合、モデル駆動型の命名法を上書きします。

Check box group (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
node	ウィジェットを表示するノード。
readonly?	ウィジェットを読み取り専用モードにする必要があるかどうかを示します。

入力	説明
columns number	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。
specific nomenclature	設定されている場合、モデル駆動型の命名法を上書きします。

Children（子）

指定されたノードの子のリストを返します。

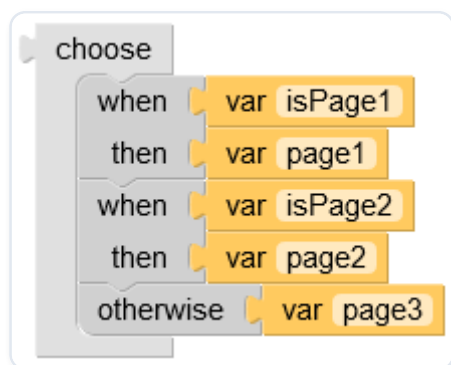


パラメーター

入力	説明
node	子を抽出するための複雑なノード。

Choose（選択）

「when」 / 「otherwise」 ブロックの内側の1つのコンテンツを返すブロック。



パラメーター

入力	説明
content	「when」ブロックのリスト、および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
- Otherwise（それ以外の場合）

Close button（閉じるボタン）

標準の「閉じる」ボタン。

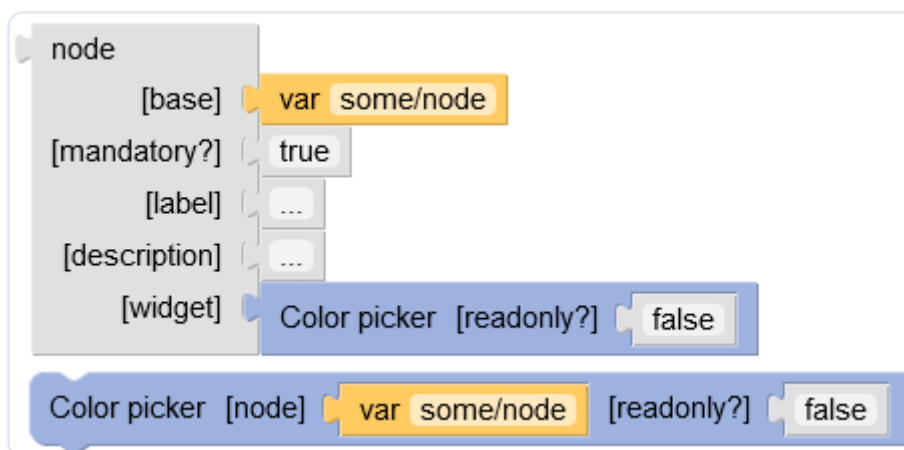
Close button [label] [default?] false

パラメーター

入力	説明
label	設定されている場合、デフォルトのラベルを上書きします。
default?	これが「Enter」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。

Color picker（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。

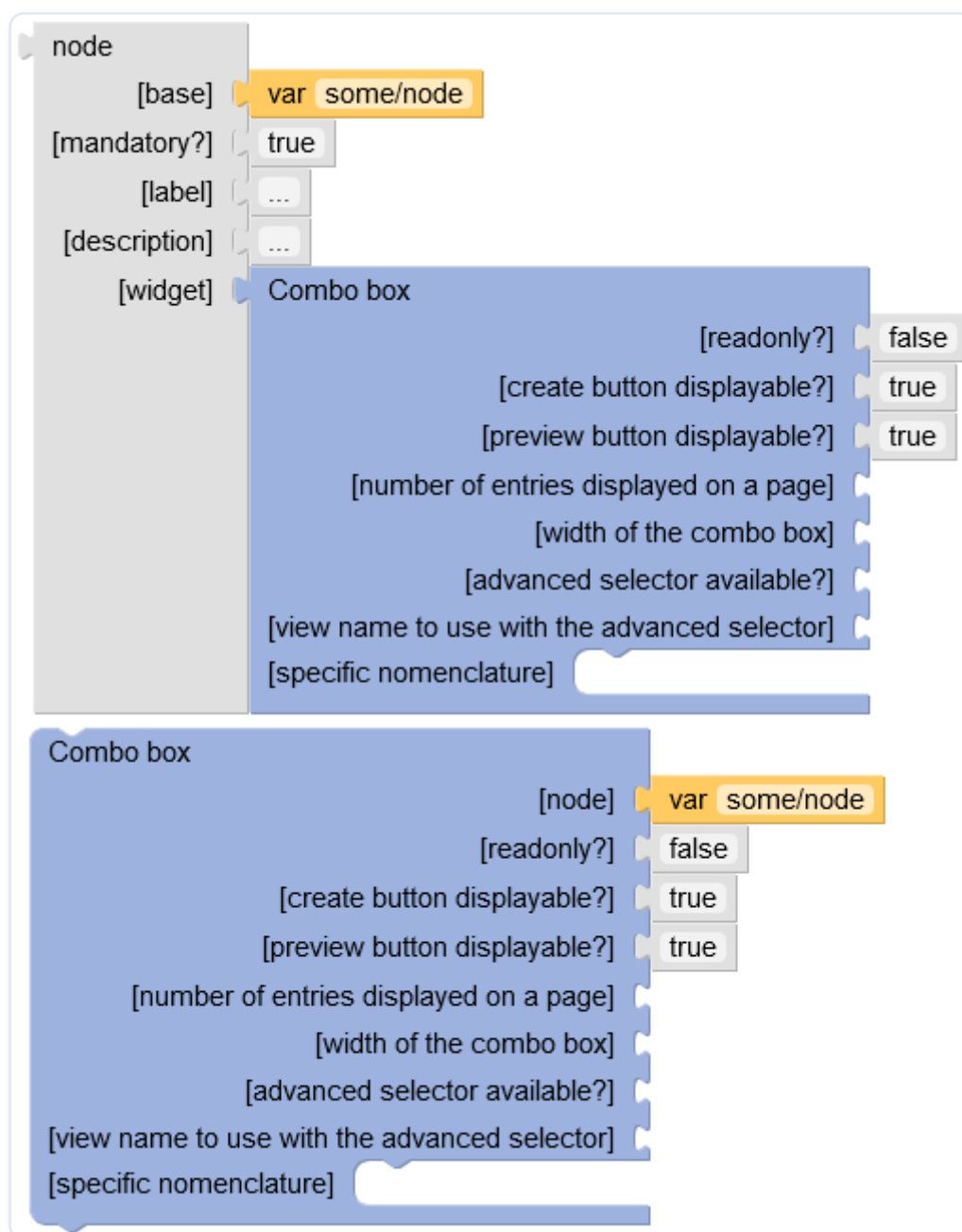


パラメーター

入力	説明
node	ウィジェットを表示するノード。
readonly?	ウィジェットを読み取り専用モードにする必要があるかどうかを示します。

Combo box（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。

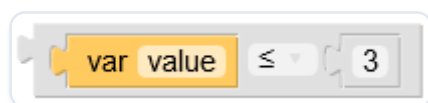


パラメーター

入力	説明
<code>node</code>	ウィジェットを表示するノード。
<code>readonly?</code>	ウィジェットを読み取り専用モードにする必要があるかどうかを示します。
<code>create button displayable?</code>	設定されている場合、基になるノードが外部キーであるときに作成ボタンを表示するかどうかを定義します。
<code>preview button displayable?</code>	設定されている場合、基になるノードが外部キーであるときにプレビューボタンを表示するかどうかを定義します。
<code>number of entries displayed on a page</code>	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数を定義します。
<code>width of the combo box</code>	設定されている場合、コンボボックスの幅を定義します。
<code>advanced selector available?</code>	設定されている場合、基になるノードが外部キーであるときに詳細セクターを表示するかどうかを定義します。
<code>view name to use with the advanced selector</code>	設定されている場合、関連する外部キーフィールド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定義します。
<code>specific nomenclature</code>	設定されている場合、モデル駆動型の命名法を上書きします。

Comparator（コンパレータ、比較）

2つの整数を比較します。



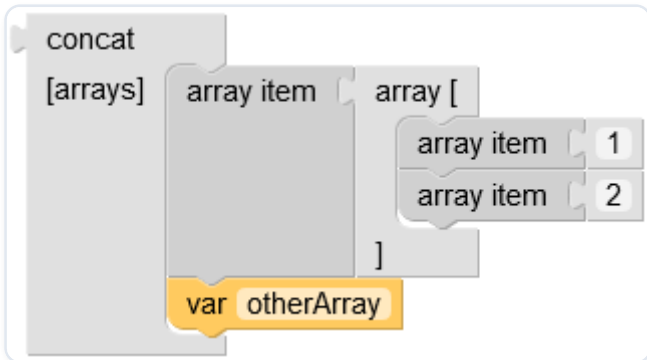
比較

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。

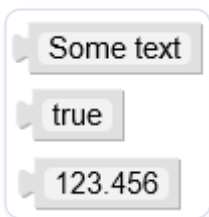


パラメーター

入力	説明
arrays	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
value	テキスト、ブール値、または数値。

コンテンツタイトル

ワークスペースのコンテンツ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
label	表示するテキスト。
horizontal alignment	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。

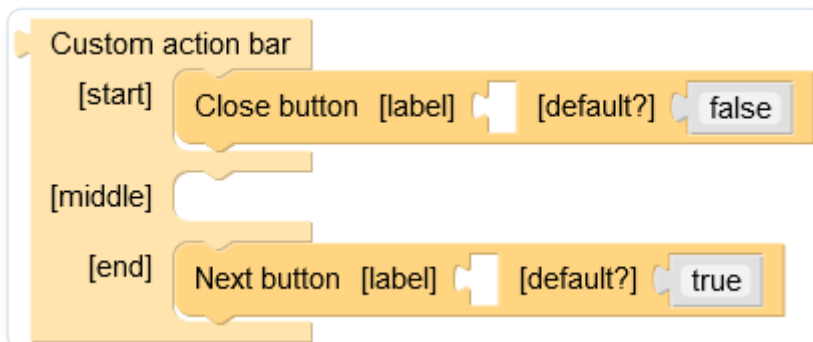
Creating record?（レコードを作成しますか？）

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?

Custom action bar（カスタムアクションバー）

カスタムアクションバー。



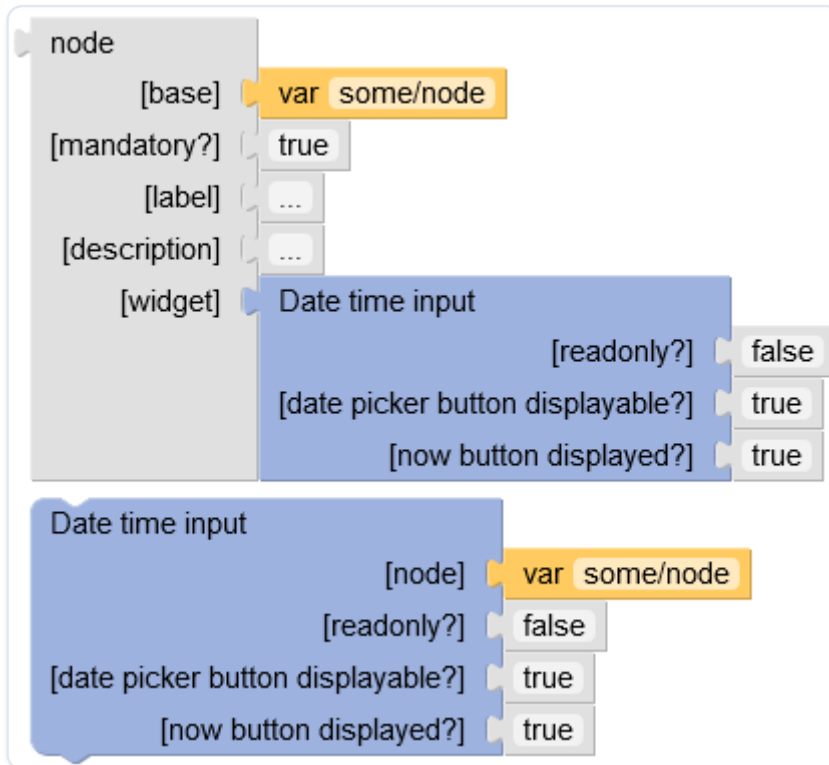
パラメーター

入力	説明
start	アクションバーの左側に表示する「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する「ボタン」のリスト。

Date time input（日時入力）

入力した日付/時刻を表示します。



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。
readonly?	ウィジェットを読み取り専用モードにする必要があるかどうかを示します。
date picker button displayable?	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）。
now button displayed?	日付を現在時刻に設定する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）。

Declare（宣言）

「with」ブロックでの変数の宣言。

```
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である必要があります。
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](#)

Default action bar（デフォルトアクションバー）

デフォルトのアクションバー。

```
Default action bar
```

Default toolbar（デフォルトツールバー）

モデル駆動型ツールバー。

```
Default toolbar
```

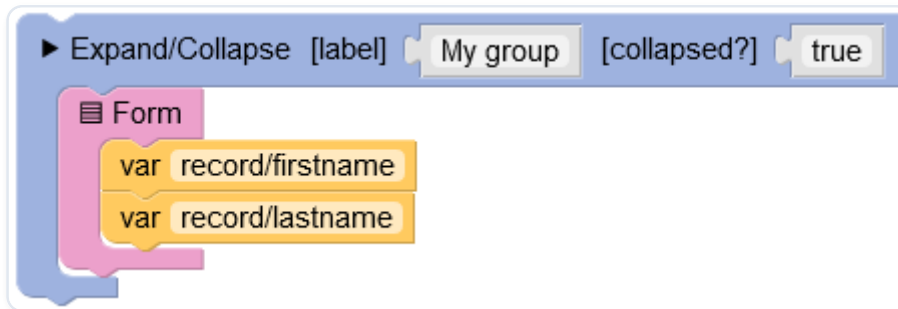
Duplicating record?（レコードを複製しますか？）

フォームがレコード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?
```

Expand/Collapse（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。

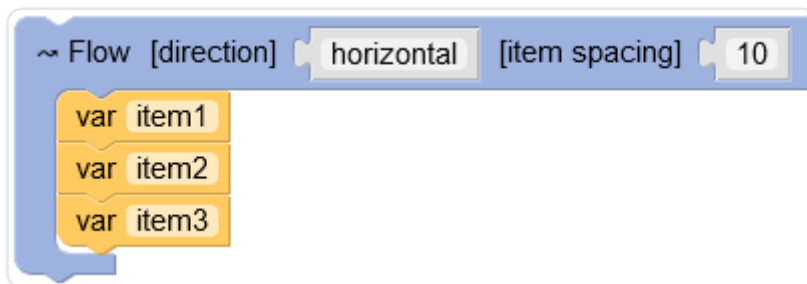


パラメーター

入力	説明
label	矢印の横に表示されるラベル。
collapsed?	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。

Flow（フロー）

その内容を流動的に表示します。エレメントが1つの行または列に収まらない場合、エレメントは折り返され、新しいエレメントが開始されます。



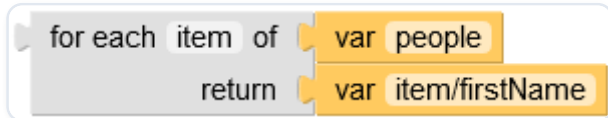
パラメーター

入力	説明
direction	「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。
item spacing	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。

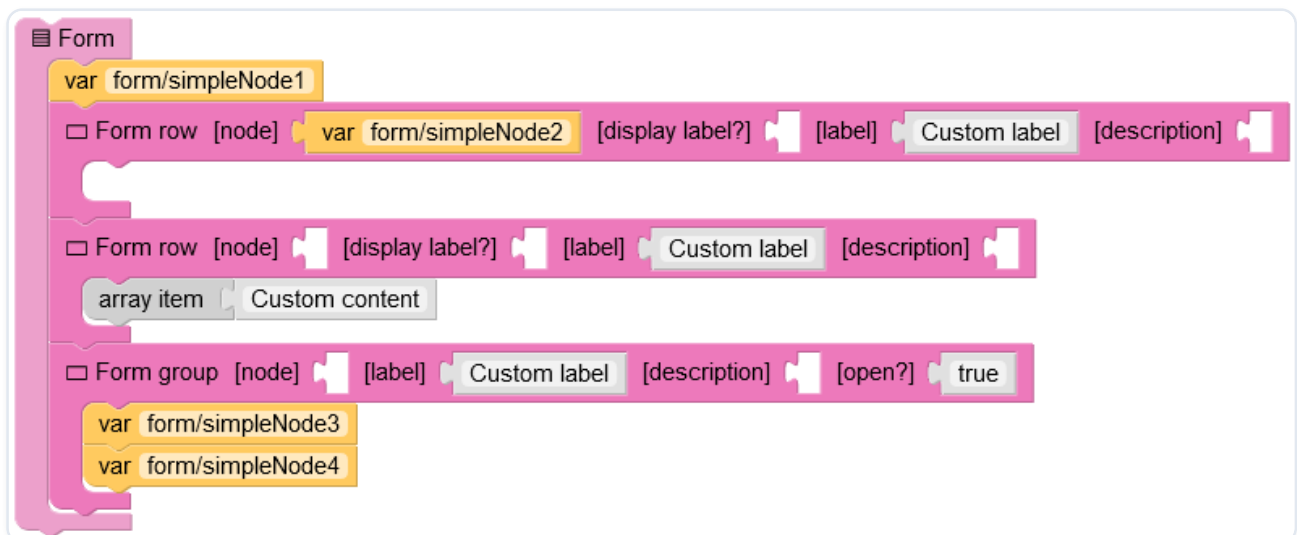


パラメーター

入力	説明
item	現在のアイテムを含む変数の名前。
array	変換するアイテムを含む配列。
return	現在のアイテムに対して何を返すか。

Form（フォーム）

左側にラベル、右側に値がある標準のテーブルのようなレイアウト。



パラメーター

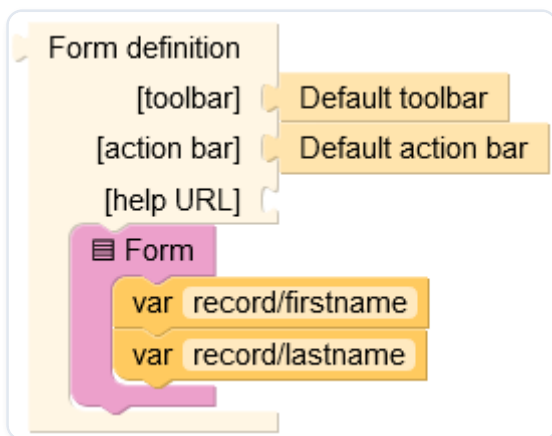
入力	説明
<code>content</code>	「フォーム行」、「フォームグループ」またはノードのリスト。ノードはモデル駆動型の表示になり、「フォーム行」と「フォームグループ」でカスタムコンテンツを表示できます。

以下も参照してください。:

- [フォーム行](#)
- [フォームグループ](#)

Form definition (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
<code>toolbar</code>	フォームの上部に表示されるツールバーの定義。
<code>action bar</code>	フォームの下に表示されるアクションバーの定義。
<code>help URL</code>	ヘルプページのURL。

入力	説明
<code>content</code>	レイアウトの内容。

Form group（フォームグループ）

「フォーム」内に、折りたたみ可能なアイテムのグループを表示します。



パラメーター

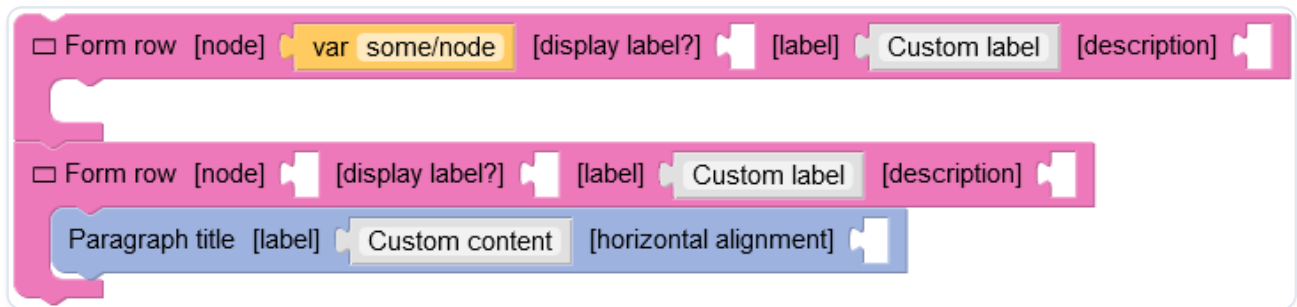
入力	説明
<code>node</code>	設定されている場合、グループラベル、説明、およびコンテンツはモデル駆動型になります。
<code>label</code>	グループのラベル。ノードとラベルが設定されている場合、モデル駆動ラベルをオーバーライドできます。
<code>description</code>	グループの説明。ノードと説明が設定されている場合、モデル駆動型の説明を上書きできます。
<code>open?</code>	グループが最初に開いているかどうかを示します。
<code>content</code>	「フォーム行」、「フォームグループ」またはノードのリスト。ノードはモデル駆動型の表示になり、「フォーム行」と「フォームグループ」で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

Form row（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を表示します。



パラメーター

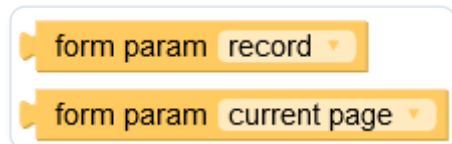
入力	説明
node	設定されている場合、ラベル、説明、およびコンテンツはモデル駆動型になります。
display label?	ラベルを表示するかどうかを示します。 「フォーム」でfalseに設定すると、コンテンツはラベルとコンテンツ領域の両方にまたがります。
label	行のラベル。ノードとラベルが設定されている場合、モデル駆動ラベルをオーバーライドできます。
description	行の説明。ノードとラベルが設定されている場合、モデル駆動ラベルをオーバーライドできます。
content	ラベルの横に表示されるコンテンツ。

以下も参照してください。:

- フォーム
- フォームグループ

Form parameter（フォームパラメーター）

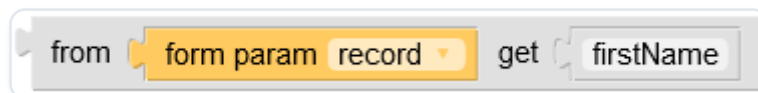
「レコード」は、表示されたレコードを表すノードです。「現在のページ」は、表示されているページを表す番号（デフォルトでは1）です。この値は、ボタンとボタンで変更できます。



From...get（から...取得）

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。

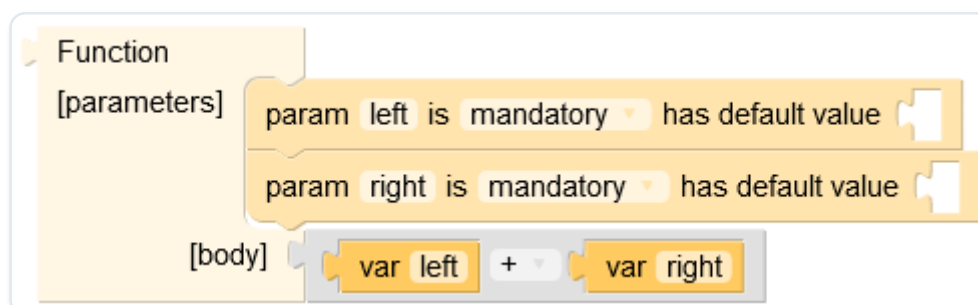


パラメーター

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」のオブジェクト内で抽出するデータのパス。

Function（関数）

無名関数の定義。関数は、本体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

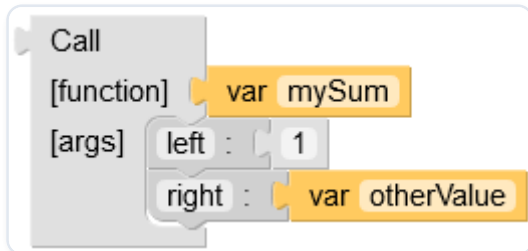
入力	説明
parameters	「param」ブロックのリスト。この関数で受け入れられるパラメーターを定義します。
body	パラメーターを使用して、関数の結果を説明する本文。

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

Function call (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
function	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

Function call argument（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
name	パラメーターの名前。
value	パラメーターを渡すための実際の値。

以下も参照してください。:

- 関数呼び出し

Function parameter（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

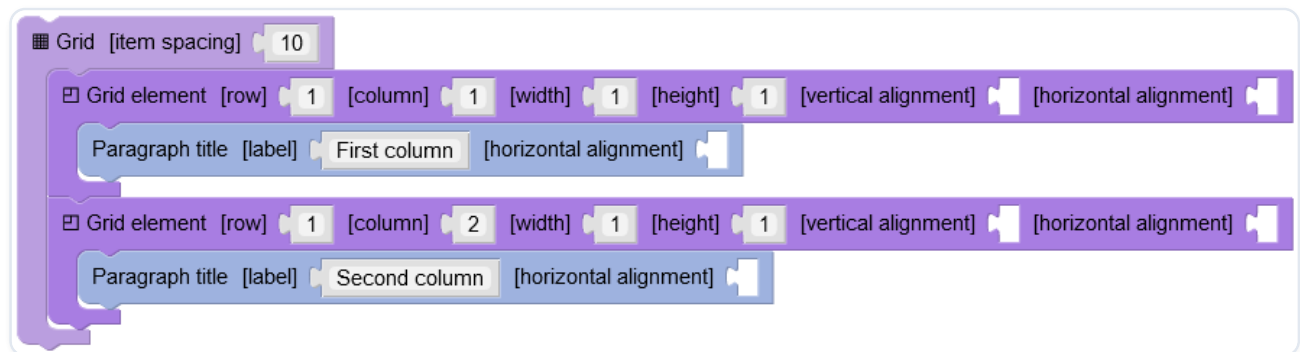
入力	説明
name	パラメーターの名前。
mandatory?	このパラメーターを呼び出し元が指定する必要があるかどうかを示します。
default value	設定されている場合、これは、呼び出し元が指定しない場合にこのパラメーターがとるデフォルト値です。

以下も参照してください。:

- 関数

Grid (グリッド)

座標を使用してコンポーネントをグリッドに配置できるレイアウト。



パラメーター

入力	説明
<code>item spacing</code>	エレメント間のスペース（ピクセル単位）。
<code>content</code>	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

Grid element (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

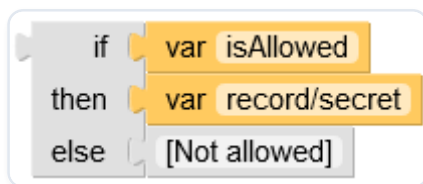
入力	説明
row	エレメントが始まる行。最小値は1です。
column	エレメントが始まる列。最小値は1です。
width	グリッドセルでのエレメントの幅。最小値は1です。
height	グリッドセルでのエレメントの高さ。最小値は1です。
vertical alignment	エレメントの垂直方向の配置。設定されている場合、受け入れられる値は「start」、「end」、「center」です。
horizontal alignment	エレメントの水平方向の配置。設定されている場合、受け入れられる値は「start」、「end」、「center」です。
content	セル内のコンテンツ。

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」のコンテンツまたは「else」のコンテンツのいずれかを返すブロック。



パラメーター

入力	説明
test	返す値を決定するテスト。これはブール値である必要があります。
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。

Input parameter（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
name	パラメーター名。

In role？（役割は？）

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]

Label（ラベル）

指定されたノードのラベルを表示します。

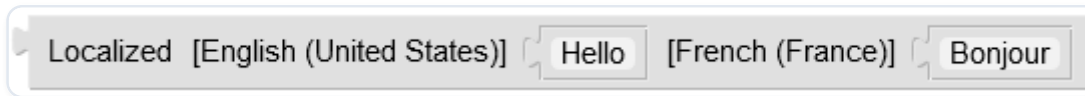
Label [node] var record/firstName

パラメーター

入力	説明
node	ラベルを表示するノード。

Localized（ローカライズ）

どの結果がユーザーロケールに依存するかをブロック化します。



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。ブロック化の結果は、現在のユーザーのロケールに対応するものになります。

Message（メッセージ）

javaのような{0}、{1}などのプレースホルダーを指定されたインデックスの引数に置き換えることにより、指定されたテキストをフォーマットします。



パラメーター

入力	説明
pattern	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。
arguments	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。

Next button（次へボタン）

「現在のページ」の値を1つインクリメントするボタン。

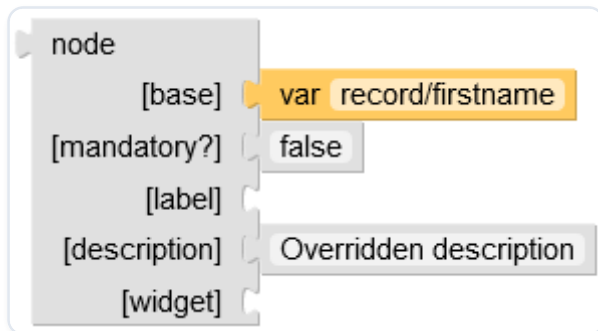


パラメーター

入力	説明
label	設定されている場合、デフォルトのラベルを上書きします。
default?	これが「Enter」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。

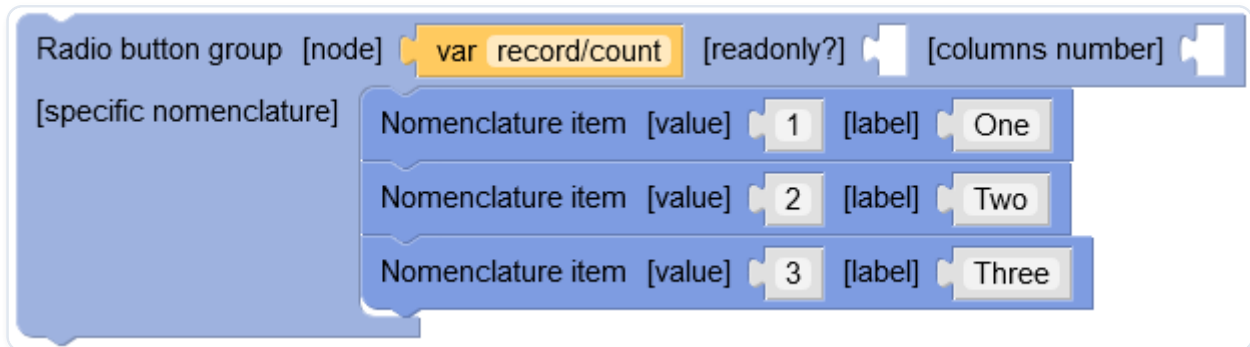


パラメーター

入力	説明
base	オーバーライドするノード。
mandatory?	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オーバーライドします。
label	設定されている場合、ノードのラベルを上書きします。
description	設定されている場合、ノードのラベルを上書きします。
widget	設定されている場合、ノードのデフォルトウィジェットをオーバーライドします。

Nomenclature item（命名項目）

命名項目は（キー、ラベル）ペアです。

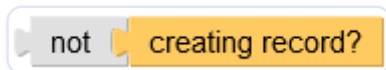


パラメーター

入力	説明
value	アイテムの値。
label	アイテムのラベル。

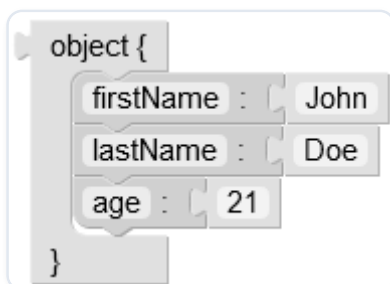
Not

指定されたブール値の逆数を返します。



Object（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

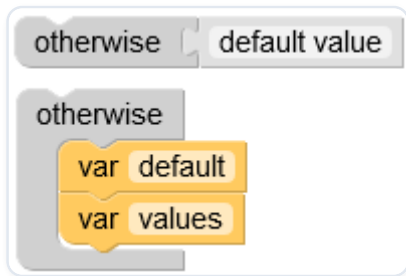
入力	説明
content	「property」ブロックのリスト。

以下も参照してください。:

- プロパティ

Otherwise（それ以外の場合）

「choose」ブロックの最終選択。大文字と小文字が一致しない場合に返されます。



パラメーター

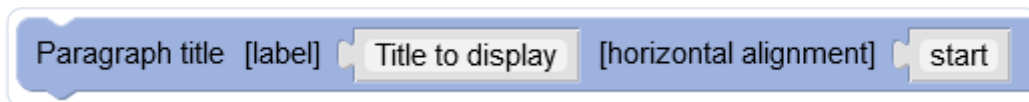
入力	説明
content	「When」 ケースとの一致がない場合に返す値。

以下も参照してください。:

- Choose（選択）

Paragraph title（段落タイトル）

段落のタイトルを表示します。



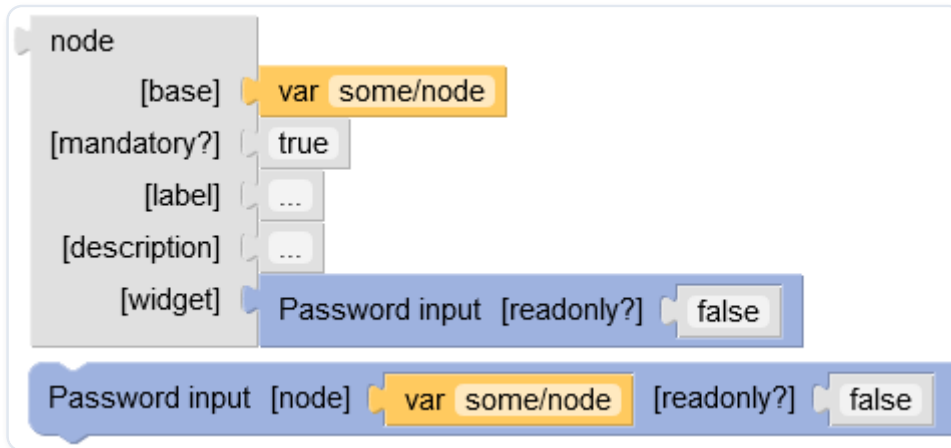
パラメーター

入力	説明
label	表示するテキスト。

入力	説明
horizontal alignment	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。
readonly?	ウィジェットを読み取り専用モードにする必要があるかどうかを示します。

Previous button(前のボタン)

「現在のページ」の値を1つ減らすボタン。



パラメーター

入力	説明
label	設定されている場合、デフォルトのラベルを上書きします。

入力	説明
default?	これが「Enter」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。

Property（プロパティ）

オブジェクトのプロパティ。

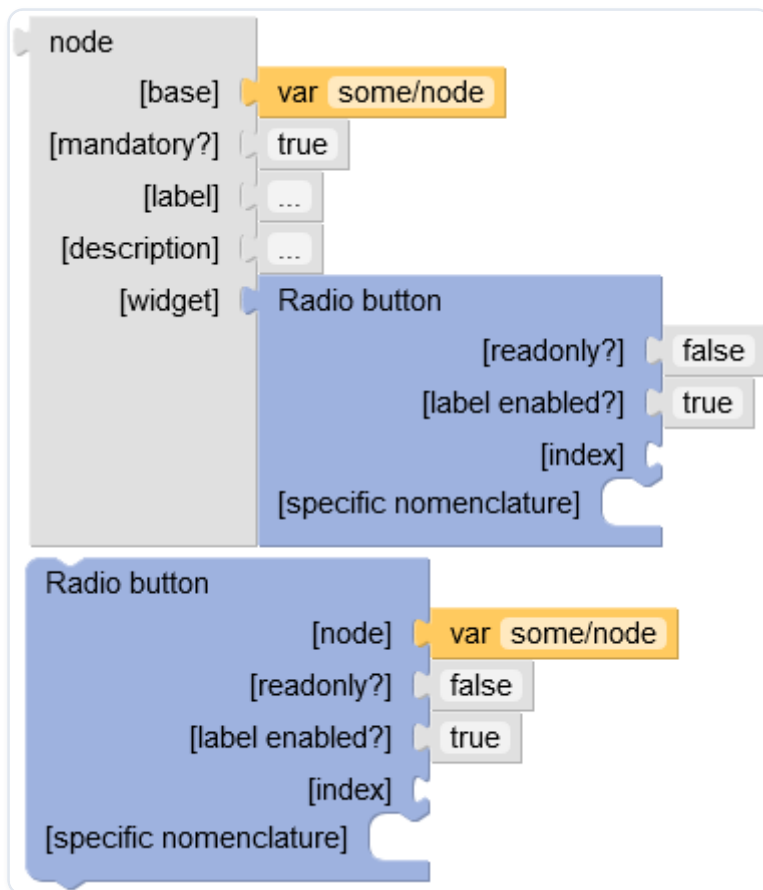


パラメーター

入力	説明
name	プロパティの名前。特定のオブジェクトに対して一意である必要があります。
value	プロパティの値。

Radio button（ラジオボタン）

ラジオボタンウィジェットを表示します。

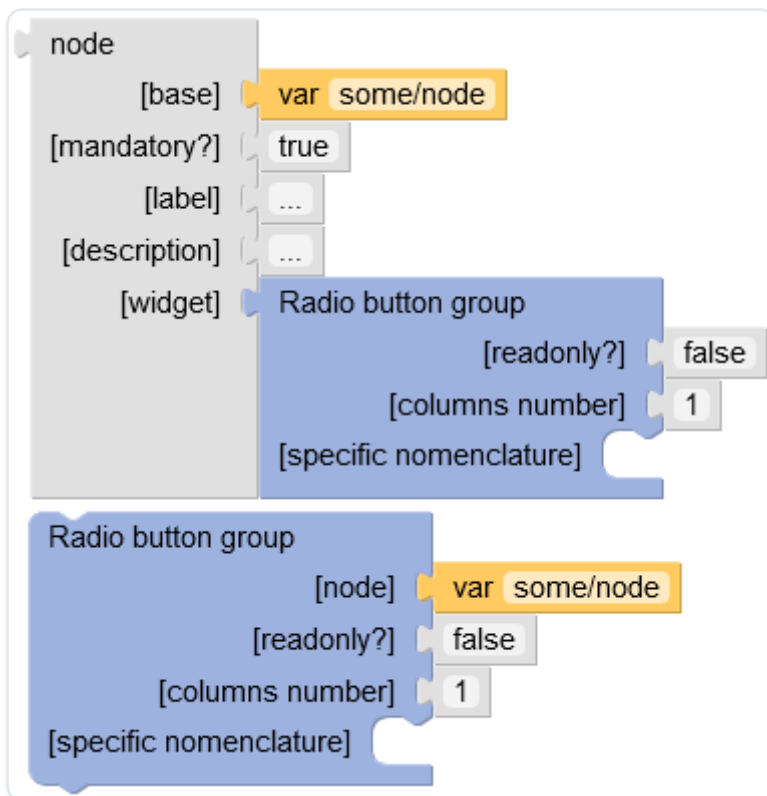


パラメーター

入力	説明
node	ウィジェットを表示するノード。
readonly?	ウィジェットを読み取り専用モードにする必要があるかどうかを示します。
label enabled?	設定されている場合、ウィジェットの横にアイテムラベルを追加するかどうかを示します。
index	この列挙項目のインデックス。
specific nomenclature	設定されている場合、モデル駆動型の命名法を上書きします。

Radio button group (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。
readonly?	ウィジェットを読み取り専用モードにする必要があるかどうかを示します。
columns number	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。
specific nomenclature	設定されている場合、モデル駆動型の命名法を上書きします。

Revert button (元に戻るボタン)

標準の「元に戻る」ボタン。



パラメーター

入力	説明
label	設定されている場合、デフォルトのラベルを上書きします。
default?	これが「Enter」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。

Save button（保存ボタン）

標準の「保存」ボタン。

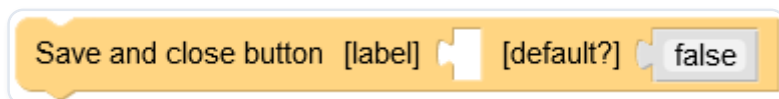


パラメーター

入力	説明
label	設定されている場合、デフォルトのラベルを上書きします。
default?	これが「Enter」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。

Save and close button（保存して閉じるボタン）

標準のボタン。

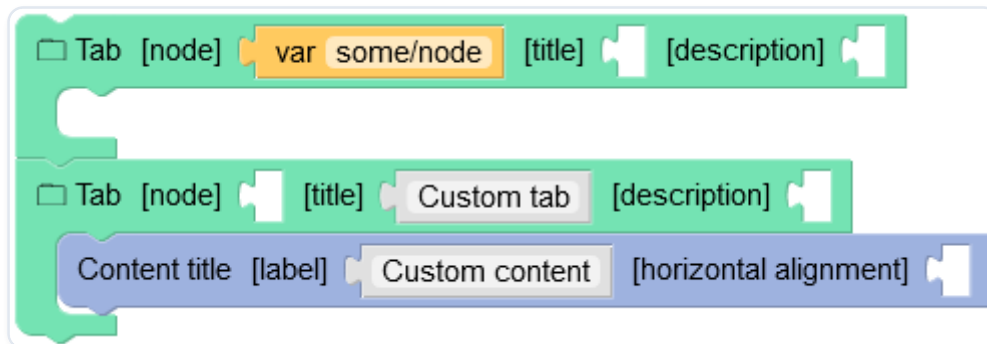


パラメーター

入力	説明
label	設定されている場合、デフォルトのラベルを上書きします。
default?	これが「Enter」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。

Tab (タブ)

「タブコンテナ」内のタブ。



パラメーター

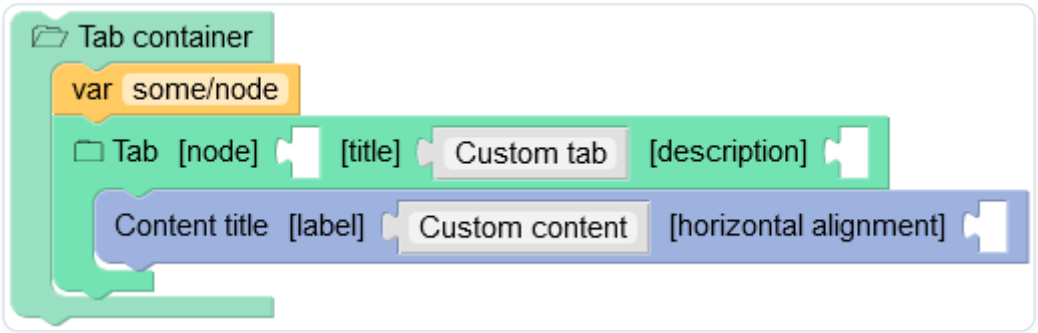
入力	説明
node	存在する場合、タブのタイトル、説明、およびコンテンツはモデル駆動型になります。
title	タブのタイトル。ノードとタイトルが設定されている場合、モデル駆動型タイトルをオーバーライドできます。
description	タブの説明。ノードと説明が設定されている場合、モデル駆動型の説明を上書きできます。
content	タブの内容。ノードとコンテンツが設定されている場合、モデル駆動型コンテンツをオーバーライドする可能性があります。

以下も参照してください。:

- [Tab container \(タブコンテナ\)](#)

Tab container (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

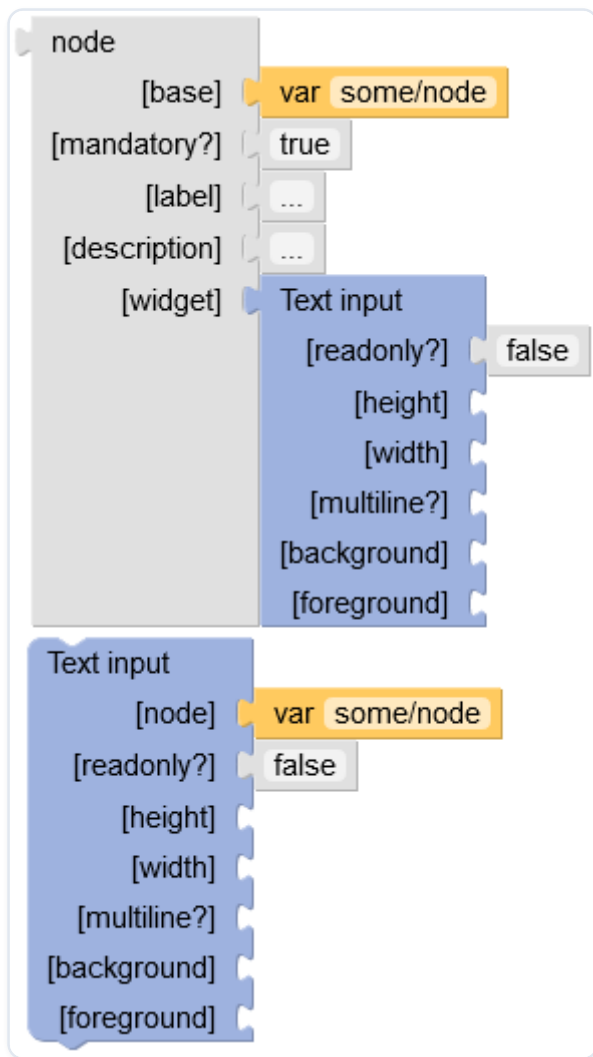
入力	説明
<code>content</code>	「タブ」のリスト。

以下も参照してください。:

- [Tab \(タブ\)](#)

Text input (テキスト入力)

テキスト入力ウィジェットを表示します。



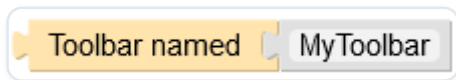
パラメーター

入力	説明
<code>node</code>	ウィジェットを表示するノード。
<code>readonly?</code>	ウィジェットを読み取り専用モードにする必要があるかどうかを示します。
<code>height</code>	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。
<code>width</code>	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。
<code>multiline?</code>	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
background	設定されている場合、16進形式で背景色を定義します。
foreground	設定されている場合、前景色を16進形式で定義します。

Toolbar named（名前の付いたツールバー）

指定された名前のツールバー。

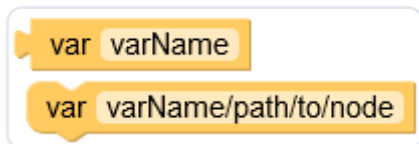


パラメーター

入力	説明
name	表示するツールバーの名前。

Variable（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の場合も、パスが受け入れられます。

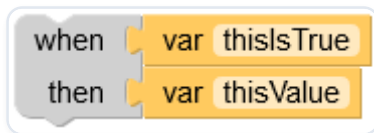


パラメーター

入力	説明
expression	「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の場合も、パスが受け入れられます。

When

「choose」ブロックでの選択。テストがtrueの場合、「then」コンテンツが返されます。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

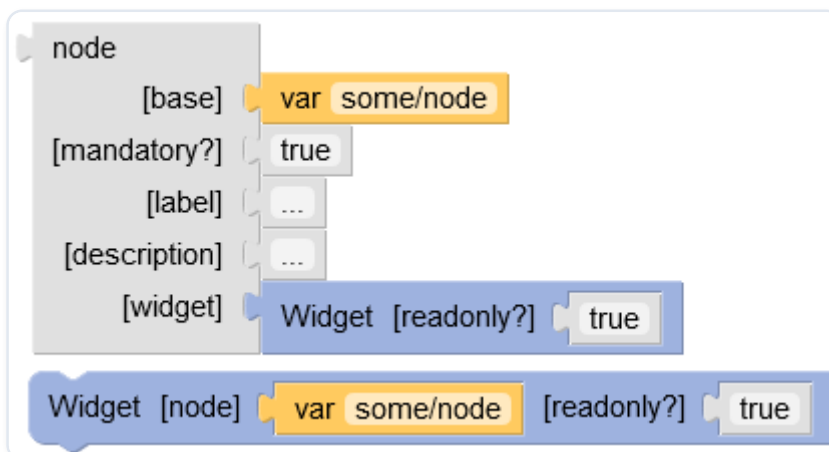
入力	説明
when	このケースを解決する必要があるかどうかを示すブールテスト。
then	テストがtrueの場合、この値が返されます。

以下も参照してください。:

- [Choose \(選択\)](#)

Widget (ウィジェット)

デフォルトのウィジェットを表示します。

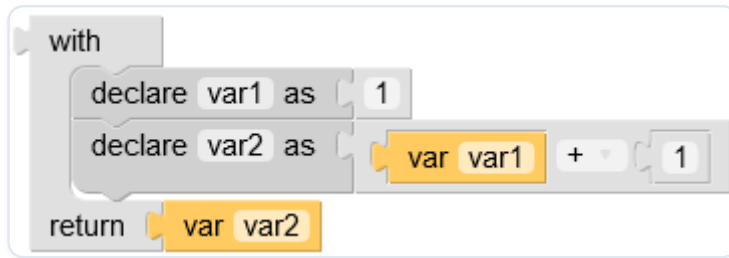


パラメーター

入力	説明
node	ウィジェットを表示するノード。
readonly?	ウィジェットを読み取り専用モードにする必要があるかどうかを示します。

With

「return」ステートメントで使用できる変数を宣言します。



パラメーター

入力	説明
<code>variables</code>	「declare」ブロックのリスト。
<code>return</code>	返す値。「variables」ステートメントで宣言された変数は、ここで「var」ブロックとともに使用できます。

以下も参照してください。:

- [宣言](#)

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」の2種類のステップがあります。

「ライブラリ」は `module.xml` で定義されている Bean であり、再利用できます。'library' Bean を使用すると、人間工学が向上します。パラメーターは定義画面に動的に表示されます。

「特定の」オブジェクトは、そのクラス名によってのみ定義される Bean です。この場合、表示は動的ではありません。

- [Beanカテゴリ](#)
- [ScriptTaskの例](#)
 - [Javaコード](#)
- [ScriptTaskBeanの例](#)

- Javaコード
- module.xmlによる構成
- UserTaskの例
 - module.xmlによるサービス宣言
 - より複雑なUserTask
- 条件の例
 - Javaコード
- ConditionBeanの例
 - Javaコード
 - module.xmlによる構成
- SubWorkflowsInvocationBeanの例
 - Javaコード
 - module.xmlによる構成
- WaitTaskBeanの例
 - Javaコード
 - module.xmlによる構成
- ActionPermissionsOnWorkflowの例
 - Javaコード
 - module.xmlによる構成
- WorkflowTriggerBeanの例
 - 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例
 - 例

Beanカテゴリ

ステップ	ライブラリ	スペシフィック
スクリプト	ScriptTaskBean	ScriptTask
条件	ConditionBean	Condition
ユーザータスク	UserTask	

ScriptTaskの例

Javaコード

次の例のように、スクリプトタスクはメソッド `execute` をオーバーライド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の例

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` をオーバーライド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の例

module.xmlによるサービス宣言

ビルトインサービスを `module.xml` で宣言し、ユーザータスクの定義で使うことができます。

```

<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>

    <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よく似ています。「UserTask」を拡張して呼び出すクラスを定義するには、フィールド「Rule」に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](#)

条件の例

Javaコード

メソッド `evaluate` はオーバーライド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の例

Javaコード

次の例のように、メソッド `EvaluationCondition` をオーバーライド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の例

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").equals("true")

        &&
        validateProductDirection.getDataContext().getVariableString("Accepted").equals(

            "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
        className="com.orchestranetworks.workflow.test.MySubWorkflowsInvocationBean"/>

    </beans>

```

</module>

WaitTaskBeanの例

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の例

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の例

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 OperationException

    {

        DisplayPolicy policy =
        DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

        MailSpec spec = aContext.createMailSpec();

        spec.notify(NotificationType.TO, "supervisor@mail.com");

```

```

        spec.setSubject("[TRIGGER] After work item creation");

        WorkItem workItem = aContext.getWorkItem();

        State state = workItem.getState();

        String body = "The work item '" +
policy.formatUserMessage(workItem.getLabel())

                + "'" has been created. \n The step id is : " +
aContext.getCurrentStepId()

                + ". \n The work item is in state : " +
policy.formatUserMessage(state.getLabel());

        if (workItem.getOfferedTo() != null)

            body += "\n The role is :" + workItem.getOfferedTo().format();

        if (workItem.getUserReference() != null)

            body += "\n The user is :" +
workItem.getUserReference().format();

        spec.setBody(body);

        spec.sendMail(Locale.US);

    }

    @Override

    public void
handleBeforeWorkItemStart(WorkflowTriggerBeforeWorkItemStartContext
aContext)

```

```

        throws RuntimeException

    {

        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.createMailSSec();

        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例

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を介し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
- レコードレーベルのカスタマイズ
- CSSとJavaScriptの利用

EBX®のWebコンポーネントとしての使用

EBX®は、Web Component API を使用して、サポートされているWebブラウザ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。

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、EBX®へのURLを示すことにより、HTML タグ `<iframe>` を使用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、EBX®を統合する複数のiframeが含まれる場合があります。その後、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することができます。

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。

強力なカスタムユーザーサービスは、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のフォームのデフォルトのレイアウトと動作を上書きすることができます。

カスタムフォームエディター	DMAの「カスタムフォーム」拡張機能を使用して、ユーザーインターフェイスのフォームのデフォルトレイアウトを上書きすることができます。この拡張機能は、デフォルトのレイアウトを使用して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持ちながら、フォームのレイアウトをカスタマイズする可能性を提供するグラフィカルエディターを提供します。 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の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、フォームのレイアウトをカスタマイズできると同時に、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実現できます。 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サービス

ユーザ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してレコードフォームと同じコンポーネントを使用しながら、フォームのレイアウトをカスタマイズでき、動作をカスタマイズできます。

以下も参照してください。:

- `UserServiceRecordFormFactory`
- [概要](#)

ウィジェットのカスタマイズ

カスタムウィジェットはJava APIに含まれており、フィールドまたはフィールドのグループのユーザインターフェイスコンポーネントを開発できます。カスタムウィジェット（`UICustomWidget`）を使用すると、特定のノードについて、入力または表示コンポーネントが配置されてい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および表示コンポーネントを使用できます。標準コンポーネント（`UIWidgets`）が利用可能であり、使用できます。カスタムウィジェットは、フォームへの入力コンポーネント、フォームへの表示、テーブルセルへの表示など、いくつかの表示アспектを実装できます。カスタムウィジェットが独自のHTMLコンポーネントを書き込む場合、フォームを送信するときにデータベースに値を保存する可能性があります。

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を使用してテーブルフィルタを実装するために使用されます。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します）。

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（外部キーなど）への参照を表示します。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の利用

登録モジュールで宣言することにより、CSSファイルとJavaScriptファイルを各EBX®ページに統合することができます。JavaScriptファイルを含めることは、コンテキストに応じて開発を通じて条件の対象となる可能性があります。

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
- 画面
- 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エLEMENT（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
Dataspace	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。
Dataset	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。
TableView	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。
Record	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

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。
Hierarchy	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。
HierarchyNode	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。
Association	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。
AssociationRecord	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** の性質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。

画面

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 displays the user service interface with the following elements and callouts:

- 1:** User name 'John Doe' in the top header.
- 2:** Message box icon in the top toolbar.
- 3:** Actions dropdown menu in the top toolbar.
- 4:** Navigation buttons (back and forward) in the top toolbar.
- 5:** Help icon in the top toolbar.
- 6:** Close button in the top toolbar.
- 7:** Tabbed interface with 'Main' and 'Groups' tabs.
- 8:** Form fields (Last Name, First Name, Start Date, Country, Job, Email, Phone, Mobile Phone).
- 9:** Bottom buttons (Save, Save and close, Revert, Close).
- A:** Header area.
- B:** Form area.

領域	機能
A. ヘッダー	1. 階層リンク
B. フォーム	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

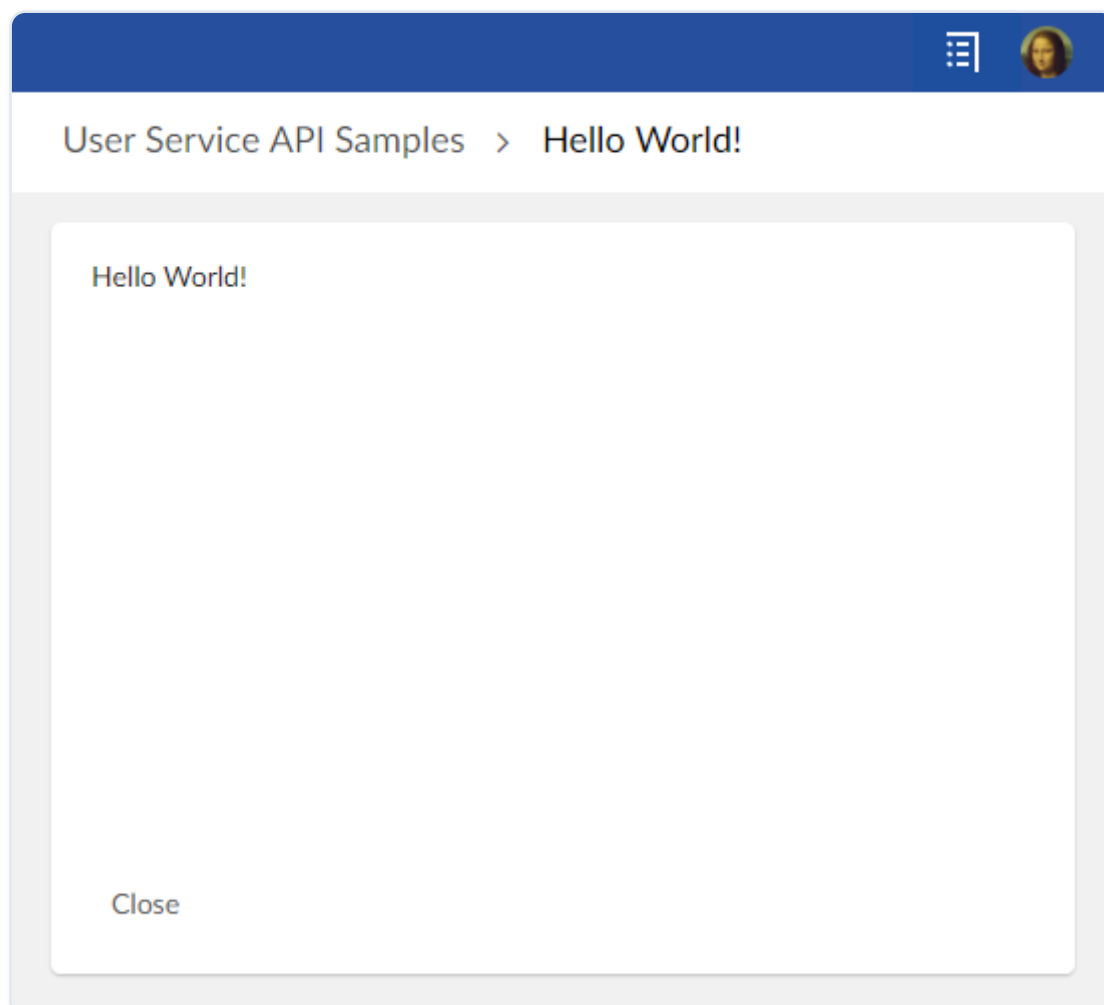
- [メインクラス](#)
- [Hello world](#)

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 instanciated with

        // the associated data model (see next example).

    }

    @Override

    public void defineProperties(UserServicePropertiesDefinitionContext
aContext)

    {

        // This label is displayed in menus that can execute the user
service.

        aContext.setLabel("Hello World Service");

    }

    @Override

    public void declareWebComponent(WebComponentDeclarationContext
aContext)

```



```

    {

    }

}

```

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](#)
- [表示の設定](#)
- [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

次の表に、実装するインターフェイスのネイチャーを示します。

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>
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
- まだ持続していない新しいレコード
- 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。

適応と同様に、パス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);

    }

    ...

}

```

ディスプレイの設定

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です。）
- コンテキストヘルプURL
- 階層リンク
- 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。

`action`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。

送信ボタンは、常にフォームを送信する送信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

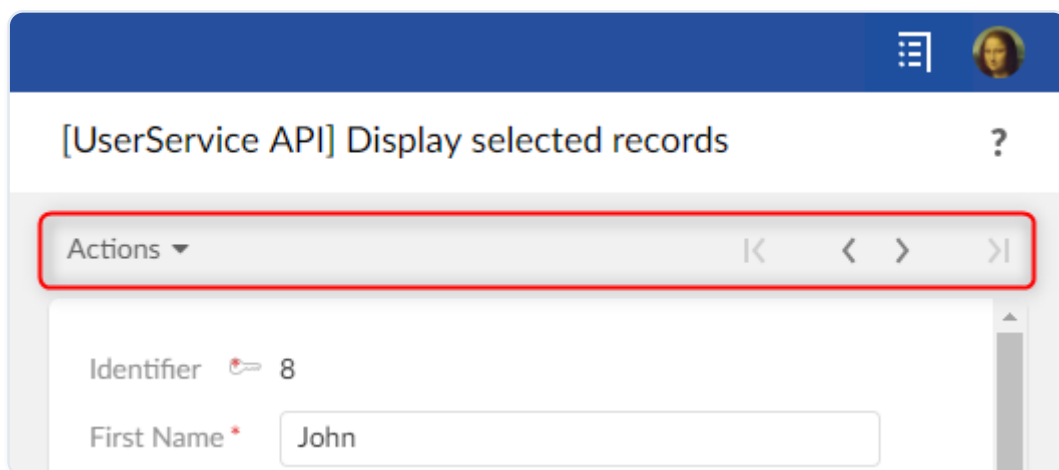
このコールバックは通常、インターフェイス `UserServicePane` を実装して、プレーンなEBX®フォームをレンダリングします。コールバックは、タブ付きのEBX®フォームをレンダリングするための `UserServiceTabbedPane` のインスタンスにすることもできます。

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。

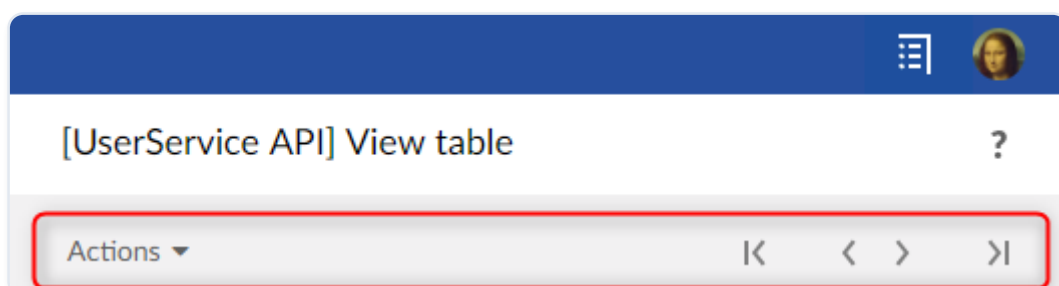
ツールバー

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、Java6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",
UserServiceClasses.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](#)
- [サービスアクティベーションスコープ](#)
- [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](#)
 - [ユーザーサービス拡張機能](#)
- [ユーザーサービスグループ](#)
 - [ビルトインの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](#)
 - [サービスのグループへの関連付け](#)

宣言インターフェイス

次の表は、各ネイチャーに対してユーザーサービスの宣言クラスが実装する必要のあるインターフェイスを示しています。

ネイチャー	宣言インターフェイス
データスペース	<code>UserServiceDeclaration.OnDataspace</code>
Dataset	<code>UserServiceDeclaration.OnDataset</code>
TableView	<code>UserServiceDeclaration.OnTableView</code>
レコード	<code>UserServiceDeclaration.OnRecord</code>
Hierarchy	<code>UserServiceDeclaration.OnHierarchy</code>
HierarchyNode	<code>UserServiceDeclaration.OnHierarchyNode</code>
Association	<code>UserServiceDeclaration.OnAssociation</code>
AssociationRecord	<code>UserServiceDeclaration.OnAssociationRecord</code>

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のクラスである必要があるため、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るモジュール登録サーブレットによって実現されます。

```
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、そのラベル、説明、確認メッセージ、およびサービスを所有するグループが含まれます。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。

サービスアクティベーションスコープ

アクティベーションスコープは、サービスが利用可能な選択を定義します。

サービスアクティベーション定義の例：

```
public class CustomServiceDeclaration implements
UserServiceDeclaration.OnTableView

{
    ...

    @Override

    public void defineActivation(ActivationContextOnTableView aContext)

    {

        // activates the service in all dataspace except the "Reference"
branch.

        aContext.includeAllDataspace(DataspaceType.BRANCH);

        aContext.excludeDataspaceMatching(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());

    }

}

```

アクティベーションスコープ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、[サービスのアクセス許可の解決](#)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、サービスの性質に応じた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利用可能になります。カスタムパラメーターを定義したり、ワークフローのユーザタスク、パースペクティブメニューのアクション、ツールバーのWebコンポーネントアクション

を構成する際にサービスのWebコンポーネントを利用できるように設定するには、`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

これらのサービスをWebコンポーネントとして使用する場合、入力/出力パラメーターを追加するために、既存のユーザーサービス（ビルトインまたはカスタム）を拡張することができます。

そのためには、ユーザーサービス拡張機能を最初にモジュールまたはデータモデルで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るモジュール登録サーブレットによって実現されます。

```
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。サービスのグループへの関連付けを参照してください。

ビルトインの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：

サービスグループキー	Description
@ebx-importExport	EBX®が提供するすべてのビルトインの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。
@ebx-views	[表示]メニューに表示す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ているサービスはデフォルトメニューには表示されませんが、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、モジュールの登録時に、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..."));

}

}

```

サービスのグループへの関連付け

サービスとグループの関連付けは、[宣言](#)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。これは、HTMLの構造が厳密な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ることを意味します。この構造により、ブラウザのレンダリングの違いを少なくして、CSSルールをより細かく制御できます。

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。このようにして、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コンポーネントの準備が整うと、iFrameが読み込まれます。

例

次の例は、任意の `UIComponentWriter` から開発され、`UIHttpManagerComponent` を使用してiFrameのURLを構築し、それを正しい方法で設定するものです。

```
// 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、コンポーネントのスタイルを設定するためにソフトウェアで使用される主要なCSSクラスを提供します。これらのCSSクラスは、管理で背景色、境界線、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統合を保証します。フローティングマージンとパディングは、可変密度に応じて変動します。アイコンのスタイルなどに。

以下も参照してください。:

- `UICSSUtils`

高度なCSS

EBX®を使用すると、すべてのページに1つ以上の外部カスケードスタイルシートを統合できます。リソースと見なされるこれらの外部CSSは、[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、次の推奨事項を順守する必要があります。これらのルールに従わないと、次のような事態が発生する可能性があります。

- 美的および機能的の両方でのソフトウェアの不適切な機能：一部のデータおよび一部の入力コンポーネントの表示が失われるリスクがあります。
- ネイティブCSSルールはCSS実装に影響を与えるため、CSSルールとプロパティの不適切な機能。

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の作成には使用しないでください。

ebx_	内部ビルトイン
yui	Yahooユーザーインターフェイスグローバル
ygtv	Yahooユーザーインターフェイスツリービュー
layout-doc	Yahooユーザーインターフェイスのレイアウト
cke_	CKエディター（HTMLエディターウィジェットで使用）
fa	Font Awesome（パースペクティブとツールバーで使用されるアイコン）

EBX®によって内部的に使用されるCSSクラス

次のCSSクラスは、コンテキストセレクトターのないルールセットに含めないでください。

以下のCSSクラスのいずれかを使用してCSSセレクトターの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が選択したツリーノード
hd	YUIフローティングペインヘッダー
bd	YUIフローティングペイン本体
ft	YUIフローティングペインフッター
container-close	YUI内部ポップアップ閉じるボタン
underlay	YUI内部ポップアップシャドウ
hastitle	タイトル付きのYUIメニューグループ
topscrollbar	YUIメニュートップスクロールゾーン
bottomscrollbar	YUIメニュー下部スクロールゾーン

withtitle	YUIカレンダー
link-close	YUIカレンダー閉じるボタン
collapse	YUIレイアウトクローズドペインインジケータ
pull-right	FontAwesomeパラメーター
pull-left	FontAwesomeパラメーター

競合を回避するための例

次のようには記述しません。

```
.selected {
    background-color: red;
}
```

次のように記述します。

```
#myCustomComponent li.selected {
    background-color: red;
}
```

JavaScript

パブリックJS機能

JavaScript関数のカタログ `JavaScriptCatalog` は、JSファイルで直接（コピーと貼り付けを介して）使用する関数のリストを提供します。

Javaでのページ生成中のJavaScript呼び出し

JavaコンポーネントのHTMLを生成する場合、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、ページ全体が読み込まれると実行されます。 `UIBodyWriter.add` で記述されたHTMLエレメントを即座に管理することができます。オンロード関数（ `window.onload = myFunctionToCallOnload;` など）の設定は、実行コンテキストがオンロードイベントの後に来るため、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。リソースと見なされるこれらの外部JavaScriptファイルは、[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、ユーザーサービスまたは特定のフォームに）。API `UIDependencyRegisterer` を使用すると、開発者は、コンテキストに応じてJavaScriptリソースを特定のページに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。そうでない場合、次のような事態につながる可能性があります。

- 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た場合、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の指示が適切に機能しなくなる可能性があります。

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済みであり、変数、関数、メソッド、クラスなどを作成するために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 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、[Java EE 展開の概要](#)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が排他的である必要があります。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 要求キュー (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。要求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 要求の `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、要求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` 相関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 ヘッダー承認に基づいています。

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==
```

- 「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である必要があります。
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、特定の要求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 要求は、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 要求は、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。 標準

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。 • <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。 • 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。

操作/プロトコル	認証方法と適用条件
	• <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	
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	

操作	SOAP	REST
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		

操作	SOAP	REST
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いて閉じる	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 (*)
その他		
Java API から Web サービスを開発		X (*)

(*) 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生成

- サポートされている標準仕様
- 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ダウンロード
 - HTTPプロトコルからのWSDLダウンロード
- HTTPの例

サポートされている標準仕様

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、さまざまなタイプの操作に対して生成できます。

操作タイプ	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用のWSDL。
directory	デフォルトのEBX®ディレクトリ操作用の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用のWSDL。
tables	特定のデータセットのテーブルに対する操作のWSDL。

操作タイプ	WSDLの説明
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。 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。

操作タイプ	アクセス権のアクセス許可
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。

操作タイプ	アクセス権のアクセス許可
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は使用されません。（ビルトインの管理者ロールは、この条件の対象となるだけです）。 - 管理用にグローバルアクセス許可が定義されています。 「ディレクトリ」データセットのアクセス許可には、現在のプロファイルへの書き込みアクセス権があります。
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。 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ます。 「ユーザーインターフェイス」データセットのアクセス許可には、現在のプロファイルへの書き込みアクセス権があります。

操作タイプ	アクセス権のアクセス許可
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ます。 「管理」データセットのアクセス許可には、現在のプロファイルへの書き込みアクセス権があります。
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形式

```
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。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。
- `{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を介して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ダウンロード操作を指定します。 空の値。
<code>login</code>	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。 <code>文字列</code> 型の値。

パラメーター名	PATHPARAM	QUERYPARAM	必須	説明
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。 可能な値は次のとおりです。 custom 、 dataset 、 directory 、 administration 、 userInterface 、 repository 、 tables または workflow 。 文字列型の値。
branch version	はい	はい	(*)	データスペースまたはスナップショット

パラメーター 名	PATHPARAM	QUERYPARAM	必須	説明
				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。
<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

パラメーター 名	PATHPARAM	QUERYPARAM	必須	説明
				ルが選択されます。 各テーブルパスはカンマ文字で区切られます。 文字列 型の値。
operations	いいえ	はい	いいえ	操作サブセットのWSDLを生成できます。 tables または directory の操作タイプ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

パラメーター 名	PATHPARAM	QUERYPARAM	必須	説明
				文字を連結したものです。 • C =レコードをカウント • D =レコードを削除 • E =認証情報を取得 • G =変更を取得 • I =レコードを挿入 • U =レコードを更新 • R =読み取り操作 (CEGSと同等) • S =レコードを選択 • 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サービスの一意のネームス

パラメーター 名	PATHPARAM	QUERYPARAM	必須	説明
				ベースURIを指定します。 (**) <code>type</code> パラメーターが <code>custom</code> 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。 <code>URI</code> 型の値。
<code>attachmentFilename</code> いいえ		はい	(***)	添付ファイル名を指定します。 (***) <code>isContentInAttachment</code> パラメーターが定義され、<code>true</code> 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。
<code>isContentInAttachment</code> 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

パラメーター名	PATHPARAM	QUERYPARAM	必須	説明
				かを指定します。 <code>ブール値</code> 型の値。 デフォルト値は <code>false</code> です。
<code>targetNamespace</code> いいえ		はい	いいえ	WSDLのターゲットネームスペースURIをオーバーライドします。 <code>URI</code> 型の値。 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）。

ステータスコード	情報
400 (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は、少なくとも1つのエラーが含まれています。 注意: 詳細については、WSDLと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 (Method not allowed)	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）。

レスポンス本文

応答本文は、返される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ヘッダーの `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にある場合、HTTPヘッダー `Content-Disposition` は `添付ファイルfilename*=UTF-8'<filename.wsd1>` に対して設定されます。

- 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'` データスペース内の `'dataset1'` データセットの読み取り操作のみを含むすべてのテーブルが含まれます。

`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'` データスペース内の `'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
 - 共通のリクエストパラメーター
 - Select 操作
 - Delete 操作
 - Count 操作
 - Update 操作
 - Insert 操作
 - getChanges 操作
 - getCredentials 操作
 - 複数連鎖操作
 - 楽観ロック
- データセットとデータスペースの操作
 - データスペースの検証
 - データセットの検証
 - データスペースの作成
 - スナップショットの作成

- データスペースのロック
- データスペースのロック解除
- データスペースのマージ
- 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
 - ワークフローの開始
 - ワークフローの再開
 - ワークフローの終了
- 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イスの操作
 - システム情報の操作

データモデルから生成された操作

TIBCO EBX® のリポジトリで使用するデータモデルでは、その操作を定義する対応する WSDL を動的に生成することができます。この WSDL を使用すると、EBX® リポジトリの読み取りと書き込みが可能になります。たとえば、パス `/root/XX/exampleTable` 上のテーブルの場合、生成されるリクエストはその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">` を含むことになります。

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 エLEMENT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は、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` パラメーターの値を変更すると、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。

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
- 権限

関連付けフィールド

テーブルレコードの読み取りアクセスは、UI マネージャーに表示されるように関連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。

注意:

制限事項：変更および Update 操作では、関連付けフィールドは管理されません。また、Select 操作では、最初のレベルの関連付けエレメントのみがエクスポートされます (関連付けられたオブジェクトのコンテンツに関連付けエレメントを含めることはできません)。

共通のリクエストパラメーター

パラメーターの一部は複数の操作に共通です。詳細は以下のとおりです。

エレメント	説明	必須
branch	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「version」パラメーターのいずれかを定義する必要があります。「Insert」、「Update」、および「Delete」操作に必要です。
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「branch」パラメーターのいずれかを定義する必要があります。
instance	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
predicate	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「Delete」操作を除いて、すべてのレコードが取得されます。 以下も参照してください。: includesMetadata パラメーター	「Delete」操作にのみ必要です。
data	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	Insert および Update 操作にのみ必要です。

エレメント	説明	必須
	細については、 インポートセクション で指定されます。	
viewPublication	このパラメーターは、論理演算子 AND として <code>predicate</code> パラメーターと組み合わせることができます。 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。	いいえ
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引き続き使用できますが、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。	いいえ

エレメント	説明	必須
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れたブロッキング制約を無効にします。 このパラメーターが存在しない場合、デフォルトは <code>false</code> です。 詳細については、ブロッキングと非ブロッキングの制約を参照してください。	いいえ
details	<code>details</code> エレメントは、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 blockingConstraintsDisabled パラメーターの言語を定義します。	いいえ
disableRedirectionToLastBroadcast	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	いいえ

エレメント	説明	必須
	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。 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 マスターでのリダイレクトが有効) です。 指定されたデータスペースが D3 プライマリノード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。	

Select 操作

テーブルでの Select リクエスト

```
<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}>

```

追加事項：

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。 このパラメーターは、論理的な AND 操作として <code>viewPublication</code> パラメーターと組み合わせることができます。	

エレメント	説明	必須
viewPublication	共通パラメーター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、<code>ebx-metadata</code> ルートエレメントの下にステップと呼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 (<code>system, teamUp</code>)。 <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。	いいえ

エレメント	説明	必須
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。以下はその例です。 <pre>...<table ebxd:lastTime="2010-06-28T10:10:31.046" ebxd:lastUser="Uadmin ebxd:uuid="9E7D0530-828C-11DF-B733-0012D01B6E76">... .</pre>	いいえ
exportCredentials	<code>true</code> の場合、SELECT で各レコードの資格情報も返されます。	いいえ
pagination	ページネーション構成を指定します。以下の子エレメントを参照してください。	いいえ
pageSize (<code>pagination</code> エレメント下にネスト)	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> 型。デフォルト値は 10 です。 以下も参照してください。: ebx.dataservices.pagination.pageSize.default	いいえ

エレメント	説明	必須
	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。 以下も参照してください。: ebx.dataservices.pagination.maxSize.default 注意: 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 (<code>pagination</code> エレメントの下にネスト)	ページネーション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、エレメント <code>lastRecordPredicate</code> として前の応答によって提供されます。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。	いいえ
disableRedirectionToLastBroadcast	共通パラメーターの説明を参照してください。	

テーブルでの Select 応答

```
<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>
```

追加事項：

エレメント	説明
data	テーブルパスの後に表示されるレコードのコンテンツ。
credentials	各レコードの各ノードのアクセス権限が含まれています。
lastRecordPredicate	ページ機能が有効な場合にのみ返されます。これは、エレメント <code>previousPageLastRecordPredicated</code> で、次のリクエストで使用されるよう最後のレコードを定義します。

楽観ロックのセクションも参照してください。

データセットでの Select リクエスト

この操作は、テーブルなしで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>
```

追加事項：

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。	
disableRedirectionToLastBroadcast	共通パラメーターの説明を参照してください。	

データセットでの Select 応答

```
<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>
```


追加事項：

エレメント	説明
data	テーブルなしのデータセットコンテンツ。

Delete 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「非表示」または「継承」として定義します。レコードは、`predicate` パラメーターで選択されます。

Delete リクエスト

```
<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}>
```

追加事項：

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。	
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。	
includeOcculting	非表示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。	いいえ
inheritIfInOccultingMode	非表示モードの場合、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。	いいえ
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、レコードを非表示にします。デフォルト値は <code>false</code> です。	いいえ
checkNotChangedSinceLastTime	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るタイムスタンプ。楽観ロックセクションも参照してください。	いいえ
blockingConstraintsDisabled	共通パラメーターの説明を参照してください。	
details	共通パラメーターの説明を参照してください。	
disableRedirectionToLastBroadcast	共通パラメーターの説明を参照してください。	

Delete 応答

指定されたパラメーターの 1 つが不正である場合、必要な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>
```

追加事項：

エレメント	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、操作全体が中止されたことを示します。
blockingConstraintMessage	このエレメントは、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、リクエスト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。

Count 操作

Count 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}>

```

追加事項：

エレメント	説明
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。
disableRedirectionToLastBroadcast	共通パラメーターの説明を参照してください。

Count 応答

```

<ns1:count_{TableName}Response xmlns:ns1="urn:ebx-
schemas:dataservices_1.0">

    <count>Integer</count>

</ns1:count_{TableName}Response>

```

追加事項：

エレメント	説明
count	リクエストの述語に対応するレコードの数。

Update 操作

Update 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}>

```

追加事項：

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。	
instance	共通パラメーターの説明を参照してください。	
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。 ブール値 型。デフォルト値は <code>false</code> です。	いいえ
byDelta	<code>true</code> に設定され、受信メッセージにエレメント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> に設定され、ノードが <code>hiddenInDataServices</code> に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、Insert および Update 操作セクションで説明しています。	いいえ
blockingConstraintsDisabled	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
details	共通パラメーターの説明を参照してください。	
disableRedirectionToLastBroadcast	共通パラメーターの説明を参照してください。	
data	共通パラメーターの説明を参照してください。	

以下も参照してください。:

- 楽観ロック

Update 応答

```
<ns1:update_{TableName}Response xmlns:ns1="urn:ebx-
schemas:dataservices_1.0">

  <status>String</status>

  <blockingConstraintMessage>String</blockingConstraintMessage>

</ns1:update_{TableName}Response>
```

追加事項:

エレメント	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、操作全体が中止されたことを示します。
blockingConstraintMessage	このエレメントは、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

エレメント	説明
	存在します。メッセージのロケールは、リクエスト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。

Inset 操作

Insert リクエスト

```
<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}>

```

追加事項：

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。	
instance	共通パラメーターの説明を参照してください。	
byDelta	<code>true</code> に設定され、受信メッセージにエレメント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> に設定され、ノードが <code>hiddenInDataServices</code> に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、Insert および Update 操作セクションで説明しています。	いいえ
blockingConstraintsDisabled	共通パラメーターの説明を参照してください。	
details	共通パラメーターの説明を参照してください。	
disableRedirectionToLastBroadcast	共通パラメーターの説明を参照してください。	
data	共通パラメーターの説明を参照してください。	

Insert 応答

```
<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>
```

追加事項：

エレメント	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、操作全体が中止されたことを示します。
blockingConstraintMessage	このエレメントは、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、リクエスト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。
predicate	挿入されたレコードの主キーに一致する述語。複数のレコードが挿入される場合、述語は入力メッセージ内のレコードの宣言順序に従います。

getChanges 操作

コンテンツポリシーに従って変更を返します。

getChanges リクエスト

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}>

```

追加事項：

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。	
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 パラメーターが使用されます。	いいえ

エレメント	説明	必須
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 <code>true</code> です。詳細については、 解決モード を参照してください。	いいえ
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 <code>false</code> です。	いいえ
pagination	<code>getChanges</code> 操作および <code>getChangesOnDataSet</code> 操作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。変更の取得の永続化コンテキストは、リクエストのページサイズパラメーターに従って、初回呼び出し時に構築されます。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 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>コンテキスト</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。	いいえ

エレメント	説明	必須
pageSize (<code>pagination</code> エレメント下にネスト)	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)
context (<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。	いいえ (next のみ)
identifier (<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。	はい
disableRedirectionToLastBroadcast	共通パラメーター の説明を参照してください。	

注意:

`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、比較はそれらの親で実行されます。

- 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、比較は最初のスナップショットで実行されます (データスペースで行われたすべての変更を含む)。
- 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、比較はその親データスペースで実行されます (現在のスナップショットの作成後に親データスペースで行われたすべての変更を含む)。
- 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。

以下も参照してください。:

- `DifferenceHelper`

getChanges 応答

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>

```

追加事項：

エレメント	説明	必須
inserted	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エレメントの下コンテンツ。	いいえ
updated	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。	いいえ

エレメント	説明	必須
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ています。	はい
change (<code>変更</code> エレメントの下にネスト)	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> エレメント下にネスト)	レコード内のパス。	はい
path (<code>change</code> エレメント下にネスト)	データセット内のパス。	はい
data (<code>updated</code> エレメント下にネスト)	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エレメント下のコンテンツ。	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	いいえ

エレメント	説明	必須
	有効にします。以下の子エレメントを参照してください。	
context (<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)
identifier (<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
pageNumber (<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
totalPages (<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}>
```

追加事項：

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_>

```

追加事項：

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。	

エレメント	説明	必須
details	共通パラメーターの説明を参照してください。	
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>
```

追加事項：

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

エレメント	説明
	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LEMENT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` ELEMENT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く	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

エレメント	説明	必須
	て、「参照」データスペースが使用されます。	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
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。
sureActivation	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」)	いいえ。指定しない場合、詳細は返されません。

エレメント	説明	必須
	は、検証メッセージが返される言語を定義します。	
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
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されています)。	言語のラベル。	いいえ
description (<code>documentation</code> エレメント下にネスト)	言語の説明。	いいえ

データスペースの検証

データスペース検証リクエスト

```
<m:validate xmlns:m="urn:ebx-schemas:dataservices_1.0">

  <branch>String</branch>

  <version>String</version>
```



```
</m:validate>
```

データスペース検証の応答

```
<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

データセットの検証リクエスト

```
<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>

```

データセット検証の応答

```

<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

データスペース作成リクエスト

```

<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>
```

データスペース作成の応答

```
<ns1:createBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

  <status>String</status>

  <childBranchName>String</childBranchName>

</ns1:createBranch_Response>
```

追加事項：

エレメント	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

スナップショット作成リクエスト

```
<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>

```

スナップショット作成の応答

```

<ns1:createVersion_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

    <status>String</status>

    <versionName>String</versionName>

</ns1:createVersion_Response>

```

追加事項：

エレメント	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

データスペースロックリクエスト

```

<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>

```

追加事項：

エレメント	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> エレメントを使用できます。	いいえ
locale (<code>message</code> エレメント下にネスト)	ユーザーメッセージのロケール。	<code>message</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>message</code> エレメント下にネスト)	ユーザーメッセージ。	いいえ

データスペースロックの応答

```

<ns1:lockBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

    <status>String</status>

</ns1:lockBranch_Response>

```

追加事項：

エレメント	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。

データスペースロックの解除

データスペースロック解除リクエスト

```
<m:unlockBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">

    <branch>String</branch>

</m:unlockBranch>
```

データスペースロック解除の応答

```
<ns1:unlockBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

    <status>String</status>

</ns1:unlockBranch_Response>
```

追加事項：

エレメント	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

データスペースマージリクエスト

```
<m:mergeBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">

  <branch>String</branch>

  <deleteDataOnMerge>boolean</deleteDataOnMerge>

  <deleteHistoryOnMerge>boolean</deleteHistoryOnMerge>

</m:mergeBranch>
```

追加事項：

エレメント	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で使用できます。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® メイン構成ファイルでプロパティ <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> を指定してデフォルト値を再定義することができます。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。	いいえ
deleteHistoryOnMerge	このパラメーターは、データスペースのマージ操作で使用できます。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	いいえ

エレメント	説明	必須
	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> です。 EBX® メイン構成ファイルでプロパティ <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> を指定することにより、デフォルト値を再定義することができます。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。	

注意:

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。

データスペースマージの応答

```
<ns1:mergeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

  <status>String</status>

</ns1:mergeBranch_Response>
```

追加事項:

エレメント	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を閉じるリクエスト

データスペースを閉じるリクエスト

```
<m:closeBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">

  <branch>String</branch>

  <deleteDataOnClose>boolean</deleteDataOnClose>

  <deleteHistoryOnClose>boolean</deleteHistoryOnClose>

</m:closeBranch>
```

スナップショットを閉じるリクエスト

```
<m:closeVersion xmlns:m="urn:ebx-schemas:dataservices_1.0">

  <version>String</version>

  <deleteDataOnClose>boolean</deleteDataOnClose>

</m:closeVersion>
```

追加事項：

エレメント	説明	必須
deleteDataOnClose	このパラメーターは、データスペースおよびスナップショットを閉じる操作で使用できます。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	いいえ

エレメント	説明	必須
	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> を指定することにより、このデフォルト値を再定義することができます。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。	
<code>deleteHistoryOnClose</code>	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> です。EBX® メイン構成ファイルでプロパティ <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> を指定することにより、デフォルト値を再定義することができます。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。	いいえ

データスペースまたはスナップショットを閉じる応答

データスペースを閉じる応答

```
<ns1:closeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
```

```
<status>String</status>
```

```
</ns1:closeBranch_Response>
```

スナップショットを閉じるリクエスト

```
<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}>
```

追加事項：

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。	はい
instance	共通パラメーターの説明を参照してください。	はい

エレメント	説明	必須
unitName	レプリケーションユニットの名前。 以下も参照してください。: レプリケーションの更新情報	はい

レプリケーション更新の応答

```
<ns1:replicationRefresh_${schema}Response xmlns:ns1="urn:ebx-
schemas:dataservices_1.0">

  <status>String</status>

</ns1:replicationRefresh_${schema}Response>
```

追加事項:

エレメント	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 以降非推奨。*SOAP メッセージ本文でパラメーターを定義します。

詳細については、[セッションパラメーター](#)を参照してください。

エレメント	説明	必須
parameters	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。 注意: 少なくとも 1 つのセッションパラメーターが定義されている場合、parameter エレメントは無視されます。	いいえ
parameter (parameters エレメント下にネスト)。複数の parameter エレメントを使用できます。	ワークフローの入力パラメーター。	いいえ
name (parameter エレメント下にネスト)	パラメーターの名前。	はい
value (parameter エレメント下にネスト)	パラメーターの値。	いいえ

ワークフローの開始

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。

注意:

ワークフローの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>
```

追加事項：

エレメント	説明	必須
publishedProcessKey	ワークフローランチャーの識別子。	はい
documentation	共通パラメーターの説明 を参照してください。	いいえ
parameters	バージョン 5.7.0 以降非推奨。 共通パラメーターの説明 を参照してください。	いいえ

サンプル応答

```
<m:workflowProcessInstanceStart_Response xmlns:m="urn:ebx-
schemas:dataservices_1.0">
```

```
<processInstanceKey>String</processInstanceKey>

</m:workflowProcessInstanceStart_Response>
```

追加事項：

エレメント	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、今後のメジャーバージョンでは最終的に削除される予定です。	いいえ
processInstanceKey	ワークフロー識別子。	いいえ

ワークフローの再開

再開識別子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使用) 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>
```

追加事項：

エレメント	説明	必須
resumeId	待機中のタスクの ID を再開します。	はい

エレメント	説明	必須
parameters	バージョン 5.7.0 以降非推奨。 共通パラメーターの説明 を参照してください。	いいえ

サンプル応答

```
<m:workflowProcessInstanceResume_Response xmlns:m="urn:ebx-
schemas:dataservices_1.0">

  <status>String</status>

  <processInstanceKey>String</processInstanceKey>

</m:workflowProcessInstanceResume_Response>
```

追加事項：

エレメント	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。	いいえ

ワークフローの終了

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>

```

追加事項：

エレメント	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。
publishedProcessKey	*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、今後のメジャーバージョンでは最終的に削除される予定です。	いいえ
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、今後のメジャーバージョンでは最終的に削除される予定です。	いいえ

サンプル応答

```
<m:workflowProcessInstanceEnd_Response xmlns:m="urn:ebx-
schemas:dataservices_1.0">

    <status>String</status>

</m:workflowProcessInstanceEnd_Response>
```

追加事項：

エレメント	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

管理サービス

ディレクトリサービス

ディレクトリ上のサービスは、デフォルトディレクトリの「Users」テーブルと「Roles」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ビルトインのロール「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。

ディレクトリへのユーザーの作成

この SOAP 挿入リクエストの例では、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>
```

SOAP 応答の挿入構文については、[Insert 応答](#)を参照してください。

ユーザーインターフェイスの操作

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イスで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
closedMessage	ユーザーインターフェイスを閉じてアクセスしたときにユーザーに表示されるメッセージ。	いいえ

ユーザーインターフェイスを閉じるリクエスト

閉じる操作は、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を削除します。

```
<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イスを閉じるの応答

```
<ns1:close_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">

  <status>String</status>

</ns1:close_Response>
```

ユーザーインターフェイスを開くリクエスト

```
<m:open xmlns:m="urn:ebx-schemas:dataservices_1.0"/>
```

ユーザーインターフェイスを開くの応答

```
<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 タブの > に含まれている情報と同じです。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、および値が含まれています。この操作を実行するには、認証されたユーザーがビルトインのロール「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。

パラメーター	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® default <code>locale</code>) は、システムアイテムメッセージを返す必要があ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。

システム情報リクエスト

この SOAP リクエストは、すべての EBX® インスタンスのシステム情報を返し、「`en_US`」`locale` を使用してフォーマットを設定します。

```
<m:systemInformation xmlns:m="urn:ebx-schemas:dataservices_1.0">

    <details locale="en_US" />

</m:systemInformation>
```

```
</m:systemInformation>
```

システム情報の応答

```
<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](#)

- 拡張形式
- コンパク形式
- 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要求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® モジュールと一緒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、[Java EE 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 形式である必要があります。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、次の組み合わせで説明されています。

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 true</code>	true	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。 ソート不可
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
<code><hidden></code>	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			

XML エlement	値	スキーマノード タイプ	ビュー	動作
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				
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。 ソート不可		
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 ソートで非表示
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				

XML エlement	値	スキーマノード タイプ	ビュー	動作
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				
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設定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` タイプです。

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 ヘッダー承認に基づいています。

```
> 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 ヘッダー承認に基づいています。

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 QWxpYmFiYTpvYGVuIHNlc2FtZQ==
```

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](#)

- 「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、特定の要求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。

操作/プロトコル	認証方法と適用条件
	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。 • 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。 • 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	
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

操作	SOAP	REST
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いて閉じる	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 (*)
その他		
Java API から Web サービスを開発		X (*)

(*) 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。

ビルトイン 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
 - メッセージ本文
- レスポンス
 - ヘッダーフィールド
 - HTTP コード
 - メッセージ本文
- 管理操作
 - ディレクトリ操作

- ユーザーインターフェイスの操作
- システム情報の操作
- トークン認証操作
 - トークン操作の作成
 - パスワード変更操作
 - トークンの無効化操作
- データ操作
 - Select 操作
 - 準備操作
 - Insert 操作
 - Update 操作
 - Delete 操作
 - Count 操作
 - 楽観ロック
 - 継承
 - ルックアップテーブルビュー操作
- 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
- ベータ機能：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を閉じる
- ヘルスオペレーション

- 概要
- 運用開始
- 動作確認
- 運用停止
- OpenAPI 操作
 - 概要
 - OpenAPI ドキュメントの生成
- ステージングオペレーション
 - ドメイン管理操作
 - ステージングエレメントの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
- 制限事項
 - 一般的な制限事項
 - コンパクトフォーマットの制限事項
 - 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) に代わるものと考えられます。選択したリソースは、リクエストのコンテンツに応じて、サードパーティのシステムで読み取りおよび書き込みが可能です。

ビルトイン RESTful サービスの HATEOAS アプローチにより、直感的で分かりやすいナビゲーションも可能になります。これは、リンクを介してデータの詳細を取得できることを意味します。

注意:

すべての操作はステートレスです。

リクエスト

ここでは、HTTP メソッド、URL フォーマット、ヘッダーフィールド、メッセージ本文など、適合 REST リクエストを作成するために使用するエレメントについて説明します。

以下も参照してください。:

- インタラクション
- セキュリティ

HTTP メソッド

ビルトイン 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の選択に使用されます (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よって異なります。2KB 以下である必要があります)。
- **POST** : テーブルへのレコードの挿入、または URL で定義されたマスターデータの選択に使用されます (サイズ制限は 2MB 以上ですが、アプリケーションサーバーによって異なります。各パラメーターは、1024 バイトまでの値に制限されます)。
- **PUT** : URL で定義されたマスターデータの更新に使用されます。
- **DELETE** : URL で定義されたレコード、またはテーブル URL とメッセージ本文のレコードキーで定義された複数レコードのいずれかを削除するために使用されます。

URL

REST URL には次のものが含まれます。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/  
{categoryVersion}/{specificPath}[:{extendedAction}]?{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。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。
- `{category}` は、[操作カテゴリ](#)に対応します。
- `{categoryVersion}` は、カテゴリの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)。
- `{queryParams}` 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

操作を特定します。URL のパスの `{category}` に追加され、次のいずれかの値を指定します。

<code>admin</code>	管理者専用の管理操作です。 詳細については、管理操作を参照してください。
<code>auth</code>	トークン認証方法を管理します。 詳細については、トークン認証操作およびトークン認証スキームを参照してください。
<code>data</code> または <code>data-compact</code>	データセットのコンテンツを表示します。テーブル、レコード、またはフィールドレコードのコンテンツ (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での変更操作を含む) をリクエストします (単純な使用では、インタラクションを簡易にするためコンパクトフォーマットが使用可能)。 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。

	詳細については、データ操作、ベータ機能：データスペース操作、およびコンパクトフォーマットの制限を参照してください。
<code>form-data</code> または <code>form-data-compact</code>	受信データを検証し、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レポートを返します (単純なユースケースでの対話を容易にするために、コンパクトフォーマットが利用可能です)。 詳細については、フォームデータ操作およびコンパクトフォーマットの制限を参照してください。
<code>health</code>	サーバーのヘルス情報は、監視ツールまたはコンテナプローブの情報を返します。 詳細については、ヘルスオペレーションを参照してください。
<code>history</code>	履歴データセットのコンテンツを表示します。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。 詳細については、データ操作を参照してください。 以下も参照してください。: • 履歴
<code>api</code>	選択したリソースの OpenAPI ドキュメントを生成します。 詳細については、OpenAPI 操作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、TIBCO EBX® によって使用されます。

Accept	応答で使用されるコンテンツタイプを (優先順に) 指定するために使用されます。最初にサポートされたものが選択され、応答ヘッダー Content-Type で指定されます。現在、サポートされているのは application/json のみです。サポートされていない場合、結果はプロパティ ebx.dataservices.rest.request.checkAccept によって異なります。 • true の場合、HTTP エラー応答がコード 406 で返されます。 • false の場合、応答はデフォルトのコンテンツタイプ application/json で返されます。 以下も参照してください。: • データサービスの構成
Accept-Language	応答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。 優先ロケールがサポートされていない場合は、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ている場合、要求は拒否されます。 以下も参照してください。: • 認証

Content-Type	リクエスト本文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、コード <code>415</code> (<i>Unsupported media type</i>) を含む HTTP エラーメッセージが返されます。 以下も参照してください。: データサービスの構成
X-Requested-With	存在する場合、認証が失敗した場合に応答に <code>WWW-Authenticate</code> ヘッダーが追加されないようにします。 以下も参照してください。: 応答ヘッダー <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のパラメーターは、すべてのデータサービス操作で使用できます。

パラメーター	説明
<code>disableRedirectionToLastBroadcast</code>	このパラメーターは、D3 アーキテクチャにのみ影響します。 <code>true</code> の場合、D3 プライマリノード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

パラメーター	説明
	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。 ブール値 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</code> が設定されていない限り、デフォルトは false (D3 マスターへのリダイレクトが有効) です。 以下も参照してください。: プライマリノード
<code>ebx-indent</code> <code>indent</code> (6.0.0 以降非推奨)	読みやすくするために、応答をインデントするかどうかを指定します。 ブール値 型。デフォルト値は false です。
<code>ebx-channel</code>	セッションチャンネルを指定します。 文字列 型。使用可能な値は次のとおりです。 <code>dataServices</code> <code>ui</code> デフォルト値は <code>dataServices</code> です。 以下も参照してください。: <code>Constants.Data.PARAM_EBX_CHANNEL</code>

メッセージ本文

JSON 形式を使用したリクエストデータが含まれています。拡張 JSON リクエスト本文およびコンパクト JSON リクエスト本文を参照してください。

注意:

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本文を定義できます。

応答

ここでは、ビルトイン RESTful サービスで返される応答について説明します。

- 標準エラー処理の詳細については、例外処理を参照してください (HTTP コードが `300` 以上の場合)。

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>	ラベルと説明の応答で使用するロケールを示します。
<code>Content-Type</code>	応答本文のコンテンツタイプを示します。
<code>Location</code>	新しいレコードが正常に挿入された場合、このレコードのクエリ URL がこのフィールドで返されます。
<code>WWW-Authenticate</code>	このヘッダーフィールドは、401 (<i>Unauthorized</i>) HTTP コードで認証が失敗した場合に HTTP 応答に追加されます。その値は、リクエスト URI に適用可能な少なくとも 1 つの認証方法を含むリストで構成されます。次の条件が確認された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

- `X-Requested-With` HTTP ヘッダーが存在しない

クライアントが認証方法を解釈できる場合は、適切な資格情報を提供してリクエストを再送信できます。

管理プロパティ

`ebx.dataservices.rest.auth.tryBasicAuthentication` を `true` に設定する必要があります。

以下も参照してください。:

- リクエストヘッダー `X-Requested-With`
- 認証

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>201</code> (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 <code>Location</code> がそのリソース URL とともに返されます。
<code>204</code> (No content)	リクエストは正常に処理されましたが、応答本文は返されません。
<code>400</code> (Bad request)	リクエストの URL または本文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)	認証済みユーザーの指定されたリソースの読み取りまたは変更権限が拒否されました。

HTTP コード	説明
	このエラーは、ユーザーが次の場合にも返されます。 - リクエストメッセージ本文に記載されているフィールドを変更できない。 - ユーザーに、REST コネクタへのアクセスが許可されていない場合。 詳細については、グローバル許可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 <code>Accept</code> 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。 以下も参照してください。: 楽観ロック
415 (Unsupported media type)	リクエストコンテンツはサポートされません。リクエストヘッダー値 <code>Content-Type</code> はこの操作でサポートされません。
422 (Unprocessable entity)	セマンティック上の理由により、新しいリソースのコンテンツを受け入れできません。
500 (Internal error)	アプリケーションによる予期しないエラー。エラーの詳細は通常、EBX® ログに記録されます。

メッセージ本文

応答本文のコンテンツのフォーマットは、HTTP コード値によって異なります。

- **200** included から **300** excluded までの HTTP コード：コンテンツのフォーマットは、関連するリクエストによって異なります ([拡張 JSON](#) および [コンパクト JSON](#) のサンプル)。

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説明しています。フォーマットの詳細については、[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。
- **データ** カテゴリからアクセス可能な管理データスペース。

注意:

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® のデフォルトのディレクトリ構成は、ビルトイン RESTful サービスで管理できます。ユーザーとロールテーブル、メーリングリストおよびその他のオブジェクトが関係します。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。

注意:

トリガーはディレクトリのテーブルに存在し、データの一貫性を保証します。

URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/Bebx-directory/ebx-directory
```

ディレクトリ構成操作

EBX® のデフォルトのディレクトリ構成は、データセットノードのように管理できます。[データ](#) カテゴリ操作を通じてアクセスおよび変更できます。各フィールドは、メタモデルが要求されたときに自己記述されます。

詳細については、[Select 操作](#)および[Update 操作](#)を参照してください。

メーリングリストの操作

EBX® ディレクトリに構成可能なデフォルトのメーリングリストは2つあります。1つは全員用、もう1つは管理者用です。これらのリストは、[データ](#) カテゴリ操作でデータセットノードと同様に扱うことができます。

詳細については、[Select 操作](#)および[Update 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、データ操作を使用して、[データ](#) カテゴリのレコードと同様に操作できます。セキュリティ上の理由から、管理者は管理者自身を削除することはできません。ユーザーの敬称は、「敬称」テーブルで使用可能なものから選択する必要があります。

詳細については、[Select 操作](#)、[Update 操作](#)、[Insert 操作](#)、[Delete 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 [データ](#) 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、ディレクトリがユーザーインターフェイスを介して管理されるときに自動的に供給されます。ただし、データサービスではなく、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」テーブルに関しては、ロールの包含の管理には手動操作が必要です。メタモデルが要求されると、各テーブルは自己記述的です。

詳細については、[Select 操作](#)、[Update 操作](#)、[Insert 操作](#)、[Delete 操作](#)を参照してください。

ユーザーインターフェイスの操作

EBX® ユーザーインターフェイスは、保守の必要性に応じてユーザーに対して開くことも閉じることもできます。処理される情報は、タブの>>>>に含まれているものと同様です。

URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/Bebx-manager/ebx-  
manager/domain/toolStatus
```

以下も参照してください。:

- アプリケーションのロック

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には、データセットノードと同様にアクセスできます。

詳細については、[Select 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イスを開くまたは閉じる

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は、Update 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イスを開く場合は、`toolStatus` のコンテンツを `true` に設定し、閉じる場合は `false` に設定します。

詳細については、[Update 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® サーバー上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：リクエスト本文が無視されるため、`POST` HTTP メソッドでは更新できません。返される情報は、ログヘッダー `kernel.log` または タブの > に含まれている情報と同じです。レスポンスには、EBX® の構成とステータスを表すいくつかのキー、ラベル、および値が含まれます。レスポンスの表示モードは、フラットにすることも階層にすることもできます。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/admin/v1/  
systemInformation
```

以下も参照してください。:

- 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> : 次の情報グループ下のフラット表示: <code>bootInfoEBX</code>、<code>repositoryInfo</code>、および <code>bootInfoVM</code>。 • <code>hierarchical</code> : 階層表示。 文字列 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。次のエラーのいずれかが含まれています。 • HTTP メソッドが <code>GET</code> および <code>POST</code> のいずれでもありません。 • HTTP パラメーター <code>systemInformationMode</code> が正しくありません。 • この操作はサポートされません。 • リクエストパスが無効です。

HTTP コード	説明
403 (Forbidden)	ユーザーは管理者ではありません。

応答本文

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ー `systemInformationMode` またはそのデフォルト値によって異なります。

フラット表示の JSON の例を参照してください。

階層表示の JSON の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。この期間内にトークンを使用して EBX® サーバーにアクセスしなかった場合、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。

注意:

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` で変更できます (デフォルト値は 30 分)。

トークンの作成操作

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ー` 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 フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/token:create
```

メッセージ本文

メッセージ本文は HTTP リクエストで定義する必要があります。必然的に、次のデータセットのいずれかが含まれます。

- `login` と `password` の値。両方の JSON 属性は必須で、`文字列` 型です。

詳細については、`Directory.authenticateUserFromLoginPassword` を参照してください。

- `specific` JSON 属性を `true` に設定します。このフラグを有効にすると、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。`login` および `password` 属性が JSON リクエストの本文で定義されている場合でも、`Specific` を `true` に設定すると、特定のユーザーの認証を有効にできます。

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` の例を参照してください。

HTTP コード

HTTP コード	説明
<code>200</code> (OK)	トークンは正常に作成されました。
<code>400</code> (Bad request)	原因は次のいずれかです。 • 構文が正しくありません。 • HTTP メソッドが <code>POST</code> ではありません。 • この操作はサポートされません。
<code>401</code> (Unauthorized)	原因は次のいずれかです。 • <code>login</code> および/または <code>password</code> が正しくありません。 • 他のメソッドの認証データが定義されています。
<code>422</code> (Unprocessable entity)	原因は次のいずれかです。 • <code>PasswordMustChange</code> : パスワードを変更する必要があります (デフォルトの

HTTP コード	説明
	ディレクトリでのみ使用可能)。パスワードの変更操作を参照してください。 <code>RestrictedAccess</code> : ユーザーアクセスは、保守またはその他の操作のために閉じられます (管理者に予約されています)。

応答本文

HTTP コードが `200 (OK)` の場合、本文はトークン値とそのタイプを保持します。

トークン作成レスポンスの [JSON](#) の例を参照してください。

トークンは、後で HTTP-Header `Authorization` を適宜設定することにより、ユーザーを認証するために使用できます。

以下も参照してください。:

- 「トークン認証スキーム」メソッド

パスワード変更操作

この操作により、既存のユーザーアカウントのパスワードが変更されます。これは、認証済みコンテキストで使用できます。`login` パラメーターが存在する場合は現在のセッションに対して検証され、存在しない場合は現在のセッションから取得されます。また、未認証のコンテキストで使用することもできます。たとえば、[トークンの作成操作](#)が、HTTP コード `422 (Unprocessable entity)` (原因: `PasswordMustChange`) で中止された場合などです。

以下を使用する必要があります。

- EBX® の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本文

URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/
```

```
user:changePassword
```

メッセージ本文

メッセージ本文はリクエストで定義する必要があります。 `password` と `passwordNew` が必須で、 `login` はオプションです (すべて 文字列)。

パスワードの変更とトークンの作成リクエストの [JSON](#) の例を参照してください。

HTTP コード

HTTP コード	説明
<code>204</code> (<i>No content</i>)	パスワードが変更されました。
<code>400</code> (<i>Bad request</i>)	原因は次のいずれかです。 - EBX® のデフォルトディレクトリが必要です。 - 構文が正しくありません。 - HTTP メソッドが <code>POST</code> ではありません。 - 指定された <code>login</code> とユーザーのセッションが一致しません。 - この操作はサポートされません。
<code>401</code> (<i>Unauthorized</i>)	原因は次のとおりです。 <code>login</code> および/または <code>password</code> が正しくありません。
<code>422</code> (<i>Unprocessable entity</i>)	原因は次のいずれかです。 <code>PasswordChangeAbort</code> : <code>passwordNew</code> が空白です。

HTTP コード	説明
	<code>PasswordChangeAbort</code> : <code>passwordNew</code> が <code>password</code> と同一。 <code>PasswordChangeAbort</code> : <code>password</code> が正しくない。 <code>RestrictedAccess</code> : ユーザーアクセスは、保守またはその他の操作のために閉じられます (管理者に予約されています)。

応答本文

HTTP コード `204 (No content)` が返された場合は、パスワードが変更されています。

トークンの無効化操作

この操作では、`POST` HTTP メソッドを使用する必要があります。メッセージ本文は必要ありません。

URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/token:revoke
```

ヘッダーフィールド

<code>Authorization</code>	このフィールドは必須です。 <code>tokenType</code> および <code>accessToken</code> フィールドには、「トークンの作成」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
204 (No content)	トークンは正常に無効化されました。
400 (Bad request)	原因は次のいずれかです。 - 構成がアクティブ化されていません。 - 構文が正しくありません。 - HTTP メソッドが POST ではありません。 - この操作はサポートされません。
401 (Unauthorized)	認証に失敗しました。

データ操作

data カテゴリの操作は、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係します。

data-compact カテゴリの操作は、データセットフィールド、テーブル、レコード、レコードフィールドに関係します。

history カテゴリの操作は、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係します。

form-data カテゴリの操作は、コンテンツの作成または更新で制約を有効に保つ必要がある場合に、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係します。

form-data-compact カテゴリの操作は、コンテンツの作成または更新で制約を有効に保つ必要がある場合に、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係します。

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。

Select 操作

Select 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

- **POST** HTTP メソッド (メッセージ本文なし)
- **POST** HTTP メソッド (メッセージ本文、**:select**、URL 拡張アクション。オプションでセッションパラメーターを含む)

URL形式は次のとおりです。

- **データセットツリー** (操作カテゴリに基づく)

data カテゴリは、選択したデータセットの階層を返します。これには、グループノードとテーブルノードが含まれます。

history カテゴリは、選択した履歴データセットの階層を返します。これには、履歴テーブルノードのプルーニングされたグループのみが含まれます。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}[:select]
```

注意:

ターミナルノードとサブノードは含まれていません。

- **データセットノード** : **data** または **data-compact** カテゴリは、選択し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]
```

注意:

history カテゴリには適用されません。

- **テーブル** (操作カテゴリに基づく)

data または **data-compact** カテゴリは、テーブルコンテンツやメタモデル、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、履歴テーブルのコンテンツやメタモデル、現在のページレコード、およびページネーションの URL を返します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}[:select]
```

以下も参照してください。:

- Count 操作
- ルックアップテーブルビュー操作

• レコード (操作カテゴリに基づく)

`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、履歴レコードのコンテンツおよび/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}
```

注意:

主キー (`primaryKey` パラメーター) によるレコードアクセスは、そのルートノードに制限されています。この制限を無効にするために、`details` フィールドで利用可能なエンコードされた主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は、`historyDetails` フィールドで使用可能なエンコードされた主キーを使用します。

- フィールド (操作カテゴリ に基づく)

`data` または `data-compact` カテゴリは、構造がその型に依存するフィールドレコードのコンテンツを返します。

`history` カテゴリは、構造がそのタイプに依存するフィールド履歴レコードの内容を返します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
[:select]
```

注意:

フィールドは、関連付けノード、選択ノード、ターミナルノード以上のいずれか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 (使用可能な値は `data` または `data-compact` です)。
- `{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くことに対応します。
- `{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は、グループノードまたはテーブルノードの場合があります。
- `{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)。
- `{xpathExpression}` は、XPath 式を使用して、レコードの主キーに対応します。
- `{pathInRecord}` は、テーブルノードから始まるパスに対応します。
- POST HTTP メソッドを本文メッセージで使用する場合は、`:select` 拡張アクションが必要です。

パラメーター

以下のパラメーターは、Select 操作に適用できます。

パラメーター	説明
<code>includeContent</code>	選択に対応するコンテンツを含む <code>content</code> フィールドが含まれます。 <code>ブール値</code> 型。デフォルト値は <code>true</code> です。
<code>includeDetails</code>	間接的に到達可能なリソースごとに、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。返される値は、その URL リソースに対応します。 <code>ブール値</code> 型。デフォルト値は <code>true</code> です。 以下も参照してください。: includeMetamodel
<code>includeHistory</code>	履歴コンテンツのこれらのフィールドが含まれます。 - メタモデルの <code>history</code> プロパティ。戻り値はブール値に対応します。 - コンテンツ内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。このポイントは、includeDetails パラメーターに結合されています。戻り値はその URL リソースに対応します。 <code>ブール値</code> 型。デフォルト値は <code>false</code> です。 注意: <code>includeHistory</code> パラメーターは <code>history</code> カテゴリでは無視されます。デフォルト値は <code>true</code> です。 以下も参照してください。:

パラメーター	説明
	• <code>includeMetamodel</code> • <code>includeDetails</code>
<code>includeLabel</code>	各単純タイプの <code>content</code> に関連付けられた <code>label</code> フィールドが含まれます。 可能な値は次のとおりです。 • <code>yes</code> : <code>label</code> は、外部キー、列挙、レコード、およびセクターの値に含まれます。 • <code>all</code> : <code>yes</code> 値、および単純型のコンテンツの場合と同様に、<code>label</code> フィールドが含まれます。 以下も参照してください。: ◦ <code>SchemaNode.displayOccurrence</code> • <code>no</code> : <code>label</code> フィールドは含まれていません (統合ユースケース)。 文字列 型。デフォルト値は <code>yes</code> です。 注意: <code>label</code> フィールドは、<code>content</code> フィールドと等しい場合は含まれません。
<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>includeMetadata</code>	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、<code>ebx-metadata</code> ルートエレメントの下にステップと呼ばれる追加エレメントが含まれています。

パラメーター	説明
	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、楽観ロックを参照してください。 文字列値 (デフォルト値は空) ステップはカンマで区切る必要があります (system, teamUp)。 ebx-all 値を使用して、すべてのステップを返すことができます。 以下も参照してください。: • 技術データ
includeMetamodel	content フィールドで返される構造の説明に対応する meta フィールドが含まれます。 ブール値 型。デフォルト値は false です。 以下も参照してください。: • メタモデル • includeHistory • includeDetails
includeMergeInfo	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する merge_info を含めます。アクセスコストが高くなる可能性があります。 ブール値 型。デフォルト値は true です。 注意: このパラメーターは、 data カテゴリでは無視されます。

パラメーター	説明
	以下も参照してください。: 履歴テーブルへの REST アクセス
<code>includeOpenApiDetails</code>	記述可能な各ノードのOpenAPI仕様URLが含まれています。 ブール値 型。デフォルト値は <code>false</code> です。 注意: <code>includeDetails</code> が <code>false</code> に設定されている場合、このクエリパラメーターは無視されます。 以下も参照してください。: - OpenAPI 操作 <code>includeDetails</code>
<code>includeSelector</code>	間接的に到達可能なリソースごとに、応答に <code>selector</code> フィールドを含めます。戻り値はその 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。 以下も参照してください。: <code>selector</code>
<code>includeSortCriteria</code>	適用されたソート基準のリストに対応する <code>sortCriteria</code> フィールドが含まれます。 ソート基準パラメーターは、以下を使用して追加されます。

パラメーター	説明
	• sort • sortOnLabel • viewPublication ブール値 型。デフォルト値は <code>false</code> です。 JSON の例
<code>includeTechnicals</code>	6.1.0 以降非推奨。 <code>includeMetadata</code> に置き換えられました。 注意: 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>includeValidation</code>	選択に対応する検証レポートを含めます。 ブール値 型。デフォルト値は <code>false</code> です。 注意: このパラメーターは、 <code>history</code> カテゴリでは無視されます。 以下も参照してください。: • includeDetails

テーブルパラメーター

次のパラメーターは、テーブル、関連、および選択ノードに適用できます。

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

パラメーター	説明
	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>文字列</code> 型の値。 注意: 履歴コードの操作値は、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。 以下も参照してください。: • 関連性で並べ替え • <code>includeMetadata</code> パラメータ 以下も参照してください。: • 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。 <code>文字列</code> 型。使用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 • <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およびマージされた子内の履歴 • <code>AllDataSpaces</code> : すべてのデータスペースの履歴

パラメーター	説明
	デフォルト値は <code>CurrentDataSpaceOnly</code> です。 以下も参照してください。: <code>history</code> 注意: 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>	非表示モードのレコードが含まれます。 ブール値 型。デフォルト値は <code>false</code> です。 以下も参照してください。: 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、主キーとそのすべてのフィールドのみを含める必要があります。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、次のいずれかとして表されます。 <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。 - その他のタイプの場合: <code>パス</code>、<code>=</code> 演算子、および <code>値</code> を指定します。 構成された主キーの例: <code>./pk1i=1 and date-equal(./pk2d, '2015-11-13')</code>

パラメーター	説明
	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。したがって、他のテーブルパラメーターは無視されます (filter、viewPublication、sortなど)。 <code>文字列</code> 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。このフィルターは強く <code>pageAction</code> 値にリンクされているため、クライアント側で変更しないでください。フィルターは、ページネーションコンテキストを把握するために使用されるレコード XPath 述語式の形式を取ります。 <code>文字列</code> 型。 以下も参照してください。: • ページ付け
<code>pageAction</code>	<code>pageRecordFilter</code> が保持する識別子から実行するページ機能の操作を指定します。 <code>文字列</code> 型。デフォルト値は <code>first</code> です。可能な値は次のとおりです。 • <code>first</code> • <code>previous</code> • <code>next</code> • <code>last</code> 以下も参照してください。:

パラメーター	説明
	RequestPagination
pageSize	1 ページあたりのレコードの最大数を指定します。 整数 型。デフォルト値はユーザー設定に基づいています。指定された値が 0 の場合、推奨される最大ページサイズで選択します。デフォルトは 10000 です。 以下も参照してください。: ebx.dataservices.pagination.maxSize.default 注意: 指定したページサイズの値が最大ページサイズを超える場合は、最大ページサイズが選択されます。
sort	指定した基準に基づく操作結果の並べ替えを指定します。条件は 1 つ以上の基準で構成され、結果は左から優先度で並べ替えられます。基準は、フィールドパスと、オプションで並べ替え順序 (値またはラベルの昇順または降順) で構成されます。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。 - sortOnLabel パラメーター。新しい基準として、sort の後に追加されます。 - sort の後に追加された新しい基準としての viewPublication パラメーター。 値の構造は次のとおりです。 <pre><path1>:<order>;...;<pathN>:<order></pre>

パラメーター	説明
	説明 • <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、次のいずれかの値を持つ並べ替え順序に対応します。 ◦ <code>asc</code> : 値の昇順 (デフォルト) ◦ <code>desc</code> : 値の降順 ◦ <code>lasc</code> : ラベルの昇順 ◦ <code>ldesc</code> : ラベルの降順 文字列 型。デフォルト値は、主キーフィールドに従って順序付けられます (値の昇順)。 注意: 履歴コードの操作値は、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。 注意: 関連性による並べ替えがアクティブになっている場合、このパラメーターは無視されます。 注意: <code>includeMetadata</code> パラメータ 以下も参照してください。: • <code>Request.setSortCriteria</code>
<code>sortByRelevancy</code>	次の条件が満たされた場合にのみ、操作結果の関連性ソートが指定されます。

パラメーター	説明
	- クイック検索は、<code>filter</code> パラメーターで直接 <code>osd:search</code> 述語を指定することで有効化されます。 - 要求は、継承されたデータセット、履歴テーブル、またはマップされたテーブルには適用されません。 <code>文字列</code> 型。可能な値は次のとおりです。 <code>lasc</code> : ラベルの昇順 <code>ldesc</code> : ラベルの降順 関連性による並べ替え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：<code>sort</code>、<code>sortOnLabel</code>、<code>sortPriority</code>、および <code>viewPublication</code> で定義された並べ替え条件。 以下も参照してください。: - 検索述語
<code>sortOnLabel</code>	レコードラベルに基づく操作結果の並べ替えを指定します。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。 <code>sortOnLabel</code> の前に追加された新しい基準としての <code>sort</code> パラメーター。 <code>viewPublication</code> パラメーター。新しい基準として <code>sortOnLabel</code> の後に追加されます。 値の構造は次のとおりです。 <code><order></code> 説明

パラメーター	説明
	• <code><order></code> は、次のいずれかの値を持つ並べ替え順序に対応します。 ◦ <code>lasc</code> : ラベルの昇順 ◦ <code>ldesc</code> : ラベルの降順 このパラメーターの動作については、defaultLabel セクションに説明が記載されています。 <code>文字列</code> 型の値。 以下も参照してください。: • 制限事項 注意: 関連性による並べ替えがアクティブになっている場合、このパラメーターは無視されます。
<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。 • <code>sort</code> • <code>sortOnLabel</code> • <code>sortFromView</code> カンマ区切りの <code>文字列</code> 型。デフォルト値は <code>sort,sortOnLabel,sortFromView</code> です。 注意: 関連性による並べ替えがアクティブになっている場合、このパラメーターは無視されます。

パラメーター	説明
<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. 論理的な AND 操作としての <code>filter</code> パラメーター。 2. <code>viewPublication</code> の前に追加された新しい基準としての <code>sort</code> パラメーター。 3. <code>viewPublication</code> の前に追加された新しい基準としての <code>sortOnLabel</code> パラメーター。 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。 <code>文字列</code> 型の値。 注意: 関連性ソートが有効の場合、このパラメーターで定義されたソート基準は無視されます。 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

セレクトパラメーター

次のパラメーターは、列挙型、外部キー、または `osd:resource` を返すフィールドにのみ適用できます (例、[JSON](#))。デフォルトでは、ページ機能は常に有効です。一部のセレクトの Select 操作には、入力値が必要です。したがって、`POST` HTTP メソッドの[メッセージ本文](#)を使用して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>	次のいずれかを指定します。 <code>true</code> : 使用可能なすべての値を返します (ラベルを含む)。 <code>false</code> : 現在のフィールドの現在の値を返します。 <code>ブール値</code> 型。デフォルト値は <code>false</code> です。 注意: 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>	セレクターによって返される最初のエレメント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。 <code>整数</code> 型。デフォルト値は <code>0</code> です。
<code>pageSize</code>	ページあたりの最大エレメント数を指定します。 <code>整数</code> 型。デフォルト値はユーザー設定に基づいています。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で選択します。デフォルトは 10000 です。 以下も参照してください。: ebx.dataservices.pagination.maxSize.default 注意: 指定したページサイズの値が最大ページサイズを超える場合は、最大ページサイズが選択されます。

パラメーター	説明
<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ています。

HTTP コード

HTTP コード	説明
<code>200</code> (OK)	選択したリソースが正常に取得されます。
<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。 - レコードまたはデータセットで選択されたフィールドはサブターミナルです。 <code>filter</code> パラメーターの XPath 述語のフォーマットが正しくないか、フィルター不可能なノードが含まれています。 <code>primaryKey</code> パラメーターの XPath 述語のフォーマットが正しくないか、レコードの主キーではありません。 <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ます。 <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か、次ページまたは前ページの選択時に <code>pageRecordFilter</code> のフォーマットが正しくないか存在しません。 <code>pageSize</code> パラメーター値が 2 未満です。

HTTP コード	説明
	• <code>viewPublication</code> パラメーターのテーブルビューが、階層、存在しない、または非公開のいずれかです。 • <code>selector</code> パラメーターが列挙されていないノードに使用されているか、<code>firstElementIndex</code> が負の数または値の数以上です。
<code>403</code> (<i>Forbidden</i>)	選択したリソースは、認証されたユーザーに対して非表示になります。
<code>404</code> (<i>Not found</i>)	選択したリソースが見つかりません。

応答本文

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すると、結果が応答本文に返されます。内容は、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、初期コンテンツまたは複製するレコードのコンテンツを含む新しい一時レコードを作成するために使用されます。一時的なレコードは、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。少なくとも読み取り専用のアクセス権を持つフィールド値のみが返されます。これらの操作は、オプションでメタデータを返し、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できます。自動インクリメントされたフィールドは、メタデータでのみ返されます。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、列挙フィールドや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 Select 操作により、入力レコードデータを提供し、カスタムプログラム列挙でメタデータを取得するなど、使用を管理できます。

以下も参照してください。:

- `TableTrigger.handleNewContext`

以下も参照してください。:

- デフォルト値

クライアント側で変更後、ビルトインのフォーム挿入操作または **Insert** 操作を使用して、レコードを永続化する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
- **POST** HTTP メソッド (メッセージ本文なし)
- **POST** HTTP メソッド (メッセージ本文あり)。オプションでセッションパラメーターを使用。

使用可能な URL フォーマットは次のとおりです。

- 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}
```

説明

- `{category}` は操作カテゴリに対応します (使用可能な値は `data` または `data-compact` です)。
- `{dataspace}` は、`B` およびデータスペース識別子に対応します。
- `{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

次のパラメーターは、準備操作に適用できます。

パラメーター	説明
<code>includeDetails</code>	間接的に到達可能なリソースごとに、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。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。 以下も参照してください。: • includeMetamodel
<code>includeMeta</code>	6.1.0 以降非推奨。 includeMetamodel に置き換えられました。

パラメーター	説明
<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> フィールドが含まれます。 ブール値 型。デフォルト値は <code>false</code> です。 以下も参照してください。: メタモデル
<code>includeSelector</code>	間接的に到達可能なリソースごとに、コンテンツに <code>selector</code> フィールドを含めます。戻り値はその 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。 以下も参照してください。: <code>selector</code>

セレクトターパラメーター

使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。

以下も参照してください。:

- セレクトターパラメーター

HTTP コード

HTTP コード	説明
<code>200</code> (OK)	選択したリソースは正常に取得されました。

HTTP コード	説明
400 (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。 - レコードで選択されたフィールドはサブターミナルです。 <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す。 <code>selector</code> パラメーターが列挙されていないノードに使用されているか、<code>firstElementIndex</code> が負の数または値の数以上です。
401 (Unauthorized)	認証に失敗しました。
403 (Forbidden)	選択したリソースは、認証済みユーザーに表示されません。
404 (Not found)	選択したリソースが見つかりませんでした。

応答本文

作成または複製要求の準備が正常に完了すると、一時レコードが応答本文に返されます。内容は、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、Select 操作レコードと類似のフォーマットを取ります。

例：拡張 JSON、コンパクト JSON。

Inset 操作

Insert 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、本文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の 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できます。

- 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：一貫した応答を確保しながら、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ます。操作は、クライアント側で定義された順序で順番に実行されます。

テーブル操作中にエラーが発生すると、すべての更新がキャンセルされ、クライアントは詳細情報を含むエラーメッセージを受信します。

```
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 (使用可能な値は `data` または `data-compact` です)。
- `{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くことに対応します。
- `{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、フィルター処理された一連のレコードを更新するときに必要です ([filter 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、Insert 操作に適用できます。

パラメーター	説明
<code>includeDetails</code>	データの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。戻り値は、その URL リソースに対応しています。 ブール値 型。デフォルト値は <code>false</code> です。 注意: レコードテーブルにのみ適用されます。 以下も参照してください。: • Insert 操作レポート

パラメーター	説明
<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。戻り値は、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。 <code>ブール値</code> 型。デフォルト値は <code>false</code> です。 注意: レコードテーブルにのみ適用されます。 以下も参照してください。: • Insert 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。 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>文字列</code> 型。デフォルト値は <code>no</code> です。 注意: レコードテーブルにのみ適用されます。 以下も参照してください。: • Insert 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。

パラメーター	説明
	• <code>true</code> の場合：既存のレコードが新しいデータで更新されます。 レコードテーブルでのリクエストの場合、これが挿入 (201) であるか更新 (204) であるかを指定するために、<code>code</code> フィールドがレポートに追加されます。 • <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。 ブール値 型の値。
<code>byDelta</code>	リクエスト本文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。これについては、更新モードセクションに説明が記載されています。 ブール値 型。デフォルト値は <code>true</code> です。 注意: 更新モードのレコードで、<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>	多数のレコードが更新される場合は、クイック検索述語、または完全な XPath 述語 を使用して、更新されるレコードを決定します。リクエスト本文は、更新するフィールドを含む record 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。それ以外の場合は、エラーが返されます。

パラメーター	説明
	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を使用すると、スコープはフィルターが設定されたビューに限定されます。 文字列 型の値。空にすることはできません。空の場合、エラーが返されます。 注意: クエリパラメーター includeDetails、includeLabel、または includeForeignKey を使用すると、最初の 1000 件のレポートエントリのみがレポートに返されます。
viewPublication	大規模な更新中に考慮される公開済みビューの名前を指定します。 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 で説明されています。 文字列 型の値。
blockingConstraintsDisabled	ブロッキング制約を無視するかどうかを指定します。存在する場合、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。それ以外の場合、操作は中止されます。 ブール値 型。デフォルト値は false です。 詳細については、ブロッキングと非ブロッキングの制約を参照してください。

メッセージ本文

リクエストはメッセージ本文を定義する必要があります。フォーマットは、挿入されたオブジェクトタイプによって異なります。

- **レコード**：レコードの Select 操作に似ていますが、レコードのヘッダーはありません (例：拡張JSON、コンパクトJSON)。
- **レコードテーブル**：テーブルの Select 操作に類似していますが、ページ情報はありません (例：拡張JSON、コンパクトJSON)。

以下も参照してください。：

- [継承](#)

HTTP コード

HTTP コード	説明
200 (OK)	リクエストがレコードテーブルに関連している場合。 挿入要求が正常に適用され、オプションのレポートが応答本文に返されます。
201 (Created)	リクエストがレコードに関連している場合。 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 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している場合。 updateOrInsert が true の場合にのみ使用可能で、既存のレコードは正常に更新されています。この場合、ヘッダーフィールド

HTTP コード	説明
	<code>Location</code> がリソース URL とともに返されます。
<code>400</code> (<i>Bad request</i>)	リクエストが正しくありません。これは、本文のメッセージ構造がメッセージ本文に記載されている内容に準拠していない場合に発生します。
<code>403</code> (<i>Forbidden</i>)	認証済みユーザーはレコードの作成が許可されていないか、リクエスト本文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>)	選択したリソースが見つかりません。
<code>409</code> (<i>Conflict</i>)	同時変更。 <code>updateOrInsert</code> が <code>true</code> であり、楽観ロックが有効で、その間にコンテンツが変更された場合にのみ使用できます。更新前にコンテンツをリロードする必要があります。
<code>422</code> (<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ます (<code>blockingConstraintsDisabled</code> が <code>false</code> の場合にのみ使用可能)。 • 同一の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ません (<code>updateOrInsert</code> が <code>false</code> の場合にのみ使用可能)。 • 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。 • 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。

応答本文

応答本文のフォーマットは、挿入されたオブジェクト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は空です。ヘッダーフィールド `Location` は、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) Insert 操作レポートに対応するエレメントのテーブルが含まれます (例：JSON)。次のオプションの少なくとも 1 つが設定されている場合、このレポートは自動的にレスポンス本文に含まれます。
 - `includeForeignKey`
 - `includeLabel`
 - `includeDetails`

以下も参照してください。：

- [継承](#)

Update 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を変更できます。 `PUT` HTTP メソッドを使用する必要があります。使用可能な URL フォーマットは次のとおりです。

- **データセットノード**：選択したノードに含まれるターミナル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}
```

注意:

`POST` HTTP メソッドでも使用できます。この場合、URL はテーブルを指す必要があります、パラメーター `updateOrInsert` は `true` に設定されている必要があります。

注意:

`updateOrInsert=true` および `byDelta=true` パラメーターを指定して `Insertoperation` を使用して、複数のレコードのコンテンツを変更します。

- フィールド：選択したレコードの単一フィールドの更新。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
```

注意:

フィールドは、ターミナルノード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 (使用可能な値は `data` または `data-compact` です)。
- `{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くことに対応します。
- `{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

Update 操作に適用できる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。無視する場合は、作成された検証エラーに関係なく操作をコミットします。無視しない場合、操作は中止されます。 <code>ブール値</code> 型。デフォルト値は <code>false</code> です。詳細については、 ブロッキングと非ブロッキングの制約 を参照してください。
<code>byDelta</code>	リクエスト本文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。これについては、 更新モード セクションに説明が記載されています。 <code>ブール値</code> 型。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フォーマットのタイムスタンプ。 楽観ロック セクションも参照してください。 <code>DateTime</code> 型の値。

メッセージ本文

リクエストはメッセージ本文を定義する必要があります。

構造は、次の場合は同一です。

- データセットノード (例 [拡張 JSON](#)、[コンパクト JSON](#))
- レコード (例：[拡張 JSON](#)、[コンパクト JSON](#))
- レコードフィールド (例：[拡張 JSON](#)、[コンパクト JSON](#))

更新された範囲に応じて、`content` エントリのみを保持します。

以下も参照してください。:

- [継承](#)

HTTP コード

HTTP コード	説明
204 (No content)	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。
400 (Bad request)	リクエストが正しくありません。これは、本文のリクエスト構造が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、指定されたリソースを更新できません。または、リクエスト本文に読み取り専用フィールドが含まれています。
404 (Not found)	選択したリソースが見つかりません。
409 (Conflict)	同時変更。 楽観ロック が有効で、その間にコンテンツが変更された場合は、更新前にリ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。 • ブロッキング検証エラーが発生します (<code>blockingConstraintsDisabled</code> が <code>false</code> の場合にのみ使用可能)。 • 主キーの値を変更できないため、レコードを更新できません。

Delete 操作

この操作では、`DELETE` HTTP メソッドを使用します。

2 つの URL フォーマットを使用できます。

- **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**：一貫した応答を提供しながら、指定されたテーブル内の複数のレコードを削除します。このモードでは、レコードテーブルを含む本文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番に実行されます。テーブル操作中にエラーが発生すると、すべての削除がキャンセルされ、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、`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くことに対応します。
- `{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、フィルター処理された一連のレコードを削除するときに必要です ([filter パラメーター](#) を参照)。

子データセットコンテキストでは、この操作により、レコードの `inheritanceMode` プロパティ値が次のように変更されます。

- 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。
- `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、継承モードが `occult` に設定されているレコードは `inherit` になります、それ以外の場合は変更はありません。
- 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。

以下も参照してください。:

- 継承

パラメーター

Delete 操作に適用できる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>	非表示のレコードを含みます。 <code>ブール値</code> 型。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。*下位互換性の理由から引き続き使用できますが、今後のバージョンでは最終的に削除される予定です。非表示モードの場合、レコードを継承します。 <code>ブール値</code> 型。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フォーマットのタイムスタンプ。 楽観ロック 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。無視する場合は、作成された検証エラーに関係なく操作をコミットします。無視しない場合、操作は中止されます。 <code>ブール値</code> 型。デフォルト値は <code>false</code> です。詳細については、 ブロッキングと非ブロッキングの制約 を参照してください。
<code>filter</code>	一括削除が適用されるレコードを定義するクイック検索述語または完全な XPath 述語式 。このパラメーターは、 <code>:mass</code> 拡張アクションで強制的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。 <code>ebx-all</code> 値は、すべてのレコードを選択するために使用されます。 viewPublication クエリパラメーターを使用すると、スコープはフィルターが設定されたビューに限定されます。 <code>文字列</code> 型の値。空に

パラメーター	説明
	することはできません。空の場合、エラーが返されます。
<code>viewPublication</code>	大規模な削除中に考慮される公開済みビューの名前を指定します。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。 <code>文字列</code> 型の値。

メッセージ本文

`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、リクエストはメッセージ本文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エレメントのテーブルが含まれ、エレメントごとに次のプロパティのいずれかが含まれます。
 - `details`：レコードの URL に対応し、Select 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。
 - `externalKey`：レコードを参照した場合に外部キーが持つ値に対応します。

以下も参照してください。:

- `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。レポートは応答本文に返されます。
<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。

HTTP コード	説明
	- メッセージ本文の構造がメッセージ本文に準拠していません。 - メッセージ本文にはレコードテーブルが含まれ、URL はレコードを指定します。
403 (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非表示にすることはできません。
404 (Not found)	選択し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ります。
409 (Conflict)	同時変更。楽観ロックが有効で、その間にコンテンツが変更された場合は、レコードの削除前にコンテンツをリロードする必要があります。 パラメータ値 <code>checkNotChangedSinceLastUpdateTime</code> は存在しますが、レコードの実際の最終更新日に対応していません。
422 (Unprocessable entity)	<code>blockingConstraintsDisabled</code> が <code>false</code> の場合にのみ使用可能で、ブロッキング検証エラーにより操作が失敗します。

応答本文

レコードの削除または非表示化が成功すると、応答本文にレポートが返されます。これには、削除、非表示化、継承されたレコードの数が含まれます。

例：JSON。

Count 操作

Count 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
- `POST` HTTP メソッド (メッセージ本文なし)
- `POST` HTTP メソッド (メッセージ本文はあるがルートに `content` フィールドがない)

URL フォーマットは次のとおりです。

- **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ターミナルノードの数を返します。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}:count
```

注意:

`history` カテゴリには適用されません。

- **テーブル (操作カテゴリに基づく)**

`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 (操作カテゴリに基づく)**

`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
```

注意:

フィールドは、関連付けノード、選択ノード、ターミナルノード以上のいずれか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 (使用可能な値は `data` または `data-compact` です)。
- `{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くことに対応します。
- `{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は、グループノードまたはテーブルノードの場合があります。
- `{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

以下のパラメーターは、Count 操作に適用できます。

パラメーター	説明
<code>count</code>	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 Count 操作か Select 操作かを指定するために使用されます。 <code>ブール値</code> 型。デフォルト値は <code>false</code> です。

テーブルパラメーター

次のパラメーターは、テーブル、関連、および選択ノードに適用できます。

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。 文字列 型の値。 注意: 履歴コードの操作値は、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。 文字列 型。使用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 <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およびマージされた子内の履歴 <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。 以下も参照してください。: <code>history</code> 注意:

パラメーター	説明
	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>	非表示モードのレコードが含まれます。 ブール値 型。デフォルト値は <code>false</code> です。
<code>viewPublication</code>	カウントの実行中に考慮される公開済み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。 論理的な AND 操作としての <code>filter</code> パラメーター。 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。 文字列 型の値。

セレクトーパラメーター

次のパラメーターは、列挙型、外部キー、または `osd:resource` を返すフィールドにのみ適用できます。

パラメーター	説明
<code>selector</code>	次のいずれかを指定します。 <code>true</code> : 使用可能なすべての値の数を返します。 <code>false</code> : 現在のフィールドで可能な値の数を返します。 ブール値 型。デフォルト値は <code>false</code> です。

パラメーター	説明
	注意: 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ています。

HTTP コード

HTTP コード	説明
<code>200</code> (OK)	選択したリソースは正常にカウントされます。
<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。 - レコードまたはデータセットで選択されたフィールドはサブターミナルです。 - 選択したデータセットフィールドがデータセットツリーです。 <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 述語のフォーマットが正しくないか、フィルター不可能なノードが含まれています。 <code>viewPublication</code> パラメーターのテーブルビューが、階層、存在しない、または非公開のいずれかです。 <code>selector</code> パラメーターが列挙されていないノードに使用されているか、<code>firstElementIndex</code> が負の数または値の数以上です。

HTTP コード	説明
403 (<i>Forbidden</i>)	選択したリソースは、認証されたユーザーに対して非表示になります。
404 (<i>Not found</i>)	選択したリソースが見つかりません。

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 操作を回避するために、楽観ロックメカニズムが提供されます。

楽観ロックを有効にするには、Select 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。

次のリクエストには、`update_time` プロパティ値を含める必要があります。指定した時間以降にレコードが変更された場合、Update または Delete 操作はキャンセルされます。

- **レコード**：選択したレコードのコンテンツ全体または一部を更新します。

`update_time` 値をリクエスト本文に追加して、変更されたレコードが更新されないようにする必要があります。

レコードの [JSON](#) の例を参照してください。

- **フィールド**：選択したレコードの単一フィールドの更新。

`update_time` 値は、変更されたレコードの更新を防ぐために、

`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` プロパティの値をリクエスト URL の `checkNotChangedSinceLastUpdateTime` パラメーターで使用する、変更されたレコードが削除されないようにすることもできます。

注意:

`checkNotChangedSinceLastUpdateTime` パラメーターは複数回使用できますが、同じレコードでのみ使用できます。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。

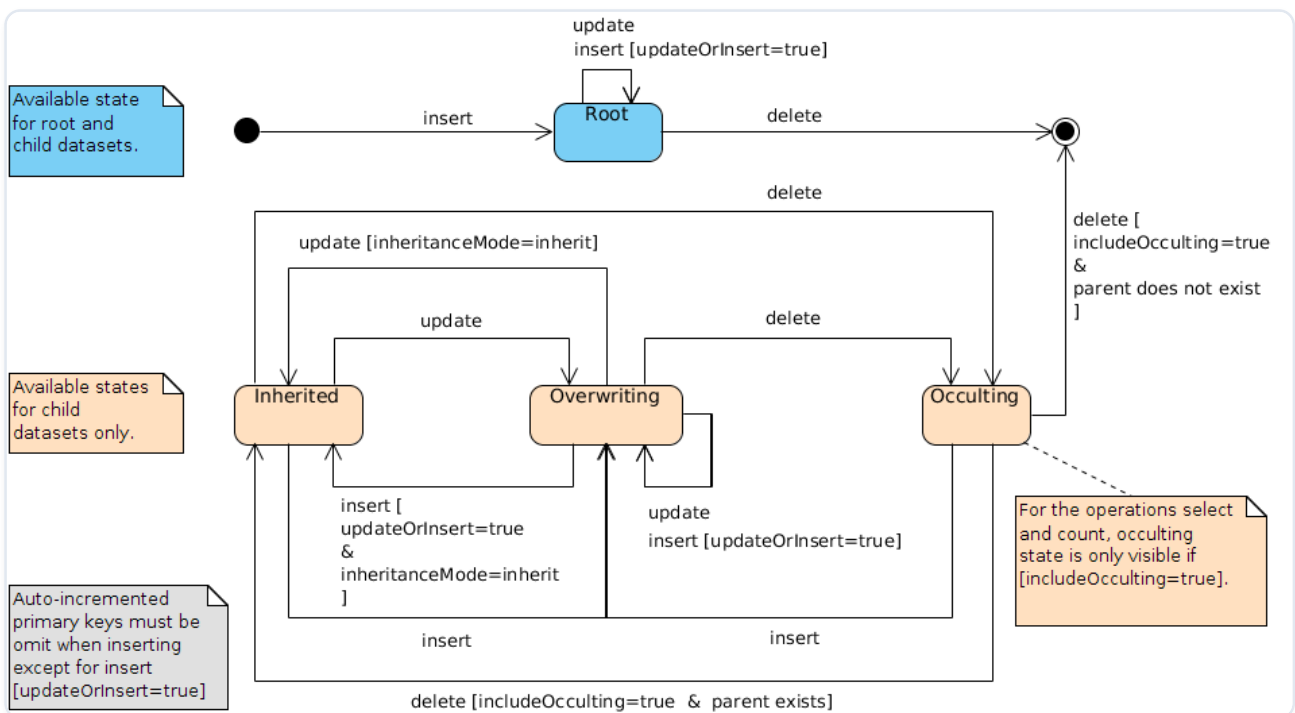
継承

EBX® の継承機能は、特定のプロパティと自動動作を使用するビルトイン RESTful サービスによってサポートされます。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、それらの状態に間接的な影響を与える可能性があります。継承の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を直接変更することが許可されています。禁止または一貫性のない明示的な変更の試みは無視されます。

以下も参照してください。:

- 継承と値の解決

ビルトイン RESTful サービスでの継承ライフサイクル



継承プロパティ

下表は、EBX® の継承機能に関連するプロパティを示しています。

プロパティ	位置	説明
<code>inheritance</code>	レコードまたはテーブルのメタモデル	データセットの継承がテーブルで有効にするかどうかを指定します。値はデータモデルから計算され、ビルトイン RESTful サービスを介して変更することはできません。 以下も参照してください。: メタモデルの <code>inheritance</code> プロパティ。
<code>inheritedField</code>	フィールドメタモデル	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、ビルトイン RESTful サービスを介して変更することはできません。 以下も参照してください。: 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

プロパティ	位置	説明
		定のケースでは、<code>content</code> プロパティが存在する場合は無視されます。<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> プロパティが存在しない場合の明示的な <code>overwrite</code> の値は無視されます。 注意: 継承されたレコードのフィールドは、常に <code>inherit</code> です。 注意: 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、レコードルックアップメカニズムを参照してください。
	<code>上書き</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> プロパティが存

プロパティ	位置	説明
		在しない場合の明示的な <code>overwrite</code> の値は無視されます。 注意: フィールドレベルの <code>inheritanceMode</code> は、<code>root</code>、<code>inherit</code>、および <code>occult</code> レコードには表示されません。 注意: <code>inheritedFieldMode</code> プロパティと <code>inheritanceMode</code> プロパティの両方を同一フィールドに設定することはできません。 使用可能な値は、<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> プロパティが存在しない場合の明示的な

プロパティ	位置	説明
		<code>overwrite</code> の値は無視されます。 注意: <code>inheritedFieldMode</code> プロパティと <code>inheritanceMode</code> プロパティの両方を同一フィールドに設定することはできません。 注意: <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、値ルックアップメカニズムを参照してください。

ルックアップテーブルビュー操作

「公開されたビューの検索」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
- `POST` HTTP メソッド (メッセージ本文なし)

URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/
{dataset}/{tablePath: [&#94;:]*}:publishedViews
```

説明

- `{dataspace}` は、`B` とデータスペース識別子、または `V` と画像識別子 (この順序) 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

この操作に特定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。
<code>400</code> (Bad request)	不正なリクエスト、エラーが含まれています。
<code>401</code> (Unauthorized)	認証に失敗しました。
<code>403</code> (Forbidden)	指定されたリソースの読み取り許可は、認証済みユーザーに対して拒否されました。
<code>404</code> (Not found)	選択したリソースが見つかりません。

応答本文

これには、許可されたビュー情報 (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がユーザーフォーム送信の制約に準拠する必要がある場合に、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係します。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。

操作のリクエスト本文は、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るという意味で、`data` カテゴリの同等の操作と非常によく似ています。データ検証を無効にすることはできず、`data` カテゴリ操作のパラメーター `blockingConstraintsDisabled` は適用されません。`blocksCommit` レベルが `onInsertUpdateOrDelete` または `onUserSubmit-`

`checkModifiedValues` に設定した少なくとも 1 つの制約に違反すると、検証フェーズは失敗します。

詳細については、[ブロッキングと非ブロッキングの制約](#)を参照してください。

フォーム挿入操作

フォームの挿入では、`POST` HTTP メソッドを使用し、データを保持するメッセージ本文を必要が必要です。この操作は、単一ランザクション内の 1 つまたは複数のレコードをサポートします。また、レコードを更新することもできます。

- **レコード**：新しいレコードを検証して挿入するか、選択したテーブルの既存のレコードを変更します。
- **レコードテーブル**：一貫した応答を確保しながら、選択したテーブルの 1 つ以上のレコードを検証および挿入または変更します。操作は、クライアント側で定義された順序で実行されます。テーブル操作中にエラーが発生すると、すべての更新がキャンセルされ、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]
```

説明

- `{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、`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くことに対応します。
- `{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、フィルター処理された一連のレコードを更新するときに必要です ([filter パラメーター](#)を参照)。

パラメーター

この挿入操作には、次のパラメーターが適用されます。

パラメーター	説明
<code>includeDetails</code>	データの詳細にアクセスするため、応答に <code>details</code> フィールドを含めます。戻り値は、その URL リソースに対応しています。 ブール値 型。デフォルト値は <code>false</code> です。 注意: 複数レコードの挿入にのみ適用されます。
<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。戻り値は、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。 ブール値 型。デフォルト値は <code>false</code> です。 注意: 複数レコードの挿入にのみ適用されます。
<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。 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)。 文字列 型。デフォルト値は <code>no</code> です。 注意: 複数レコードの挿入にのみ適用されます。

パラメーター	説明
<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：既存のレコードが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、操作が中止されます。 <code>ブール値</code> 型。デフォルト値は <code>false</code> です。
<code>filter</code>	クイック検索述語、または大量のレコードが更新される場合は完全な <code>XPath 述語</code> 式を使用して更新されたレコードを特定します。リクエストの本文は、更新するフィールドを含む record 構造である必要があります。このパラメータは、:mass 拡張アクションを使用する必要があります。それ以外の場合は、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と、スコープはフィルターが設定されたビューに限定されます。 <code>文字列</code> 型の値。空にすることはできません。空の場合、エラーが返されます。 注意: クエリパラメーター <code>includeDetails</code>、<code>includeLabel</code>、または <code>includeForeignKey</code> を使用すると、最初の 1000 件のレポートエントリのみがレポートに返されます。

パラメーター	説明
<code>viewPublication</code>	大規模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については、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>文字列</code> 型の値。

メッセージ本文

リクエストはメッセージ本文を定義する必要があります。フォーマットは、`data` カテゴリ Insert 操作の[メッセージ本文](#)に似ています。

HTTP コード

HTTP コード	説明
<code>200</code> (OK)	- リクエスト本文には複数のレコードが含まれています。Insert/Update リクエストが正常に適用され、レポートが応答本文に返されます。 - リクエスト本文は 1 つのレコードのみを保持し、<code>updateOrInsert</code> は <code>true</code> です。Update リクエストは正常に適用され、レポートが応答本文に返されます。
<code>201</code> (Created)	リクエストの本文には、存在しないレコードが 1 つだけ保持されます。新しいレコードが作成され、ヘッダーフィールドの <code>Location</code> がそのリソース URL とともに返されます。さらに、レポートは応答本文に返されます。

HTTP コード	説明
400 (<i>Bad request</i>)	リクエストが正しくありません。これは、本文のメッセージ構造が以下の場合に発生します。 • メッセージ本文の記述に準拠していません。 • メッセージ本文に記載されている内容に準拠していますが、レコードの Insert/Update の数が 100 を超えています。
403 (<i>Forbidden</i>)	認証済みユーザーはレコードの作成が許可されていないか、リクエスト本文に読み取り専用フィールドが含まれています。
404 (<i>Not found</i>)	選択したリソースが見つかりません。
409 (<i>Conflict</i>)	同時変更。 <code>updateOrInsert</code> が <code>true</code> であり、楽観ロックが有効で、その間にコンテンツが変更された場合にのみ使用できます。更新前にコンテンツをリロードする必要があります。
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。 • ブロッキング制約に違反しています。この場合、適切なレポートが応答本文に返されます。 • 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ません (<code>updateOrInsert</code> が <code>false</code> の場合にのみ使用可能)。 • 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。

HTTP コード	説明
	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。

応答本文

応答本文は常に検証レポートを保持します。ただし、失敗した場合、応答本文は JSON **例外処理** 応答に対応します。

- レコード：ヘッダーフィールド **Location** が 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本文に含まれます。
- レコードテーブル：(オプション) 複数の検証レポートに対応する、各エレメントの検証レポートのリストが含まれます。

以下も参照してください。:

- 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 Update 操作については、単一のデータセットまたはレコードの変更が可能であり、**PUT** HTTP メソッドを使用する必要があります。使用可能な URL フォーマットは次のとおりです。

- 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：選択し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}
```

注意:

`POST` HTTP メソッドでも使用できます。この場合、URL はテーブルを指す必要があります、パラメーター `updateOrInsert` は `true` に設定されている必要があります。

- **フィールド**：選択し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}
```

注意:

フィールドは、ターミナルノード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 (可能な値は `form-data` または `form-data-compact` です)。
- `{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くことに対応します。
- `{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

Update 操作に適用できるパラメーターは次のとおりです。

パラメーター	説明
byDelta	リクエスト本文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。これについては、 更新モード セクションに説明が記載されています。 <code>ブール値</code> 型。デフォルト値は <code>true</code> です。
checkNotChangedSinceLastUpdateTime	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フォーマットのタイムスタンプ。 楽観ロック セクションも参照してください。 <code>DateTime</code> 型の値。

メッセージ本文

リクエストはメッセージ本文を定義する必要があります。フォーマットは、 `data` カテゴリ更新操作[メッセージ本文](#)と同じです。

HTTP コード

HTTP コード	説明
200 (Ok)	更新要求が正常に適用され、応答本文にレポートが返されます。
400 (Bad request)	リクエストが正しくありません。これは、本文のリクエスト構造が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、指定されたリソースを更新できません。または、リクエスト本文に読み取り専用フィールドが含まれています。
404 (Not found)	選択したリソースが見つかりません。
409 (Conflict)	同時変更。 楽観ロック が有効で、その間にコンテンツが変更された場合は、更新前にリ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。 • ブロッキング制約に違反しています。この場合、適切なレポートが応答本文に返されます。 • 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。

応答本文

レスポンス本文のフォーマットと動作は、[フォーム挿入操作](#)と同じです。

ベータ機能：データスペース操作

これらの操作は、データスペースの選択またはライフサイクル管理を実行します。

以下も参照してください。:

- [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

Select 操作では、`GET` または `POST` のいずれかのメソッドを使用できます。ページネーションメカニズムは常に有効になっています。

URL フォーマットは次のとおりです。

- **Root**：ルートデータスペースを返します。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/
```

- **Children**：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
```

- **Snapshots** : 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
```

- **Information** : 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
```

説明

- `{dataspace}` は、`B` の後にデータスペースの識別子が続くか、`v` の後にスナップショットの識別子が続くことに対応します。

パラメーター

次のクエリパラメーターは、** Root**、** Children**、および **Snapshots** の操作に適用できます。

パラメーター	説明
<code>includeClosed</code>	閉じたデータスペースを選択範囲に含めます。 ブール値 型。デフォルト値は <code>false</code> です。
<code>includeAdministration</code>	選択に管理データスペースを含めます。 ブール値 型。デフォルト値は <code>false</code> です。
<code>pageRecordFilter</code>	レコードを指す、ページネーション用のサーバー側で構築されたフィルターを指定します。このフィルターは強く <code>pageAction</code> 値

パラメーター	説明
	にリンクされているため、クライアント側で変更しないでください。フィルターは、ページネーションコンテキストを把握するために使用されるレコード XPath 述語式の形式を取ります。 文字列 型。 以下も参照してください。: ページ付け
pageAction	pageRecordFilter が保持する識別子から実行するページ機能の操作を指定します。 文字列 型。デフォルト値は first です。可能な値は次のとおりです。 - first - previous - next - last 以下も参照してください。: - RequestPagination
pageSize	1 ページあたりのレコードの最大数を指定します。 整数 型。デフォルト値はユーザー設定に基づいています。値は 2 から 100 の間である必要があります。

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。
400 (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。 • <code>pageAction</code> パラメーターの値は許可された値に含まれません。 • <code>pageRecordFilter</code> は、次または前のページを選択するときにフォーマットが正しくないか、存在しません。 • <code>pageSize</code> パラメーターの値が範囲外です。
401 (Unauthorized)	認証に失敗しました。
403 (Forbidden)	選択したリソースは、認証済みユーザーに表示されません。
404 (Not found)	選択したリソースが見つかりませんでした。

応答本文

選択の完了後、結果がレスポンス本文に返されます。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。

フォーマットは、選択し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については、[JSON](#) の例を参照してください。
- 情報については、[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

指定されたとおりにデータスペースまたはスナップショットを作成します。この操作では、本文リクエストで `POST` メソッドを使用します (特定のクエリパラメーターはありません)。

以下も参照してください。:

- `Repository.createHome`

URL フォーマットは次のとおりです。

- 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
```

説明

- `{dataspace}` は、`B` およびデータスペース識別子に対応します。

リクエスト本文

本文は、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。

以下も参照してください。:

- `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。

HTTP コード	説明
400 (Bad request)	リクエストの本文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。
401 (Unauthorized)	認証に失敗しました。
403 (Forbidden)	認証されたユーザーに指定されたリソースを作成する権限が拒否されました。
404 (Not found)	リクエスト本文で指定されたリソースが見つかりません。
500 (Internal error)	アプリケーションによる予期しないエラー。エラーの詳細は通常、EBX® ログに記録されます。

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、次のことが適用されます。

- 同一ユーザーによるロックでは、ロックが保持されます。
- 別のユーザーによるロックで待機期間中の場合、ロックが取得されます。
- 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、次のように設定された `Content-Type` ヘッダーを使用します。

- `application/x-www-form-urlencoded` : 本文に HTTP パラメーターを使用します。
- `application/json` : URL に HTTP パラメーター、JSON 本文にセッションパラメーターを使用します。

成功した場合、応答本文は返されません。

以下も参照してください。:

- データスペースをロックまたはロック解除するための権限
- `LockSpec.lock`

URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:lock
```

説明

- `{dataspace}` は、`B` およびデータスペース識別子に対応します。

パラメーター

次のクエリパラメーターが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、ロックの取得を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、ロックの試行はすぐに実行されます。いくつかの理由により、待機時間は <code>60</code> 秒を超えることはできません。それ以外の場合、値は最大値で上書きされます。 <code>整数</code> 型。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> (<i>No content</i>)	リクエストは正常に処理されましたが、応答本文は返されません。
<code>400</code> (<i>Bad request</i>)	リクエストの URL または本文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>)	認証されたユーザーの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。

HTTP コード	説明
404 (Not found)	URL で指定されたリソースが見つかりません。
409 (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でロックが拒否されます。
500 (Internal error)	アプリケーションによる予期しないエラー。エラーの詳細は通常、EBX® ログに記録されます。

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。データスペースの状態により、次のことが適用されます。

- 同一ユーザーによるロックでは、ロックが解除されます。
- 別のユーザーによるロックでは、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、ロックが解除されます。
- 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、次のように設定された `Content-Type` ヘッダーを使用します。

- `application/x-www-form-urlencoded` : 本文に HTTP パラメーターを使用します。
- `application/json` : URL に HTTP パラメーター、JSON 本文にセッションパラメーターを使用します。

成功した場合、応答本文は返されません。

以下も参照してください。:

- データスペースをロックまたはロック解除するための権限
- `LockSpec.unlock`

URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:unlock
```

説明

- `{dataspace}` は、`B` およびデータスペース識別子に対応します。

パラメーター

次のクエリパラメーターが操作に適用されます。

パラメーター	説明
<code>forceByAdministrator</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、管理者は強制的にロック解除できます。 <code>ブール値</code> 型。デフォルト値は <code>false</code> です。

HTTP コード

HTTP コード	説明
<code>204</code> (<i>No content</i>)	リクエストは正常に処理されましたが、応答本文は返されません。
<code>400</code> (<i>Bad request</i>)	リクエストの URL または本文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>)	認証済みユーザーの指定されたリソースのロック解除権限が拒否されました。
<code>404</code> (<i>Not found</i>)	URL で指定されたリソースが見つかりません。
<code>409</code> (<i>Conflict</i>)	リソースはすでに別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っても <code>forceByAdministrator</code> パラメー

HTTP コード	説明
	ターが <code>false</code> の場合、ロック解除は拒否されます。
<code>500</code> (<i>Internal error</i>)	アプリケーションによる予期しないエラー。エラーの詳細は通常、EBX® ログに記録されます。

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、履歴やデータに対して削除を実行することができます。

この操作は `POST` メソッドを使用し、次のように設定された `Content-Type` ヘッダーを使用します。

- `application/x-www-form-urlencoded` : 本文に HTTP パラメーターを使用します。
- `application/json` : URL に HTTP パラメーター、JSON 本文にセッションパラメーターを使用します。

成功した場合、応答本文は返されません。

注意:

データサービスを介して実行されるマージでは、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、親のデータを自動的に上書きします。

以下も参照してください。:

- `ProcedureContext.doMergeToParent`
- データと履歴の削除

URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:merge
```

説明

- `{dataspace}` は、`B` およびデータスペース識別子に対応します。

パラメーター

次のクエリパラメーターが操作に適用されます。

パラメーター	説明
<code>deleteHistoryOnMerge</code>	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。 <code>ブール値</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。 <code>EBX®</code> メイン構成ファイル でプロパティ <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> を指定することにより、デフォルト値を再定義することができます。
<code>deleteDataOnMerge</code>	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。 <code>ブール値</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。 <code>EBX®</code> メイン構成ファイル でプロパティ <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> を指定してデフォルト値を再定義することができます。

HTTP コード

HTTP コード	説明
<code>204</code> (<i>No content</i>)	リクエストは正常に処理されましたが、応答本文は返されません。
<code>400</code> (<i>Bad request</i>)	リクエスト URL に無効なコンテンツが含まれています。

HTTP コード	説明
401 (<i>Unauthorized</i>)	認証に失敗しました。
403 (<i>Forbidden</i>)	認証済みユーザーの指定されたリソースのマージ権限が拒否されました。
404 (<i>Not found</i>)	URL で指定されたリソースが見つかりません。
500 (<i>Internal error</i>)	アプリケーションによる予期しないエラー。エラーの詳細は通常、EBX® ログに記録されます。

ベータ機能：データスペースまたはスナップショットを閉じる

指定されたデータスペースまたはスナップショットを閉じます。閉じた後、履歴および/またはデータに対して削除を実行することができます。

この操作は `POST` メソッドを使用し、次のように設定された `Content-Type` ヘッダーを使用します。

- `application/x-www-form-urlencoded` : 本文に HTTP パラメーターを使用します。
- `application/json` : URL に HTTP パラメーター、JSON 本文にセッションパラメーターを使用します。

成功した場合、応答本文は返されません。

以下も参照してください。:

- `Repository.closeHome`
- データと履歴の削除

URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/.../data/v1/{dataspace}:close
```

説明

- `{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くことに対応します。

パラメーター

次のクエリパラメーターが操作に適用されます。

パラメーター	説明
<code>deleteHistoryOnClose</code>	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。 <code>ブール値</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。 <code>EBX®</code> メイン構成ファイル でプロパティ <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> を指定することにより、デフォルト値を再定義することができます。
<code>deleteDataOnClose</code>	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、閉じるときに削除するかどうかを設定します。 <code>ブール値</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。 <code>EBX®</code> メイン構成ファイル でプロパティ <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> を指定してデフォルト値を再定義することができます。

HTTP コード

HTTP コード	説明
<code>204</code> (<i>No content</i>)	リクエストは正常に処理されましたが、応答本文は返されません。
<code>400</code> (<i>Bad request</i>)	リクエストの URL または本文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。

HTTP コード	説明
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の指定されたリソースを閉じるための権限が拒否されました。
404 (<i>Not found</i>)	URL で指定されたリソースが見つかりません。
422 (<i>Unprocessable entity</i>)	ブロッキング制約に違反しています。
500 (<i>Internal error</i>)	アプリケーションによる予期しないエラー。エラーの詳細は通常、EBX® ログに記録されます。

ヘルスオペレーション

概要

オペレーションの正常性は、EBX® サーバーの監視に役立ちます。

運用開始

EBX® が開始されて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 (Service unavailable) を返します。

URL フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/.../health/v1/started
```

HTTP コード

HTTP コード	説明
200 (<i>OK</i>)	リクエストは正常に処理され、応答本文が返されます。
503 (<i>Service unavailable</i>)	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。

応答本文

JSON の例を参照してください。

動作確認

EBX® が開始されて確認された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 (Service unavailable) を返します。

URL フォーマットは次のとおりです。

```
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、応答本文が返されます。
503 (Service unavailable)	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。

応答本文

JSON の例を参照してください。

運用停止

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 (Service unavailable) を返します。

URL フォーマットは次のとおりです。

```
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、応答本文が返されます。
503 (Service unavailable)	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。

応答本文

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、使用可能な REST ビルトインリソースとそれらに関連付けられた操作を構造化して説明することにより、開発と使用を容易にします。

`includeOpenApiDetails` クエリパラメーターを使用すると、データの Select 操作を通じて OpenAPI ドキュメントの HATEOAS リンクを利用できます。

REST OpenAPI サービスの権限は、[グローバル許可](#)でグローバルに定義されています。

特定の言語でクライアントを生成する場合は、データ型を正しくマップすることが重要です。[OpenAPI 形式](#)を使用して、入力を検証するか、選択したプログラミング言語の特定の型に値をマップします。詳細については、[単純型のコンテンツ](#)を参照してください。

注意:

特定のフォーマットをサポートしないツールは、デフォルトで `type` のみに戻る場合があります。

OpenAPI ドキュメントの生成

この操作では、`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 フォーマットは次のとおりです。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/{category}/v1/
{dataspace}/{dataset}/{pathInDataset}
```

説明

- `{category}` は、**data** または **form-data** の中の**操作カテゴリ**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くことに対応します。
- `{dataset}` はデータセット識別子に対応します。
- `{pathInDataset}` は、次のパスに対応します。
 - テーブルノード
 - データセットターミナルノードまたはそれ以上

注意:

生成されたドキュメントは、ユーザーの権限に依存しません。フィールド全体が表示されません。

HTTP コード

HTTP コード	説明
<code>200</code> (<i>OK</i>)	リクエストは正常に処理されました。
<code>401</code> (<i>Unauthorized</i>)	認証に失敗しました。
<code>403</code> (<i>Forbidden</i>)	指定されたリソースを読み取るための権限が拒否されました。
<code>404</code> (<i>Not found</i>)	URL で指定されたリソースが見つかりません。

ステージングオペレーション

Staging REST API は、ソースサーバーとターゲットサーバーと対話するように設計されています。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、次のリンクの OpenAPI を介して入手できます。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/staging/v1
```

開発実行モードでは、次のリンクから **Swagger UI** を利用できます。

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/api/v1/staging/v1/ui
```

以下も参照してください。:

- [ビジュアルドキュメント](#)

ドメイン管理操作

domains 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します。
- 一意の名前でドメインを選択します。
- 新しいドメインを作成します。
- ドメインのコンポーネントを選択します。
- コンポーネントをドメインに追加します。
- ドメインからコンポーネントを削除すると、コンポーネントが削除されます。

ステージングエレメントのナビゲーション操作

elements サービスは、インスタンスのエレメント全体を階層的に表示するように設計されています。これは主に、レベルを処理する階層ビューをサポートする UI インターフェイス用に設計されています。構造化された本文の代わりにクエリパラメーターを使用する目的は、リンクを使用してナビゲーションを容易にすることです。提供される機能は次のとおりです。

- エレメントのコンテンツを展開します。
- 選択したエレメントの下にある特定の種類のエレメントのコンテンツを展開します。

アーカイブ操作

archives サービスは、単一のリクエストで実行中のインスタンスとの間でステージングアーカイブをインポート/エクスポートするように設計されています。また、非同期インポート用のアーカイブのアップロードにも使用されます。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します。
- 実行中のインスタンスにアーカイブをインポートします。
- 非同期インポート用のアーカイブをアップロードします。

cURL 経由で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 経由で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

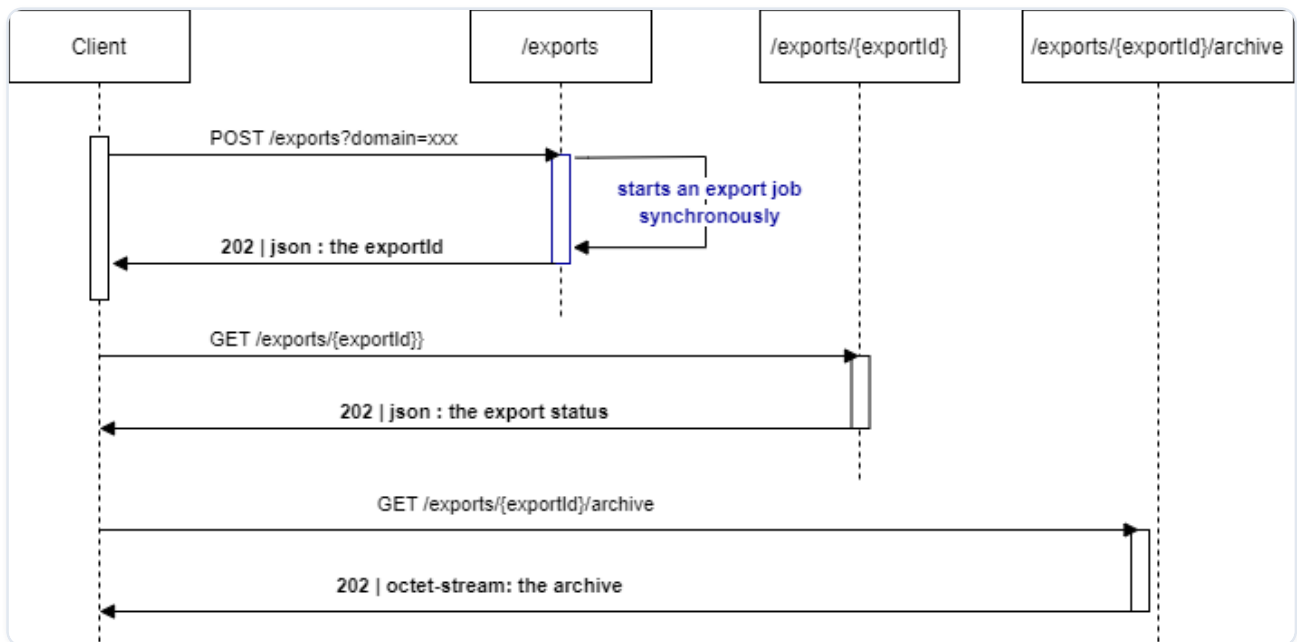
exports サービスは、高度なエクスポート機能进行处理するように設計されています。アーカイブはエクスポートされ、事前に定義された期間、サーバーに保存されます。ユーザーはアーカイブに関する情報を取得でき、有効期限が切れるまでダウンロードできます。提供される機能は次のとおりです。

- エクスポートを開始します。
- 開始されたエクスポートのステータスを取得します。
- エクスポートされたアーカイブをダウンロードします。

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機能します。

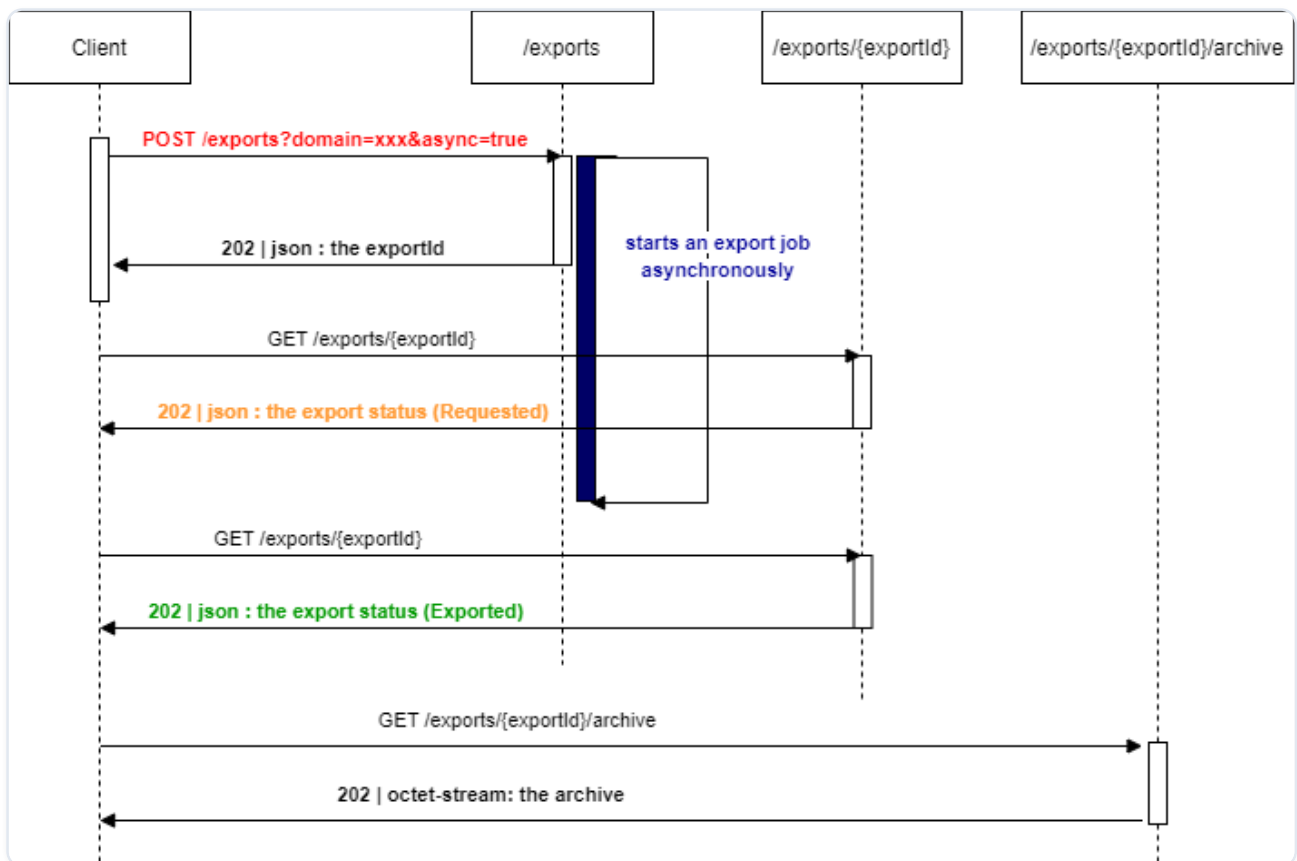
同期モード

サーバー上のアーカイブ全体をエクスポートした後に応答するブロッキングモード。アーカイブはすぐにダウンロードできます。



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。クライアントは、ステータスエンドポイントをポーリングして、エクスポートのステータスを取得する必要があります。ステータスが「エクスポート済み」になると、アーカイブはダウンロード可能になります。



一貫性を保つために、ステージング非同期ジョブは、60 分以内に完了しない場合は中止されます。

インポート操作

imports サービスは、高度なインポート機能进行处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、事前に定義された期間サーバーに保存されます。ユーザーは、インポートを非同期的に開始したり、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。

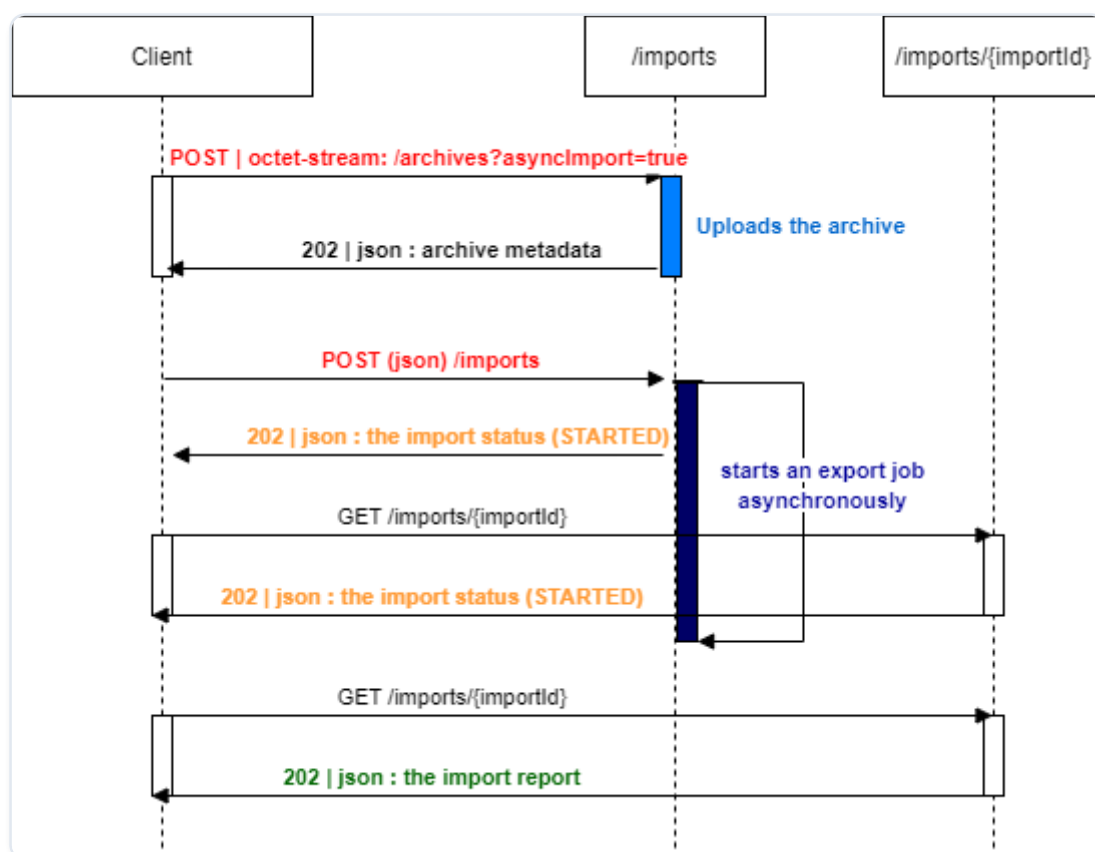
アーカイブは、オプション `asyncImport=true` を指定した `archives` サービスを使用してアップロードされます。

提供される機能は次のとおりです。

- アップロード済みのアーカイブを使用してインポートを開始します。
- 開始されたインポートのステータス/レポートを取得します。

古いインポートからアーカイブ名を取得して、異なるオプションで同じアーカイブの再インポートを開始できます。有効期限が切れるまで利用できます。

このエンドポイントは非同期モードでのみ機能します。



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。
- データセットノードおよびサブターミナルのノードに適用されるフィールド操作はサポートされません。

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。

コンパクトフォーマットの制限事項

- 継承、追跡情報、セッションパラメーター、およびプロシージャコンテキストは、コンパクトフォーマットで処理されません。代わりに `extended` を使用してください。
- `履歴カテゴリ` は、コンパクトフォーマットではサポートされません。

読み取り操作

- `セクター` 内では、ページネーションコンテキストは `nextPage` プロパティに制限されています。

- `sortByRelevancy` パラメーターが有効な場合、`sort`、`sortOnLabel`、`sortPriority`、および `viewPublication` で定義されたソート基準パラメーターは無視されます。
- `viewPublication` パラメータ内では、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、階層表示を使用して REST URL で参照することはできません。

詳細については、[システム情報操作](#)を参照してください。

- 作成の準備操作は、データセットノードをサポートしていません。
- データスペースの `Select` 操作では、ラベルで並べ替えられません。
- スナップショットの `Select` 操作では、スナップショットは作成日で並べ替えられません。
- スナップショットの `Select` 操作に初期スナップショットを含めることはできません。

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イスの制御ポリシー `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、リソースで `Select` 操作を呼び出します。

詳細については、[ブロッキングと非ブロッキングの制約](#)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

- ドキュメントの生成は、ユーザーインターフェイスからは利用できません。
- ドキュメント生成 REST サービスは YAML フォーマットをサポートしていません。
- 以下を除いて、[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。
 - データセットツリー、データセットノード、フィールドの `Select` 操作。
 - 1 回のリクエストで複数のレコードを挿入または削除します。
 - HTTP ヘッダー `content-type: x-www-form-urlencoded` を使用して、本文リクエストでクエリパラメーターを送信します。

- 「基本認証スキーム」は、説明されている唯一の方法です。

JSON 形式

- はじめに
- 拡張形式
- 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ブ化による、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。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 `content` プロパティが少なくとも含まれています。

• 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、ローカル列挙型の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、一部の技術データは `content` プロパティの横に追加できません。

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": "CM"  
    },  
    {  
      "content": "PI"  
    }  
  ]  
}
```

```

    {
        "content": "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。

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。:

- [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。

```

{
    "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"

    },

    "table2": {

        "details": "http://.../rootName/table2",

        "openApiDetails": "http://.../api/.../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](#)

• 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>description</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>	テーブルの内容を履歴化す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、そ	いいえ

JSON プロパティ	JSON 形式	説明	必須
		れ以外の場合は <code>false</code> です。 以下も参照してください。: 履歴	
<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。継承がアクティブになっている場合、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。: 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>description</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、そのデータを読み書きできるかどうかです。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。 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、制約チェックが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>SchemaFacetEnumeration</code> その値に従って、<code>selector</code> 要求パラメーターを使用して可能な値を取得するためにフィールドを使用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、フィールドをソート条件で利用できるかどうかを指定します。	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

(*) テーブル、レコード、およびレコードフィールドの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
<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>データセット</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	いいえ

JSON プロパティ	JSON 形式	説明	必須
		じデータセットに対応します。	
<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>データスペース</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。	
データセット	String	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
tablePath	String	ターゲットテーブルのパスを指定します。	はい
details	文字列	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。: includeDetails パラメーター	いいえ
fieldToSource	String	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 tableRefInverse または linkTable の場合に定義されます。	いいえ

JSON プロパティ	JSON 形式	説明	必須
<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>	文字列	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>SchemeLinkedField.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。 以下も参照してください。: <code>SchemaLinkedField.getLinkedField</code>	はい

JSON プロパティ	JSON 形式	説明	必須
<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
- `SchemeLinkedField`

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
localModify	Boolean	列挙依存フィールドが、同じレコード内のフィールドに対して実行された変更によって影響を受けるかどうかを指定します。	はい
データスペース	文字列	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
データセット	String	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
targetPath	String	列挙型依存関係のターゲットパスを指定します。	はい
details	String	列挙型依存関係のターゲット REST リソーステーブル	いいえ

JSON プロパティ	JSON 形式	説明	必須
		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	

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化) 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 `Array` 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。可能な値は、 <code>lasc</code> または <code>ldesc</code> です。詳細については、	はい

JSON プロパティ	JSON 形式	説明	必須
		<code>sortByRelevancy</code> 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、`validation` プロパティがルートノードに追加されます。

`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。 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
		<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 `Array` 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 <code>オブジェクト</code> などの単純なタイプにすることができます。	いいえ

JSON プロパティ	JSON 形式	説明	必須
<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、 ブロッキングと非ブロッキングの制約 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。
<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定します。

タイプ	説明
<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定します。
<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指定します。
<code>excludeSegment</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。
<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。

タイプ	説明
<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化) 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>関連付け</code> と <code>選択</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、 <code>details</code> プロパティで提供される	いいえ

JSON プロパティ	JSON 形式	説明	必須
		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操作で使用可能)。	<code>いいえ</code>

JSON プロパティ	JSON 形式	説明	必須
		• <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>includeHistory</code>	いいえ

JSON プロパティ	JSON 形式	説明	必須
<code>label</code>	文字列	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>	文字列	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>	String	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態が含まれます。両方の継承が使用され	いいえ

JSON プロパティ	JSON 形式	説明	必須
		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。: - 継承モード - 値検索メカニズム	
<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 <pre>http://.../ BReference/ dataset/root/ table/pk/ enumField? selector=true</pre>	いいえ
<code>validation</code>	<code>Array</code>	現在のノードコンテキストに関する検証レポートが含まれています。	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。: 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。:

- 単純型フォーマット

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
Group <code>xs:complexType</code>	<code>オブジェクト</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" }, "country": { "content": "France" } }</pre>	<code>グループ</code>

XML スキーマ	JSON 形式	例	メタタイプ
		<pre>} }</pre>	
List <code>maxOccurs > 1</code>	<code>Array</code> JSON <code>オブジェクト</code> 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、<code>ノード値</code>として表されます。	<code>xs:int</code> タイプ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> 複数オカレンスグループの例。	単純型のメタ または <code>グループ</code>

XML スキーマ	JSON 形式	例	メタタイプ
		<pre>[{ "content": { "road": { "content": "11 rue scribe" }, "zipcode": { "content": "75009" }, "country": { "content": "France" } } }, { "content": { "road": { "content": "711 Atlantic Ave" }, "zipcode": { "content": "MA 02111" } } }]</pre>	

XML スキーマ	JSON 形式	例	メタタイプ
		<pre> }, "country": { "content": "United States" } } }] </pre>	

セレクトター

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化)。

```

{

  "rows": [

    {

      "content": "F",

      "label": "feminine"

    },

    {

```

```
        "content": "M",  
  
        "label": "masculine"  
    }  
  
    ],  
  
    "pagination": {  
  
        "nextPage": null  
    }  
  
}
```

以下も参照してください。:

- [includeSelector](#)

コンパクト形式

- [はじめに](#)
- [グローバル構造](#)
 - [JSONリクエスト本文](#)
 - [JSON応答本文](#)
- [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。コンパクトフォー

マット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 `Object` 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、ローカル列挙型の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。

```
{  
  
  "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

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 `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。この要求は、フィールドの更新時に使用されます。

リクエストの構造は、[ノード値](#)で定義されているものと同じです。

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](#)

• 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](#)
- [Prepare操作](#)

• フィールド

コンパクトなjson形式は、関連付けノードと選択ノードをサポートしていません。

他の種類のノードの場合、それらには現在の**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](#)) *によって是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説明するKey-ValueJSONエントリに対応します。

以下も参照してください。:

- [単純型フォーマット](#)

グループ、リスト、列挙の内容

XMLスキーマ	JSON形式	例
Group <code>xs:complexType</code>	オブジェクト 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>	配列 JSON <code>Object</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、ノード値で構成されています。	<code>xs:int</code> タイプ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。 <pre>[{ "road": "11 rue scribe", "zipcode": "75009",</pre>

XMLスキーマ	JSON形式	例
		<pre>"country": "France" }, { "road": "711 Atlantic Ave", "zipcode": "MA 02111", "country": "United States" }]</pre>
Enumeration（列挙） <code>xs:string</code>	オブジェクト <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
 - 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リクエスト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要求が成功した場合、`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リクエスト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要求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要求が失敗した場合、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、	はい

JSON プロパティ	JSON 形式	説明	必須
		<code>onInsertUpdateOrDelete</code> 、 <code>onUserSubmit-checkModifiedValues</code> 、 <code>never</code> です。 詳細については、 ブロッキングと非ブロッキングの制約 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

この機能により、限られたパラメーター化可能な数のデータを返すことができます。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、およびデータスペースのタイプのデータ

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。このコンテキストにより、UI と同様に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](#)
- [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](#)

このコンテキストに関連する詳細情報は、以降で説明します。

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であり、応答では利用できないかどうかのみを定義します。

注意:

(**) セレクターには存在しません。

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" "特殊文字のエスケープには、円記号 (\) を前置します。" null	string name text	タイプ: 文字列 形式: n/a
osd:html	String (Unicode文字。 RFC4627 を参照)	" HTMLタグは問題なく記述できます "	html	タイプ: 文字列 形式: html
osd:email	String (Unicode文字。 RFC4627 を参照)	"employee@mycompany.com"	email	タイプ: 文字列 形式: メール
osd:locale	String (言語タグ。 RFC1766 を参照)	"en-US"	locale	タイプ: 文字列 形式: ロケール
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	タイプ: 文字列 形式: n/a

XML スキーマ	JSON 形式	例	メタタイプ	OPENAPI
<code>xs:boolean</code>	Boolean	true false null	boolean	タイプ: ブール値 形式: n/a
<code>xs:decimal</code>	Number または null	-10.5 20.001 15 -1e-13	10 進数	タイプ: 数値 形式: double
<code>xs:date</code>	"yyyy-MM-dd" 形式の 文字列	"2015-04-13"	date	タイプ: 文字列 形式: 日付
<code>xs:time</code>	文字列 の形式: • "HH:mm:ss" • "HH:mm:ss.SSS"	"11:55:00" "11:55:00.000"	time	タイプ: 文字列 形式: 日時
<code>xs:dateTime</code>	文字列 の形式: • "yyyy-MM-ddTHH:mm:ss" • "yyyy-MM-ddTHH:mm:ss.SSS"	"2015-04-13T11:55:00" "2015-04-13T11:55:00.000"	dateTime	タイプ: 文字列 形式: 日時
<code>xs:anyURI</code>	String (URI。RFC3986 を参照)	"https://fr.wikipedia.org/wiki/René_Descartes"	anyURI	タイプ: 文字列 形式: uri

XML スキーマ	JSON 形式	例	メタタイプ	OPENAPI
<code>xs:int</code> <code>xs:integer</code>	<code>Number</code> また は <code>null</code>	1596	<code>int</code>	タイプ: <code>整数</code> 形式: <code>int32</code>
<code>osd:resource</code>	<code>String</code> リソース形式 の値が含まれ ます。	"ebx-tutorial : ext-images : frontpages/ Bach.jpg"	<code>resource</code>	タイプ: <code>文字 列</code> 形式: <code>n/a</code>
<code>osd:color</code>	形式: <code>"#[A-Fa-f0-9]{6}"</code> の <code>文字列</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>	タイプ: <code>文字 列</code> 形式: <code>n/a</code>
<code>osd:datasetName</code>	形式: <code>"[a-zA-Z_][a-zA-Z0-9_]*"</code> 、最 大 64 文字の <code>文字列</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>データセット</code>	タイプ: <code>文字 列</code> 形式: <code>n/a</code>
<code>osd:dataspaceKey</code>	形式: <code>[BV][a-zA-Z0-9_:\-\\]+</code> 、最大 33 文字の <code>文字列</code> 。	"Bebx-tutorial"	<code>データスパー ス</code>	タイプ: <code>文字 列</code> 形式: <code>n/a</code>

XML スキーマ	JSON 形式	例	メタタイプ	OPENAPI
	データスペースのフォーマット設定 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 操作を呼び出すと、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。
- `externalKey` 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

JSON プロパティ	JSON 形式	説明	必須
<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
<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、RESTful データサービスの [Update 操作](#)と [Insert 操作](#)、および Java API の `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)

JSON ソースドキュメントの状態	動作
	• Update 操作の場合、フィールド値は変更されません。 • Insert 操作の場合、動作は byDelta モードが無効になっている場合と同じです。 byDelta 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。 • 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 null 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 null 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 null 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。 • 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:

JSON ソースドキュメントの状態	動作
	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

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。:

- トークン操作の作成

• 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。:

- 。トークン操作の作成

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
details	String	ビューアクセス URL に対応します。	はい

JSON プロパティ	JSON 形式	説明	必須
<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。 • <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。	はい

```
{
  "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

JSON プロパティ	JSON 形式	説明	必須
<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
<code>isSelectAllowed</code>	<code>Boolean</code>	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>	<code>Boolean</code>	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>	<code>String</code>	データスペースの子 REST リソース URL を指定しま	いいえ

JSON プロパティ	JSON 形式	説明	必須
		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。	
<code>snapshots</code>	<code>String</code>	データスペースのスナップショット REST リソース 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。	はい

```
{
  "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。	はい
<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キュメントに対応します。	いいえ
<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	いいえ

JSON プロパティ	JSON 形式	説明	必須
		ストされています)。	
<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)。	いいえ
<code>locked</code>	<code>Boolean</code>	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>	<code>Boolean</code>	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>paranet</code>	<code>String</code>	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>	<code>Boolean</code>	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>	<code>String</code>	所有者プロファイルを指定します。	いいえ

JSON プロパティ	JSON 形式	説明	必須
<code>creator</code>	<code>String</code>	作成者プロファイル を指定します。	はい
<code>creationDate</code>	<code>Date</code>	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>。	いいえ

JSON プロパティ	JSON 形式	説明	必須
		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。	
<code>owner</code>	<code>String</code>	所有者プロフィールを指定します。 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キュメントに対応します。 デフォルト値は <code>null</code> です。	いいえ
<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

JSON プロパティ	JSON 形式	説明	必須
<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>dataspaceKeyToCopyPermissionsFrom</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: スナップショットの作成には適用されません。	いいえ

```
{
  "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

正常性および応答の確認

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 (複数值) 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修飾子を除く) 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。:

- データモデルアシスタント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>	<code>VARCHAR</code>	<code>java.lang.String</code>	
<code>xs:boolean</code>	ブール値	<code>java.lang.Boolean</code>	値：TRUE、FALSE、UNKNOWN
<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>	<code>VARCHAR</code>	<code>java.net.URI</code>	
<code>xs:Name</code> (<code>xs:string</code> restriction)	<code>VARCHAR</code>	<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)	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>osd:html</code> (<code>xs:string</code> restriction)	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>osd:email</code> (<code>xs:string</code> restriction)	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>osd:password</code> (<code>xs:string</code> restriction)	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>osd:color</code> (<code>xs:string</code> restriction)	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>osd:resource</code> (<code>xs:anyURI</code> restriction)	<code>VARCHAR</code>	内部クラス
<code>osd:locale</code> (<code>xs:string</code> restriction)	<code>VARCHAR</code>	<code>java.util.Locale</code>
<code>osd:dataspaceKey</code> (<code>xs:string</code> restriction)	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>osd:datasetName</code> (<code>xs:string</code> restriction)	<code>VARCHAR</code>	<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
<code>creator</code>	レコードの作成者。	<code>VARCHAR</code>	<code>java.lang.String</code>	<pre>SELECT t."ebx-metadata"."system"."creator" FROM myTable t SELECT t.* FROM myTable t WHERE t."ebx-metadata"."system"."creator" = 'User1' SELECT t.* FROM myTable t ORDER BY t."ebx-metadata"."system"."creator"</pre>
<code>creation_time</code>	レコードの作成時刻。	タイムスタンプ	<code>java.util.Date</code>	<pre>SELECT t."ebx-metadata"."system"."creation_time" FROM myTable t SELECT t.* FROM myTable t WHERE t."ebx-metadata"."system"."creation_time"</pre>

名前	説明	SQL タイプ	JAVA クラス	例
				<pre>> 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 myTable t WHERE t."ebx- metadata"."system"."updat = 'User1' SELECT t.* FROM myTable t ORDER BY t."ebx- metadata"."system"."updat</pre>

名前	説明	SQL タイプ	JAVA クラス	例
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_time" FROM myTable t WHERE t."ebx-metadata"."system"."update_time" > TIMESTAMP '2001-06-10 14:57:20' SELECT t.* FROM myTable t ORDER BY t."ebx-metadata"."system"."update_time" </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。 Adaptation。	内部型	com.onwbp.adaptation	<pre> SELECT t."\$adaptation" FROM myTable t WHERE t.value>100 </pre>
\$pk	レコードの主キーの文字列表現。	VARCHAR	java.lang.String	<pre> SELECT t.* FROM myTable1 t </pre>

名前	説明	SQL タイプ	JAVA クラス	例
	以下も参照してください。: • <code>PrimaryKey.syntax</code>			<pre>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"</pre>

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順位を、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` instead of `SELECT MAX(val) FROM myTable`

`SELECT val FROM myTable ORDER BY val LIMIT 1` instead of `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
<code>value1 = value2</code>	等しい <code>SELECT 1 = 0 : false</code> <code>SELECT 4 = 4 : true</code>
<code>value1 <> value2</code>	等しくない <code>SELECT 1 <> 0 : true</code> <code>SELECT 4 <> 4 : false</code>
<code>value1 > value2</code>	より大きい <code>SELECT 1 > 0 : true</code> <code>SELECT 4 > 4 : false</code>

演算子の構文	説明と例
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>
value1 BETWEEN value2 AND value3	value1 が value2 以上、かつ value3 以下かどうか <code>SELECT 4 BETWEEN 3 AND 10 : true</code> <code>SELECT 1 BETWEEN 3 AND 10 : false</code>
value1 NOT BETWEEN value2 AND value3	value1 が value2 以上、かつ value3 以下かどうか <code>SELECT 4 NOT BETWEEN 3 AND 10 : false</code> <code>SELECT 1 NOT BETWEEN 3 AND 10 : true</code>

演算子の構文	説明と例
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。 <pre>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</pre>
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</pre>

演算子の構文	説明と例
	<code>SELECT 'abcd%' NOT LIKE '%!_' ESCAPE '!' : true</code>
value IN (value *)	値がリスト内の値と等しいかどうか <code>SELECT firstname FROM employee WHERE name IN ('Smith', 'Hamilton') : John, Maria</code>
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 <code>SELECT firstname FROM employee WHERE name NOT IN ('Smith', 'Hamilton') : Jennifer</code>
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 <code>SELECT e.firstname FROM employee e WHERE e.department.id IN (SELECT d.id FROM department d WHERE d.name='IT') : John</code>

算術演算子と関数

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
+数値	数値を返します。 <code>SELECT +4 : 4</code>
-数値	数値を返します。 <code>SELECT -4 : -4</code>
数値1+数値2	数値1と数値2を返します。 <code>SELECT 18 + 4 : 22</code>
数値1-数値2	数値1から数値2を引いたものを返します。 <code>SELECT 18 - 4 : 14</code>
数値1 *数値2	数値1に数値2を掛けた値を返します。 <code>SELECT 18 * 4 : 72</code>

演算子の構文	説明と例
数値1 /数値2	数値1を数値2で割った値を返します。 <code>SELECT 18 / 4 : 4</code>
ABS (数値)	数値の絶対値を返します。 <code>SELECT ABS(-243.5) : 243.5</code>
ACOS (数値)	数値のアーコサインを返します。 <code>SELECT ACOS(-1) : 3.141592653589793</code>
ASIN (数値)	数値のアークサインを返します。 <code>SELECT ASIN(0.25) : 0.25</code>
ATAN (数値)	数値のアークタングェントを返します。 <code>SELECT ATAN(2.5) : 1.1902899496825317</code>
ATAN2 (数値、数値)	数値座標のアークタングェントを返します。 <code>SELECT ATAN2(0.50, 1) : 0.4636476090008061</code>
CEIL (数値)	数値を切り上げ、数値以上の最小の数値を返します。 <code>SELECT CEIL(25.75) : 26</code>
CEILING (数値)	数値を切り上げ、数値以上の最小の数値を返します。 <code>SELECT CEILING(25.75) : 26</code>
COS (数値)	数値の余弦を返します。 <code>SELECT COS(2) : -0.4161468365471424</code>
COT (数値)	数値の余接を返します。 <code>SELECT COT(6) : -3.436353004180128</code>
DEGREES (数値)	数値をラジアンから度に変換します。 <code>SELECT DEGREES(1.5) : 85.94366926962348</code>
EXP (数値)	eを数値の累乗で返します。 <code>SELECT EXP(1) : 2.718281828459045</code>
FLOOR (数値)	数値を切り捨て、数値以下の最大数を返します。 <code>SELECT FLOOR(25.75) : 25</code>

演算子の構文	説明と例
LN (数値)	数値の自然対数 (底e) を返します。 <code>SELECT LN(2) :</code> 0.6931471805599453
LOG10 (数値)	数値の10を底とする対数を返します。 <code>SELECT LOG10(2) :</code> 0.3010299956639812
MOD (数値1、数値2)	nude1をnumeric2で割った余り (モジュラス) を返します。数値1が負の場合にのみ、結果は負になります。 <code>SELECT MOD(18, 4) :</code> 2
POWER (数値1、数値)	nude1をnumeric2の累乗で返します。 <code>SELECT POWER(4, 2) :</code> 16.0
RADIANS (数値)	数値を度からラジアンに変換します。 <code>SELECT RADIANS(180) :</code> 3.141592653589793
RAND ()	指定されている場合、シード値として数値を使用してランダムなdoubleを返します。 <code>SELECT RAND(6) :</code> 0.9891840064573959
RAND_INTEGER (数値)	0から数値-1までのランダムな整数を生成し、指定されている場合はシードを使用して乱数ジェネレーターを初期化します。 <code>SELECT RAND_INTEGER(100, 5) :</code> 0 <code>SELECT RAND_INTEGER(6, 100) :</code> 11
ROUND (numeric1、numeric2)	数値1を数値2に丸め、小数点のすぐ近くに配置します。 <code>SELECT ROUND(135.375, 2) :</code> 135.38
SIGN (数値)	数値の符号を返します。 <code>SELECT SIGN(255.5) :</code> 1
SIN (数値)	数値の正弦を返します。 <code>SELECT SIN(2) :</code> 0.9092974268256817
SQRT (数値)	数値の平方根を返します。 <code>SELECT SQRT(64) :</code> 8.0

演算子の構文	説明と例
TAN (数値)	数値の接線を返します。 <code>SELECT TAN(1.75) : -5.52037992250933</code>
TRUNCATE (numeric1)	数値1を0に切り捨てます (numeric2が指定されていない場合)。小数点のすぐ近くに配置します。 <code>SELECT TRUNCATE(135.375) : 135</code> <code>SELECT TRUNCATE(135.375, 2) : 135.37</code>

論理演算子

次の表に、EBX®でサポートされているすべての SQL 論理演算子、およびそれらの標準 SQL 構文を示します。

演算子の構文	説明と例
boolean1 OR boolean2	boolean1がTRUEであるかboolean2がTRUEであるか。 <code>SELECT TRUE OR TRUE : true</code> <code>SELECT TRUE OR FALSE : true</code> <code>SELECT FALSE OR TRUE : true</code> <code>SELECT FALSE OR FALSE : false</code>
boolean1 AND boolean2	boolean1がTRUEでboolean2がTRUEかどうか。 <code>SELECT TRUE AND TRUE : true</code> <code>SELECT TRUE AND FALSE : false</code> <code>SELECT FALSE AND TRUE : false</code> <code>SELECT FALSE AND FALSE : 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、UNKNOWNを返します。 <code>SELECT NOT TRUE : false</code> <code>SELECT NOT FALSE : true</code> <code>SELECT NOT UNKNOWN : null</code>
boolean IS FALSE	ブール値がFALSEかどうか。ブール値が不明な場合はFALSEを返します。 <code>SELECT TRUE IS FALSE : false</code> <code>SELECT FALSE IS FALSE : true</code>
boolean IS NOT FALSE	ブール値がFALSEでないかどうか。ブール値が不明な場合はTRUEを返します。 <code>SELECT</code>

演算子の構文	説明と例
	TRUE IS NOT FALSE : true SELECT FALSE IS NOT FALSE : false
boolean IS TRUE	ブール値がTRUEかどうか。ブール値が不明な場合はTRUEを返します。 SELECT TRUE IS TRUE : true SELECT FALSE IS TRUE : false
boolean IS NOT TRUE	ブール値がTRUEでないかどうか。ブール値が不明な場合はFALSEを返します。 SELECT TRUE IS NOT TRUE : false SELECT FALSE IS NOT TRUE : true
boolean IS UNKNOWN	ブール値が不明かどうか。 SELECT TRUE IS UNKNOWN : false SELECT FALSE IS UNKNOWN : false
boolean IS NOT UNKNOWN	ブール値が不明ではないかどうか。 SELECT TRUE IS NOT UNKNOWN : true SELECT FALSE IS NOT UNKNOWN : true

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)	文字列の文字数を返します。 SELECT CHAR_LENGTH('Alfreds Futterkiste') : 19
CHARACTER_LENGTH(string)	CHARACTER_LENGTH(string) として SELECT CHARACTER_LENGTH('Alfreds Futterkiste') : 19

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)	指定されている場合、指定されたポイントから開始して、string2でstring1が最初に出現する位置を返します。 <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サブストリングを返します。 <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TIMESTAMPWITH 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TIMESTAMPWITH 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。
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</pre> <pre>SELECT YEAR(DATE '1968-07-20') : 1968</pre> <pre>SELECT YEAR(hiringDate) FROM employee WHERE name='Smith' : 2015</pre> <pre>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</pre> <pre>SELECT QUARTER(hiringDate) FROM employee WHERE name='Smith' : 4</pre> <pre>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</pre> <pre>SELECT MONTH(hiringDate) FROM employee WHERE name='Smith' : 10</pre> <pre>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</pre> <pre>SELECT WEEK(hiringDate) FROM employee WHERE name='Smith' : 42</pre> <pre>SELECT</pre>

演算子の構文	説明と例
	<pre>WEEK(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 26</pre>
DAYOFYEAR(date)	日時値式から曜日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>
DAYOFMONTH(date)	日時値式から日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(TIME '10:34:21') : 10 SELECT HOUR(workStart) FROM employee WHERE name='Smith' : 8 SELECT HOUR(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 16</pre>

演算子の構文	説明と例
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</pre> <pre>SELECT MINUTE(TIME '10:35:21') : 35</pre> <pre>SELECT MINUTE(workStart) FROM employee WHERE name='Smith' : 0</pre> <pre>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 HOUR(TIMESTAMP '1969-07-20 09:34:21') : 9</pre> <pre>SELECT SECOND(TIME '10:34:22') : 22</pre> <pre>SELECT SECOND(workStart) FROM employee WHERE name='Smith' : 0</pre> <pre>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</pre> <pre>SELECT FK_AS_STRING(e.department) FROM</pre>

演算子の構文	説明と例
	<pre>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 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

- パッケージングと登録
- 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、JAX-RS 2.1 ([JSON-370](#)) および JSON-B ([JSON-367](#)) 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。要求入力、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。:

- JAX-RS: Java™ API for RESTful Web Services 2.1

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` アノテーションと 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® **ドキュメント** 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 javax.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 javax.servlet.http.*;

import javax.ws.rs.*;

import javax.ws.rs.container.*;

import javax.ws.rs.core.*;

import com.orchestranetworks.rest.annotation.*;

import com.orchestranetworks.rest.inject.*;
```

```

/**
 * The REST Toolkit Track service v1.
 */

@Produces(MediaType.APPLICATION_JSON)

@Consumes(MediaType.APPLICATION_JSON)

@Path("/track/v1")

@Documentation("Track service")

public final class TrackService
{

    @Context

    private ResourceInfo resourceInfo;


    @Context

    private SessionContext sessionContext;


    private static final Map<Integer, TrackDTO> TRACKS = new
    ConcurrentHashMap<>();


    /**
     * Gets service description
     */

    @GET

```

```

@Path("/description")

@Documentation("Gets service description")

@Produces({ MediaType.TEXT_PLAIN, MediaType.APPLICATION_JSON })

@AnonymousAccessEnabled

public String handleServiceDescription()

{

    return
this.resourceInfo.getResourceMethod().getAnnotation(Documentation.class).value();

}

/**

 * Selects tracks.

 */

@GET

@Path("/tracks")

@Documentation("Selects tracks")

public Collection<TrackDTO> handleSelectTracks(

    @QueryParam("singerFilter") final String singerFilter, // a
URL parameter holding a Java regular expression

    @QueryParam("titleFilter") final String titleFilter) // a URL
parameter holding a Java regular expression

{

    final Pattern singerPattern =

```

```

TrackService.compilePattern(singerFilter);

    final Pattern titlePattern =
TrackService.compilePattern(titleFilter);

    return TRACKS.values()

        .stream()

        .filter(Objects::nonNull)

        .filter(track -> singerPattern == null ||
singerPattern.matcher(track.singer).matches())

        .filter(track -> titlePattern == null ||
titlePattern.matcher(track.title).matches())

        .collect(Collectors.toList());

}

private static Pattern compilePattern(final String aPattern)

{

    if (aPattern == null || aPattern.isEmpty())

        return null;

    try

    {

        return Pattern.compile(aPattern);

    }

    catch (final PatternSyntaxException ignore)

```



```

        {

            // ignore invalid pattern

            return null;

        }

    }

    /**

    * Counts all tracks.

    */

    @GET

    @Path("/tracks:count")

    @Documentation("Counts all tracks")

    public int handleCountTracks()

    {

        return TRACKS.size();

    }

    /**

    * Selects a track by id.

    */

    @GET

    @Path("/tracks/{id}")

```

```

    @Documentation("Selects a track by id")

    public TrackDTO handleSelectTrackById(@PathParam("id") Integer id)

    {

        final TrackDTO track = TRACKS.get(id);

        if (track == null)

            throw new NotFoundException("Track id [" + id + "] does not
found.");

        return track;

    }

    /**

    * Deletes a track by id.

    */

    @DELETE

    @Path("/tracks/{id}")

    @Documentation("Deletes a track by id")

    public void handleDeleteTrackById(@PathParam("id") Integer id)

    {

        if (!TRACKS.containsKey(id))

            throw new NotFoundException("Track id [" + id + "] does not
found.");

        TRACKS.remove(id);

    }

```

```

/**

 * Inserts or updates one or several tracks.

 * <p>

 * The complex response structure corresponds to one of:

 * <ul>

 * <li>An empty content with the <code>location<code> HTTP header
defined

 * to the access URI.</li>

 * <li>A JSON array of {@link ResultDetailsDTO} objects.</li>

 * </ul>

 */

@POST

@Path("/tracks")

@Documentation("Inserts or updates one or several tracks")

public Response handleInsertOrUpdateTracks(List<TrackDTO> tracks)

{

    int inserted = 0;

    int updated = 0;

    final ResultDetailsDTO[] resultDetails = new
ResultDetailsDTO[tracks.size()];

    int resultIndex = 0;

```

```

final URI base = this.sessionContext.getURIInfoUtility()

    .createBuilderForRESTApplication()

    .path(this.getClass())

    .segment("tracks")

    .build();

for (final TrackDTO track : tracks)
{
    final String id = String.valueOf(track.id);

    final URI uri = UriBuilder.fromUri(base).segment(id).build();

    final int code;

    if (TRACKS.containsKey(track.id))
    {
        code = HttpServletResponse.SC_NO_CONTENT;

        updated++;
    }
    else
    {
        code = HttpServletResponse.SC_CREATED;

        inserted++;
    }
}

```

```

    }

    TRACKS.put(track.id, track);

    resultDetails[resultIndex++] = ResultDetailsDTO.create(

        code,

        null,

        String.valueOf(track.id),

        uri);
}

if (inserted == 1 && updated == 0)

    return Response.created(resultDetails[0].details).build();

return Response.ok().entity(resultDetails).build();
}

/**
 * Updates one track.
 */

@PUT

@Path("/tracks/{id}")

```

```

    @Documentation("Update one track")

    public void handleUpdateOneTrack(@PathParam("id") Integer id, TrackDTO
aTrack)

    {

        final TrackDTO track = TRACKS.get(id);

        if (track == null)

            throw new NotFoundException("Track id [" + id + "] does not
found.");

        if (aTrack.id != null && !aTrack.id.equals(track.id))

            throw new BadRequestException("Selected track id [" + id

                + "] is not equals to body track id.");

        TRACKS.put(aTrack.id, aTrack);

    }

}

```

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 `アダプテーション` 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); ;

```

```

aSingerDTO.content.asContentHolderForInput().copyTo(createContext);

        aContext.doCreateOccurrence(createContext,
singersTable);

    });

    if (procedureResult.hasFailed())

        throw new UnprocessableEntityException(

            procedureResult.getException().getLocalizedMessage());

    }

    /**

    * updates one singer.

    */

    @PUT

    @Path("/singers/{id}")

    @Documentation("updates one singer")

    public void handleUpdateOneSinger(@PathParam("id") final Integer id,
final SingerDTO aSingerDTO)

    {

        final ProgrammaticService svc = ... ;

        final AdaptationTable singersTable = ... ;

        final ProcedureResult procedureResult = svc.execute(

            (aContext) -> {

```

```

        // find the singer adaptation by id

        final Adaptation singerRecord = ... ;

        if (singerRecord == null)

            throw new NotFoundException("Singer with the id
["+ id + "] has not been found.");

        final ValueContextForUpdate createContext =
aContext.getContext(singerRecord.getAdaptationName()); ;

aSingerDTO.content.asContentHolderForInput().copyTo(createContext);

        aContext.doModifyContent(singerRecord, createContext);

    });

    if (procedureResult.hasFailed()){

        final Exception ex = procedureResult.getException();

        final Throwable cause = ex.getCause();

        if(cause instanceof NotFoundException)

            throw (NotFoundException) cause;

        throw new
UnprocessableEntityException(ex.getLocalizedMessage());

    }

}

```

```
}
```

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 (JSON-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 javax.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。

ビルトイン用の 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 要求からのデータ。
- 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、以前の構成全体を上書きします (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 要求からのデータ。
- **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 要求からのデータ。
- 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 `javax.ws.rs.WebApplicationException` を拡張する Java 例外をスローすることにより、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 (処理不能エンティティ) 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。

```
{
  "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"
    }
  ]
}
```

```
}
```

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。

パッケージングと登録

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、[Java EE 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` メソッドを上書き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 javax.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.RestApplication
```

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使用が容易になります。

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。

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。

EBX® スクリプティング

- [レコードの権限](#)
- [関数フィールド](#)
- [関数フィールド 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 core.date](#)
 - [ユニット core.datetime](#)
 - [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.resource](#)
 - [ユニット core.string](#)
 - [ユニット core.time](#)

- ユニット core.uri

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 thenelse」 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
 - はじめに
 - データスペース
 - データセット
 - セッション
- 関数
 - ロール
 - 文字列照合関数
 - 集計関数
 - 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。

[Data Model Assistant \(DMA\)](#) 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：** if**、** then**、** else**、** begin**、** end**、** return**、** null**、** and**、** or**、** not**、** true**、** false**。
- 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。

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
ブール値	xs:boolean
10 進数	xs:decimal xs:int xs:integer

タイプ	EBX® 対応型
文字列	xs:string xs:anyURI xs:Name osd:text osd:html osd:email osd:password osd:color osd:resource osd:locale osd:dataspaceKey osd:datasetName
タイムスタンプ	xs:dateTime
日付	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'
Noël	'No\u00EB'
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
整数	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、他のエイリアスが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可能性があります。

```
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;
```

注意

現在、次のことはできません。

- の間に集約関数を使用する。
- 値を集計する関連付けのフィールドを選択する。

演算子

デフォルトでは、操作の評価順序は優先順位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、優先順位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。

優先順位	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) . (フィールドへのアクセス) () (括弧)	リストインデックスは 10 進数にする必要があります。	任意の型にすることができます。	左から右へ
7	not	ブール値	ブール値	

優先順位	演算子	オペランド型	結果型	結合性
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

AND	TRUE	FALSE	NULL
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

```
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

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	文字列	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	文字列	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

名前	タイプ	説明
		プショット用かをさらに指定します。
isSnapshot	ブール値	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	文字列	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
userId	文字列	現在のユーザーの ID
userEmail	文字列	現在のユーザーのメール

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;
```

関数

ロール

次の表は、** isMember()** 関数についての説明です。

構文	説明
isMember('rolea', 'roleb'...)	現在のユーザーが指定されたロールの少なくとも 1 つを持っている場合、true を返します。ビルトインロール administrator、** readOnly**、** everyone** は、引用符なしで指定する必要があります。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、isMember(administrator) が is_member('administrator') と同じ結果を返さない可能性があることを意味します。同じロールが他のビルトインロールにも適用されます。

例

```

if isMember('SALES', 'SUPPORT') then

    return readWrite;

// Role administrator and not surrounded by quotes:

if isMember(administrator, 'SUPPORT') then

    return readOnly;

```

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;
```

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
isNull(value)	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;
```

関数フィールド

- 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
 - 単純型
 - 複合型
 - リスト型
 - 可変型と不変型
 - キーワード型
- リテラル
 - 文字列リテラル
 - 10 進数または整数リテラル
 - 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
 - 優先順位
 - 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
- 範囲
- 複合型変数
- 複数値フィールド
- 事前定義された変数
 - はじめに
 - レコード
 - ルート
 - データスペース
 - データセット
- 関数とプロシージャ
 - 関数
 - エクスポートされた関数
 - 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
- ループ
 - 「For in do」 ループ
 - 「While do」 ループ
- ユニット
- ログ取得とデバッグ
- 初期スクリプト

はじめに

関数 DSL (ドメイン固有言語) を使用して、関数フィールドを定義します。

この DSL は、手続き型で静的に型付けされた言語です。

関数フィールドを作成および変更するには、[Data Model Assistant \(DMA\)](#) 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。

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>


<function or procedure definition 1>

<function or procedure definition 2>

...

<function or procedure definition M>
```

詳細については、[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。

例

```
// This field returns the full address for current record.

export function getValue(): string
```



```
begin

  var address := record.FirstName | ' ' | record.LastName;

  for street in record.OfficeAddress.Street do

    begin

      address |= '\n' | street;

    end;

  end;

  address |= '\n' | record.OfficeAddress.ZipCode | ' ' |
record.OfficeAddress.City;

  address |= '\n' | record.OfficeAddress.Country;

  return address;

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 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 record.isActive then

    return false;
```

キーワード

予約済みのキーワードは、 **and**、 **** or****、 **** not****、 **** uses****、 **** as****、 **** export****、 **** typeof****、 **** mutable****、 **** immutable****、 **** unmodifiable****、 **** function****、 **** procedure****、 **** const****、 **** var****、 **** if****、 **** then****、 **** else****、 **** for****、 **** while****、 **** in****、 **do**、 **** begin****、 **** end****、 **** return****、 **** true****、 **** false****、 **** 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。

型

単純型

次の単純な型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
ブール値	boolean		xs:boolean
10 進数 (無制限の精度)	decimal		xs:decimal
整数 (32ビット)	int		xs:int xs:integer
文字列	string		xs:string xs:Name osd:text osd:html osd:email osd:password 1 osd:color osd:dataspaceKey osd:datasetName
タイムスタンプ (ミリ秒の精度でタイムゾーンなし)	timestamp	年月 (1 から 12) 日 (1 から 31) 時間 (0 から 23) 分 (0 から 59) 秒 (0.000 から 59.999)	xs:dateTime
日付 (タイムゾーンなし)	date	年月 (1 から 12) 日 (1 から 31)	xs:date
時間 (ミリ秒の精度)	time	時間 (0 から 23) 分 (0 から 59) 秒 (0.000 から 59.999)	xs:time

タイプ	キーワード	プロパティ	EBX® 対応型
ロケール	locale		osd:locale
URI (Uniform Resource Identifier)	uri		xs:anyURI
リソース	resource		osd:resource

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。

スキーマノードの型への参照 (たとえば、関数パラメーターまたは変数を宣言するとき) は、[keyword typeof](#) の使用を意味します。

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>
```

次の構文を使用して、変更不可能なリストを宣言します。

```
unmodifiable list<item_type>
```

リストが変更できない場合、項目を追加、削除、または置き換えることはできません。リストの項目は、その型に応じて、変更できる場合とできない場合があります。

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、10 進型の任意の式にすることができます。小数部分を削除して、インデックス値を整数に変換します。

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 ([keyword typeof](#) の章を参照)。

可変型と不変型

不変の複合型またはリストオブジェクトを変更することはできません。

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。

単純型は常に不変です。

変数または互換性のある不変型の戻り値に可変値を割り当てることができますが、不変値を可変変数または戻り値に割り当てると、コンパイル時にエラーが発生します。

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。このキーワードは、キーワード **mutable** または **immutable** の後に使用できます。

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
```

デフォルトでは、レコードフィールドの型は常に不変です。次の構文を使用して、可変型を指定できます。

```
mutable 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[*]
```

可変形式の型では、キーワード「mutable」を使用できます。

```
mutable typeof record.Addresses[*]
```

型に変更可能な形式がない場合、キーワードは単に無視されます。これは単純な型の場合です。

===

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'

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 進数または整数のリテラル

次の 10 進形式がサポートされています。

形式	例
整数	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。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

優先順位

デフォルトでは、操作の評価順序は優先順位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、優先順位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。

優先順位	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 10 進数にする必要があります。	任意の型にすることができます。	左から右へ
8	not	ブール値	ブール値	
7	* /	10 進数	10 進数	左から右へ
6	+ -	10 進数	10 進数	左から右へ
5		(文字列の連結)	文字列	文字列
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。

優先順位	演算子	オペランド型	結果型	結合性
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	または	ブール値	ブール値	左から右へ

算術演算子

すべての算術演算子 (*、/、+ および -) は、34 桁の精度 (IEEE 754R Decimal128 形式)。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、10 進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

ブール演算子はスレッド値ロジックを使用します。

AND 演算子の真偽 (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 演算子の真偽 (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

比較演算子

比較演算子 (< , <= , > , >= , = and <>) は、同じ型の 2 つのオペランドを比較します。int 型の変数は、比較前に常に **10 進数** に変換されます。

コンパレータの戻り値は **boolean** 型です。すべてのオペランドが **null** でない場合のみ、この値は **true** または **false** になります。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機能

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
:=	あらゆる型に使用できます。	標準割り当て。
	=	文字列
+=	10 進数	a += b は a := a + b と同等
-=	10 進数	a -= b は a := a - b と同等
*=	10 進数	a *= b は a := a * b と同等
/=	10 進数	a /= b は a := a / b 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

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

型検出による宣言

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 := 30;                     // Variable of type decimal.

var address := record.address; // Variable of an immutable complex.


const MAX_AGE := 100;              // Decimal constant.

const DEFAULT_COUNTRY := 'France'; // String constant.

end

```

注意

定数は必ずしも不変ではありません。

明示的な型定義

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 : decimal;

    age := 30;

    var address : typeof record.address;

    address := record.address;

    const MAX_AGE : decimal := 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

        do_something(); // 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 age : decimal := 'test';

end
```

範囲

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 := record.OfficeAddress;

// Declaration using typeof notation.

var address2 : typeof 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 := 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 := 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 10 進式の場合があります。式の整数部分のみが使用されます。

事前定義された変数

はじめに

事前定義された変数により、コンテキストへのアクセスが可能になります。これらのコンテキスト変数は定数であり、不変の複合型です。

レコード

事前定義された変数 **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。現在のレコードへの読み取り専用アクセスを許可します。

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

```
// Returns the full name.  
  
export function getValue(): string
```

```
begin

    return record.FirstName | ' ' | record.LastName;

end
```

ルート

事前定義された変数 **root** は、現在のスクリプトがデータセットの関数インスタンスフィールド用である場合にのみ使用できます。現在のデータセットのインスタンス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

```
// Returns information on current dataset data.

export function getValue(): string

begin

    return root.City | ' ' | root.Region;

end
```

データスペース

事前定義された変数 **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	文字列	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

名前	タイプ	説明
		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	文字列	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ショット用かをさらに指定します。
isSnapshot	ブール値	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 dataspace.isSnapshot then

    return 'Snapshot: ' | dataspace.name;

  else

    return 'Branch: ' | dataspace.name;

  end
```

データセット

事前定義された変数 **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	文字列	データセットの名前

例

```
// Returns the current dataset name.

export function getValue(): string

begin

    return dataset.name;

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
```

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
```


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
```

エクスポートされた関数

エクスポートされた関数は、EBX® から直接呼び出すことができます。

スクリプトごとに許可されるエクスポートされた関数は 1 つだけです。そのシグネチャ (名前、および戻り型) は、関数フィールドの型によって異なります。

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 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 root.<path>
begin
    ...
end
```

```
end
```

関数フィールドが最初に編集される場合、EBX® はエクスポートされた関数の正しい定義を備えた初期スクリプトを提供します。

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
```

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。

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。

「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。

注意

式

```
if condition-expression then  
  
    statements-a;  
  
else  
  
    statements-b;
```

次のように指定することはできません。

```
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。

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;
```

```

    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 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

```

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 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。

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` を使用して新しい関数フィールドが作成されると、初期スクリプトが作成されます。

次の例は、文字列型のフィールド用に作成されたスクリプトです。


```

export function getValue(): string

begin

    return 'A string value';

end

```

エクスポートされた関数シグネチャ (名前と戻り値の型) はフィールドの型によって異なるため、変更しないでください。

関数フィールド 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 core.date](#)
- [ユニット core.datetime](#)
- [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.resource](#)
- [ユニット core.string](#)
- [ユニット core.time](#)
- [ユニット core.uri](#)

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メソッドを提供するスクリプトユニットです。

ユニット	説明
core.date	日付関数を提供するスクリプトユニットです。
core.datetime	日時関数を提供するスクリプトユニットです。
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関数を提供するスクリプトユニットです。
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です。
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.resource	このスクリプトユニットメソッドは、スキーマタイプ <code>ods:resource</code> と互換性のあるリソース参照を生成します。
core.string	文字列操作関数を提供するスクリプトユニットです。
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。

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>function isNull(value: any_type): boolean</code> 値がnullの場合、 <code>true</code> を返します。
<code>function isTrueOrNull(value: boolean): boolean</code> 値がtrueまたはnullの場合、 <code>true</code> を返します。
<code>function isFalseOrNull(value: boolean): boolean</code> 値がfalseまたはnullの場合、 <code>true</code> を返します。

- function isNull(value: any_type): boolean
- function isTrueOrNull(value: boolean): boolean
- function isFalseOrNull(value: boolean): boolean

function isNull(value: any_type): boolean

値がnullの場合、 `true` を返します。

パラメーター：

任意の型の値。

戻り値：

`値` がnull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。

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：

`true` (`value` がnullまたは `true` の場合)、または `false` の場合。 nullを返すことはありません。

function isFalseOrNull(value: boolean): boolean

値がfalse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：

`true` (`value` がnullまたは `false` の場合)、または `false` の場合。 nullを返すことはありません。

ユニット core.complex

複雑な値を作成および更新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

`of<complexType>():complexType` 複合型のインスタンスを作成します。

`of<foreignKeyType>():foreignKeyType` 外部キータイプのインスタンスを作成します。

- `of<complexType>():complexType`
- `of<foreignKeyType>():foreignKeyType`

`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 record.OfficeAddress
begin

    var value := complex.of<typeof record.OfficeAddress>();

    value.Street := '4323 Broadway';

    value.City := 'New York';

    value.State := 'NY';

    value.Zip := '10019';

    value.Country := 'USA';

    return value;

end
```

関数タイプ：

`complexType`：オブジェクトタイプ。必須です。式 `typeofidentifier` である必要があります。

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。

of<foreignKeyType>():foreignKeyType

外部キータイプのインスタンスを作成します。

例：

```
uses core.complex as complex;

export function getValue(): typeof record.Supervisor
begin
    var foreignKey := complex.foreignKeyOf<typeof record.Supervisor>();

    foreignKey.Id := 435;

    return foreignKey;
end
```

関数タイプ：

externalKeyType：外部キータイプ。必須です。式 `typeof identifier` である必要があります。

戻り値：

新しい外部キーオブジェクト。常に変更可能です。

ユニット core.date

日付関数を提供するスクリプトユニットです。

メソッド

`function addYears(value: date, years: decimal): date` 日付に年を追加します。

`function addMonths(value: date, months: decimal): datetime` 日付に月を追加します。

メソッド

`function addDays(value: date, days: decimal): date` 日付に日を追加します。

`function fromDatetime(value: datetime): date` 日付と時刻を日付に変換します。

`function days(startValue: date, endValue: date): decimal` 2つの日付の間の日数を返します。

`function seconds(startValue: date, endValue: date): decimal` 2つの日付の間の秒数を返します。

- `function addYears(value: date, years: decimal): date`
- `function addMonths(value: date, months: decimal): datetime`
- `function addDays(value: date, days: decimal): date`
- `function fromDatetime(value: datetime): date`
- `function days(startValue: date, endValue: date): decimal`
- `function seconds(startValue: date, endValue: date): decimal`

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

years：追加される年数。整数である必要があります。

戻り値：

新しい日付。パラメーターがnullの場合はnullとなります。

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

months：追加される月数。整数である必要があります。

戻り値：

新しい日付。パラメーターがnullの場合はnullとなります。

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

months：追加される日数。整数である必要があります。

戻り値：

新しい日付。パラメーターがnullの場合はnullとなります。

function fromDatetime(value: datetime): date

日付と時刻を日付に変換します。入力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となります。

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となります。

function seconds(startValue: date, endValue: date): decimal

2つの日付の間の秒数を返します。

パラメーター：

startValue：入力開始日を含む

endValue：入力終了日のみ

戻り値：

秒数。パラメーターがnullの場合はnullとなります。

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 addYears(value: datetime, years: decimal): datetime 日付と時刻に年を追加します。
function addMonths(value: datetime, months: decimal): datetime 日付と時刻に月を追加します。
function 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 addSeconds(value: datetime, seconds: decimal): datetime 日付と時刻に秒を追加します。
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。
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 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 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 addYears(value: datetime, years: decimal): datetime
- function addMonths(value: datetime, months: decimal): datetime
- function addDays(value: datetime, days: decimal): datetime

- function addHours(value: datetime, hours: decimal): datetime
- function addMinutes(value: datetime, minutes: decimal): datetime
- function addSeconds(value: datetime, seconds: decimal): datetime
- function of(year: decimal, month: decimal, day: decimal, hour: decimal, minute: decimal, second: decimal): datetime
- function fromDate(value: date): datetime
- function fromDateAndTime(dateValue: date, timeValue: time): datetime
- function days(startValue: datetime, endValue: datetime): decimal
- function seconds(startValue: datetime, endValue: datetime): decimal

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

months：追加される年数。整数である必要があります。

戻り値：

新しい日時。パラメーターがnullの場合はnullとなります。

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。整数である必要があります。

戻り値：

新しい日時。パラメーターがnullの場合はnullとなります。

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

months：追加される日数。整数である必要があります。

戻り値：

新しい日時。パラメーターがnullの場合はnullとなります。

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。整数である必要があります。

戻り値：

新しい日時。パラメーターがnullの場合はnullとなります。

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。整数である必要があります。

戻り値：

新しい日時。パラメーターがnullの場合はnullとなります。

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となります。

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。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

月：入力月

日：入力日

hours：入力時間

分：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となります。

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。

パラメーター：

value：入力日

戻り値：

新しい日時。パラメーターがnullの場合はnull。

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となります。

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。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：入力された終了日時

戻り値：

日数。パラメーターがnullの場合はnullとなります。

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（両端を含む）

endValue: 入力された終了日時のみ

戻り値：

秒数。パラメーターがnullの場合はnullとなります。

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

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 isEmpty(value: list<ItemType>): boolean 値がnullまたはゼロサイズのリストの場合、**true** を返します。

メソッド

`function contains(values: list<ItemType>, element: ItemType): boolean` `values` リストに `element` が含まれている場合、`true` を返します。

`function containsAll(values: list<ItemType>, element: ItemType): boolean` `values` リストにすべての `element` が含まれている場合、`true` を返します。

`procedure add(values: list<ItemType>, element: ItemType)` `element` を `values` リストに追加します。

`function remove(values: list<ItemType>, element: ItemType): boolean` `values` リストから `element` の最初の出現を削除します。

`procedure addAll(values: list<ItemType>, elements: list<ItemType>): boolean` すべての `elements` を `values` リストに追加します。

`function removeAll(values: list<ItemType>, elements: list<ItemType>): boolean` `values` リストからすべての `elements` を削除します。

`function retainAll(values: list<ItemType>, elements: list<ItemType>): boolean` `elements` に含まれている `values` リスト内のエレメントのみを保持します。

`function indexOf (values : list<ItemType>, element : ItemType)` `values` リストで最初に出現する `element` のインデックスを返します。

`function lastIndexOf(values: list<ItemType>, element: ItemType)` `values` リストで最後に出現した `element` のインデックスを返します。

`function subList<ItemType>(values: list<ItemType>, startIndex: decimal, endIndex: decimal): list<ItemType>` `values` 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。

`procedure sort(values: list<ItemType>)` 指定された `values` リストを、そのエレメントの自然順序に従って昇順で並べ替えます。

`procedure reverseSort(values: list<ItemType>)` 指定された `values` リストを、そのエレメントの自然順序に従って降順で並べ替えます。

`procedure reverse(values: list<ItemType>)` `values` のエレメントの順序を逆にします。

- `function of<ItemType>(value: ItemType [, value: ItemType]...): list<ItemType>`
- `function isEmpty(value: list<ItemType>): boolean`

- function contains(values: list<ItemType>, element: ItemType): boolean
- function containsAll(values: list<ItemType>, element: ItemType): boolean
- procedure add(values: list<ItemType>, element: ItemType)
- function remove(values: list<ItemType>, element: ItemType): boolean
- procedure addAll(values: list<ItemType>, elements: list<ItemType>): boolean
- function removeAll(values: list<ItemType>, elements: list<ItemType>): boolean
- function retainAll(values: list<ItemType>, elements: list<ItemType>): boolean
- function indexOf (values : list<ItemType>、 element : ItemType)
- function lastIndexOf(values: list<ItemType>, element: ItemType)
- function subList<ItemType>(values: list<ItemType>, startIndex: decimal, endIndex: decimal): list<ItemType>
- procedure sort(values: list<ItemType>)
- procedure reverseSort(values: list<ItemType>)
- procedure reverse(values: list<ItemType>)

function of<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。すべてのエレメントは同じタイプである必要があります。

例：

```
uses core.list as list;

function getCities(): list<string>
begin
    return list.of('Paris', 'Bruxelles', 'Berlin');
end
```

関数タイプ：

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である必要があります。

戻り値：

新しいリスト。更新可能です。

function isEmpty(value: list<ItemType>): boolean

値がnullまたはゼロサイズのリストの場合、`true` を返します。

パラメーター：

value：入力リスト

戻り値：

`value` がnullまたはゼロサイズのリストの場合は `true`、それ以外の場合は `false`。nullを返すことはありません。

function contains(values: list<ItemType>, element: ItemType): boolean

`values` リストに `element` が含まれている場合、`true` を返します。

パラメーター：

value：入力リスト。コンテンツは、`element` と同じ種類である必要があります。

element：検索する入力エレメント。`value` の内容と同じ種類である必要があります。

戻り値：

`value` に `element` が含まれている場合は `true`、それ以外の場合は `false` となります。

値がnullの場合、`null` を返します。

function containsAll(values: list<ItemType>, element: ItemType): boolean

`values` リストにすべての `element` が含まれている場合、`true` を返します。

パラメーター：

value：入力リスト。コンテンツは、`element` と同じ種類である必要があります。

element：検索する入力エレメント。 `値` > と同じ種類である必要があります。

戻り値：

`value` にすべての `element` が含まれている場合は `true`、それ以外の場合は `false` となります。

パラメーターがnullの場合、 `null` を返します。

procedure add(values: list<ItemType>, element: ItemType)

`element` を `values` リストに追加します。

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。

function remove(values: list<ItemType>, element: ItemType): boolean

`values` リストから `element` 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：

`true` （ `values` リストの内容がこの呼び出しによって変更された場合）、それ以外の場合は `false` となります。

procedure addAll(values: list<ItemType>, elements: list<ItemType>): boolean

すべての `elements` を `values` リストに追加します。

パラメーター：

values：入力リスト

element：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：

`values` リストの内容がこの呼び出しによって変更された後は `true`、それ以外の場合は `false` となります。

function removeAll(values: list<ItemType>, elements: list<ItemType>): boolean

`values` リストからすべての `elements` を削除します。

パラメーター：

`values`：入力リスト

`element`：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：

`true`（`values` リストの内容がこの呼び出しによって変更された場合）、それ以外の場合は `false` となります。

function retainAll(values: list<ItemType>, elements: list<ItemType>): boolean

`elements` に含まれている `values` リスト内のエレメントのみを保持します。つまり、`values` リストから、`elements` に含まれてい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

`element`：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：

`true`（`values` リストの内容がこの呼び出しによって変更された場合）、それ以外の場合は `false` となります。

function indexOf(values: list<ItemType>, element: ItemType)

`values` リストで最初に出現する `element` 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：

values リストで最初に出現する element のインデックス、または見つからない場合は null。

function lastIndexOf(values: list<ItemType>, element: ItemType)

values リストで最後に出現した element 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。

戻り値：

values リストで最後に出現した element のインデックス、または見つからない場合は null。

function subList<ItemType>(values: list<ItemType>, startIndex: decimal, endIndex: decimal): list<ItemType>

values リストの要素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。

関数タイプ：

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である必要があります。

endIndex：入力終了インデックス。正の整数またはゼロである必要があります。

戻り値：

新しいリスト。パラメーターがnullの場合はnullとなります。

procedure sort(values: list<ItemType>)

指定された `values` リストを、そのエレメントの自然順序に従って昇順で並べ替えます。
`values` がnullまたはゼロサイズのリストの場合、何もしません。

パラメーター：

values：ソートする入力リスト

procedure reverseSort(values: list<ItemType>)

指定された `values` リストを、そのエレメントの自然順序に従って降順で並べ替えます。
`values` がnullまたはゼロサイズのリストの場合、何もしません。

パラメーター：

values：ソートする入力リスト。

procedure reverse(values: list<ItemType>)

`values` のエレメントの順序を逆にします。 `values` がnullまたはゼロサイズのリストの場合、何もしません。

パラメーター：

values：逆にする入力リスト。

ユニット core.locale

ロケール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
<code>function of(language: string, country: string): locale</code> ロケールを作成します。
<code>function fromString(value: string): locale</code> 文字列からロケールを作成します。

- `function of(language: string, country: string): locale`
- `function fromString(value: string): locale`

function of(language: string, country: string): locale

ロケールを作成します。

パラメーター：

言語：ISO 639alpha-2またはalpha-3言語コード。

国：ISO 3166alpha-2国コードまたはUNM.49数値-3市外局番。

戻り値：

ロケール。

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：

値：ロケール。たとえば、「en-US」。

戻り値：

ロケール。

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
<code>procedure error(message: string)</code> エラーメッセージをログに記録します。
<code>procedure warn(message: string)</code> 警告メッセージをログに記録します。
<code>procedure info(message: string)</code> 情報メッセージをログに記録します。

- `procedure error(message: string)`
- `procedure warn(message: string)`
- `procedure info(message: string)`

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：

値：message：メッセージ。

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：

値：message：メッセージ。

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：

値：message：メッセージ。

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 sin(value: decimal): decimal</code> 角度の三角関数の正弦を返します。
<code>function cos(value: decimal): decimal</code> 角度の三角余弦を返します。
<code>function tan(value: decimal): decimal</code> 角度の三角法のタンジェントを返します。
<code>function asin(value: decimal): decimal</code> 値のアークサインを返します。
<code>function acos(value: decimal): decimal</code> 値のアークコサインを返します。
<code>function atan(value: decimal): decimal</code> 値のアークタンジェントを返します。
<code>function toDegrees(value: decimal): decimal</code> 度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。
<code>function toRadians(value: decimal): decimal</code> ラジアンで測定された角度を、度で測定されたほぼ同等の角度に変換します。
<code>function exp(value: decimal): decimal</code> オイラーの数値eを <code>10進数</code> の累乗で返します。
<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 scaleByPowerOfTen(value: decimal, power: decimal): decimal</code> <code>value</code> のスケールを <code>power</code> の10で返します。

メソッド

`function sqrt(value: decimal): decimal` 値の正の平方根を返します。

`function cbrt(value: decimal): decimal` `10進数` 値の立方根を返します。

`function abs(value: decimal): decimal` `10進数` 値の絶対値を返します。

`function round(value: decimal): decimal` 入力値の整数の丸め値を返します。

`function roundHalfEven(value: decimal): decimal` 入力値の整数の四捨五入値を返します。

`function roundHalfDown(value: decimal): decimal` 入力値の整数の切り捨て値を返します。

`function roundUp(value: decimal): decimal` 入力値の整数の切り上げ値を返します。

`function roundDown(value: decimal): decimal` 入力値の整数の切り捨て値を返します。

`function floor(value: decimal): decimal` 入力値の整数のフロア値を返します。

`function ceil(value: decimal): decimal` 入力値の整数フロア値を返します。

`function pi(): decimal` `pi` 値を返します：3。

`function e(): decimal` `e` 値を返します：2。

- `function sin(value: decimal): decimal`
- `function cos(value: decimal): decimal`
- `function tan(value: decimal): decimal`
- `function asin(value: decimal): decimal`
- `function acos(value: decimal): decimal`
- `function atan(value: decimal): decimal`
- `function toDegrees(value: decimal): decimal`
- `function toRadians(value: decimal): decimal`
- `function exp(value: decimal): decimal`
- `function log(value: decimal): decimal`
- `function scaleByPowerOfTen(value: decimal, power: decimal): decimal`
- `function sqrt(value: decimal): decimal`

- function cbrt(value: decimal): decimal
- function abs(value: decimal): decimal
- function round(value: decimal): decimal
- function roundHalfEven(value: decimal): decimal
- function roundHalfDown(value: decimal): decimal
- function roundUp(value: decimal): decimal
- function roundDown(value: decimal): decimal
- function floor(value: decimal): decimal
- function ceil(value: decimal): decimal
- function pi(): decimal
- function e(): decimal

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IEEE754浮動小数点を使用します。

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：

入力値のサイン。入力値がnullの場合はnull。

function cos(value: decimal): decimal

角度の三角余弦を返します。

計算は倍精度変換後に実行され、倍精度64ビットIEEE754浮動小数点を使用します。

パラメーター：

value：ラジアン単位の入力値。

戻り値：

入力値の余弦。入力値がnullの場合はnullとなります。

function tan(value: decimal): decimal

角度の三角法のタンジェントを返します。

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IEEE754浮動小数点を使用します。

パラメーター：

value：ラジアン単位の入力値。

戻り値：

入力値の接線。入力値がnullの場合はnullとなります。

function asin(value: decimal): decimal

値のアークサインを返します。

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IEEE754浮動小数点を使用します。

パラメーター：

value：入力値。絶対値は1より大きくてはなりません。

戻り値：

入力値のアークサイン。入力値がnullの場合はnullとなります。

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 pi 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IEEE754浮動小数点を使用します。

パラメーター：

value：入力値。絶対値は1より大きくてはなりません。

戻り値：

入力値のアークコサイン。入力値がnullの場合はnullとなります。

function atan(value: decimal): decimal

値のアーク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IEEE754浮動小数点を使用します。

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。

function toDegrees(value: decimal): decimal

度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IEEE754浮動小数点を使用します。

パラメーター：

value：ラジアン単位の入力値

戻り値：

度単位の入力値の測定値。入力値がnullの場合はnull。

function toRadians(value: decimal): decimal

ラジアンで測定された角度を、度で測定されたほぼ同等の角度に変換します。

ラジアンから度への変換は一般的に不正確です。

ユーザーは、`*cos (toRadians (90.0) ) *`が `0.0` と正確に等しいことを **期待しない** 必要があります。

計算は倍精度変換後に実行され、倍精度64ビットIEEE754浮動小数点を使用します。

パラメーター：

値：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。

function exp(value: decimal): decimal

オイラーの数値eを `10進数` の累乗で返します。

計算は倍精度変換後に実行され、倍精度64ビットIEEE754浮動小数点を使用します。

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。

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IEEE754浮動小数点を使用します。

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となります。

function scaleByPowerOfTen(value: decimal, power: decimal): decimal

`value` のスケールを `power` の10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。整数である必要があります。

戻り値：

`power` の10でスケーリングされた入力値、または入力値または累乗がnullの場合はnull。

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IEEE754浮動小数点を使用します。

パラメーター：

value：入力値。正またはnullである必要があります。

戻り値：

入力値の正の平方根。入力値がnullの場合はnull。

function cbrt(value: decimal): decimal

`10進数` 値の立方根を返します。計算では、倍精度64ビットIEEE754浮動小数点を使用します。

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。

function abs(value: decimal): decimal

`10進数` 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。

function round(value: decimal): decimal

入力値の整数の丸め値を返します。この方法では、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。

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：

入力値の半偶数値を丸めます。入力値がnullの場合はnullです。

function roundHalfDown(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：

入力値の半分を切り捨てた値入力値がnullの場合はnullとなります。

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。

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となります。

function floor(value: decimal): decimal

入力値の整数の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となります。

function ceil(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ceil値、または入力値がnullの場合はnullとなります。

function pi(): decimal

`pi` 値を返します：3.141592653589793238462643383279503

戻り値：

円周率の値。

function e(): decimal

e 値を返します：2.718281828459045235360287471352662

戻り値：

e値。

ユニット core.resource

このスクリプトユニットメソッドは、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します。

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 toImage(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 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。

`function toHtml(moduleName: string, filePath: string): string` 画像リソースへの参照を返します。

- `function toImage(moduleName: string, filePath: string): string`

- `function tolcon(moduleName: string, filePath: string): string`
- `function toJavaScript(moduleName: string, filePath: string): string`
- `function toStyleSheet(moduleName: string, filePath: string): string`
- `function toHtml(moduleName: string, filePath: string): string`

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。

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。

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。

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。

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。

ユニット core.string

文字列操作関数を提供するスクリプトユニットです。

メソッド

function isEmpty(value: string): boolean 値がnullまたは長さゼロの文字列の場合、`true` を返します。

function right(value: string, index: decimal): string `value` 文字列から適切な文字を含む新しい文字列を返します。

function substring(value: string, startIndex: decimal, endIndex: decimal): string `value` 文字列の文字のサブシーケンスを含む新しい文字列を返します。

function toUpperCase(value: string): string 入力 `value` 文字列から大文字までのすべての文字を含む新しい文字列を返します。

メソッド

function toLowerCase(value: string): string 入力 `value` 文字列から小文字までのすべての文字を含む新しい文字列を返します。

function replaceAll(value: string, pattern: string, replacement: string): string 入力 `value` 文字列内のすべての `pattern` オカレンスが `replacement` に置き換えられた新しい文字列を返します。

function replaceFirst(value: string, pattern: string, replacement: string): string 入力 `value` 文字列の最初のパターンオカレンスが `replacement` に置き換えられた新しい文字列を返します。

function startsWith(value: string, prefix: string): boolean `value` 文字列がプレフィックスで始まる場合、`true` を返します。

function endsWith(value: string, suffix: string): boolean `value` 文字列がサフィックスで終わる場合、`true` を返します。

function contains(value: string, searchString: string): boolean `value` 文字列に `searchString` が含まれている場合、`true` を返します。

function join(separator: 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 isEmpty(value: string): boolean**
- **function right(value: string, index: decimal): string**
- **function substring(value: string, startIndex: decimal, endIndex: decimal): string**
- **function toUpperCase(value: string): string**
- **function toLowerCase(value: string): string**
- **function replaceAll(value: string, pattern: string, replacement: string): string**
- **function replaceFirst(value: string, pattern: string, replacement: string): string**
- **function startsWith(value: string, prefix: string): boolean**
- **function endsWith(value: string, suffix: string): boolean**
- **function contains(value: string, searchString: string): boolean**
- **function join(separator: string [, substring: string]...): string**

function isEmpty(value: string): boolean

値がnullまたは長さゼロの文字列の場合、`true` を返します。

パラメーター：

value：文字列値

戻り値：

value がnullまたは長さ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。

function right(value: string, index: decimal): string

value 文字列から適切な文字を含む新しい文字列を返します。

文字列は、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。ゼロ以上で文字列の長さより短い整数である必要があります。

戻り値：

新しい文字列。パラメーターがnullの場合はnullとなります。

function substring(value: string, startIndex: decimal, endIndex: decimal): string

value 文字列の文字のサブ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。ゼロ以上で文字列の長さより短い整数である必要があります。

endIndex：入力終了インデックス。 startIndex以上で、文字列の長さより短い整数である必要があります。

戻り値：

新しい文字列。パラメーターがnullの場合はnullとなります。

function toUpperCase(value: string): string

入力 `value` 文字列から大文字までのすべての文字を含む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となります。

function toLowerCase(value: string): string

入力 `value` 文字列から小文字までのすべての文字を含む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となります。

function replaceAll(value: string, pattern: string, replacement: string): string

入力 `value` 文字列内のすべての `pattern` オカレンスが `replacement` に置き換えられ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となります。

function replaceFirst(value: string, pattern: string, replacement: string): string

入力 `value` 文字列の最初のパターンオカレンスが `replacement` に置き換えられ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となります。

function startsWith(value: string, prefix: string): boolean

value 文字列がプレフィックスで始まる場合、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：

value が prefix で始まる場合は true、それ以外の場合は false となります。

パラメーターがnullの場合、null を返します。

function endsWith(value: string, suffix: string): boolean

value 文字列がサフィックスで終わる場合、true を返します。

パラメーター：

value：入力文字列

suffix：検索する入力サフィックス

戻り値：

value が suffix で終わる場合は true、それ以外の場合は false となります。

パラメーターがnullの場合、null を返します。

function contains(value: string, searchString: string): boolean

value 文字列に searchString が含まれている場合、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：

value に searchString が含まれている場合は true、それ以外の場合は false となります。

パラメーターがnullの場合、null を返します。

function join(separator: 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：

separator：入力セパレータ文字列

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である必要があります。

戻り値：

新しい文字列は、セパレータ入力で区切られたすべての入力サブ文字列で構成されます。

パラメーターがnullの場合、またはサブ文字列が指定されていない場合は、nullを返します。

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
function addHours(value: time, hours: decimal): time 時間に時間を追加します。
function addMinutes(value: time, minutes: decimal): time 時間に分を追加します。
function addSeconds(value: time, seconds: decimal): time 時間に秒を追加します。
function of(hours: decimal, minutes: decimal, seconds: decimal): time 時間、分、秒を指定して時間を作成します。
function fromDatetime (value: datetime): datetime 日付と時刻を時刻に変換します。
function seconds(startValue: time, endValue: time): decimal 2回の間の秒数を返します。

- function addHours(value: time, hours: decimal): time
- function addMinutes(value: time, minutes: decimal): time
- function addSeconds(value: time, seconds: decimal): time
- function of(hours: decimal, minutes: decimal, seconds: decimal): time
- function fromDatetime (value: datetime): datetime
- function seconds(startValue: time, endValue: time): decimal

function addHours(value: time, hours: decimal): time

時間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。整数である必要があります。

戻り値：

新しい時刻。パラメーターがnullの場合はnull。

function addMinutes(value: time, minutes: decimal): time

時間に分を追加します。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。整数である必要があります。

戻り値：

新しい時刻。パラメーターがnullの場合はnull。

function addSeconds(value: time, seconds: decimal): time

時間に秒を追加します。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。

function of(hours: decimal, minutes: decimal, seconds: decimal): time

時間、分、秒を指定して時間を作成します。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：

hours：入力時間

分：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。

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。

function seconds(startValue: time, endValue: time): decimal

2回の間の秒数を返します。

パラメーター：

startValue：入力開始日時（両端を含む）

endValue: 入力された終了日時のみ

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。

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。

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：入力値例：https://www.tibco.com

戻り値：

URI

Java API (英語) Reference

- [TIBCO EBX® 6.1.2 Java API \(英語\)](#)

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、[TIBCO 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。

[TIBCO 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 サポートへの問い合わせ方法

[TIBCO Support](#) の概要をご覧ください。次の方法で TIBCO Support に連絡できます。

- 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、[TIBCO Support 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、TIBCO 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。[TIBCO Support Web](#) サイトにログインするには、ユーザー名とパスワードも必要です。ユーザー名をお持ちでない場合は、Web サイトで [をクリックしてリクエスト](#) することができます。

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します。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することができます。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

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, TIBCO EBX®, TIBCO EBX® Data Exchange Add-on, TIBCO EBX® Add-on's Root Module, TIBCO EBX® Digital Asset Manager Add-on, TIBCO EBX® Match and Merge Add-on, TIBCO EBX® Data Model and Data Visualization Add-on, TIBCO EBX® Insight Add-on, TIBCO EBX® Graph View Add-on, TIBCO EBX® Add-on for Oracle Hyperion EPM, TIBCO EBX® Information Governance Add-on, TIBCO EBX® GO Add-on, TIBCO EBX® Activity Monitoring Add-on, TIBCO EBX® Rule Portfolio Add-on, and TIBCO EBX® Information Search Add-on are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

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 "READ ME" FILES.

This and other products of Cloud Software Group, Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright© 2006-2023. Cloud Software Group, Inc. All Rights Reserved

リリースノート

- [バージョン 6.1.2](#)
- [すべてのリリースノート](#)

6.1.2

- 6.1 サービスパック
 - 6.1.2
 - 6.1.1
- 監査証跡
- タイルビュー
- チームアップ
- 階層
- レコードメタデータ
- ステージング
- ワークフロー
- パースペクティブ
- データモデリング
 - エンティティ名
 - 検索戦略
- データ検証
- リクエストとクエリの最適化
- Java API
- データサービス
 - REST Toolkit API
 - ビルトイン RESTful サービス
 - ビルトイン SOAP 操作

- EBX® メイン構成ファイル
- EBX® Container Edition
- このリリースの制限事項
 - 階層の制限事項
 - クエリの制限事項
 - 値のラベル付けの制限事項
 - データ検証の制限事項
 - リンクフィールドの制限事項
- バージョンアップグレード
 - コンパイルおよびビルド環境
 - サポートされる環境の変更
 - 既存の EBX® 5 環境の移行
 - 既存の EBX® 6 環境からのアップグレード
 - データセット作成オプションの削除
 - 下位互換性
- 6.1.2 既知の問題
 - ユーザーインターフェイス
- 6.1.2 修正済みの問題
 - コアエンジン
 - データサービス
 - データワークフロー
 - ドキュメント
 - 階層
 - チームアップ
 - ユーザーインターフェイス
 - ユーザーと権限
- 6.1.1 修正済みの問題
 - クラウド

- Container Edition
- コアエンジン
- データモデリング
- データサービス
- データサービス -REST ツールキット
- データ検証
- ドキュメント
- 階層
- ステージング
- ユーザーインターフェイス

- 6.1.0 修正済みの問題

- 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
- データビュー - 階層
- データワークフロー
- データスペースと履歴
- ドキュメント
- インフラストラクチャー
- クエリ
- チームアップ
- ユーザーインターフェイス
- ユーザーと権限

6.1 サービス パック

6.1.2

6.1.2 は、2023 年 9 月 8 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。

ログの構成	Apache Log4j™ 2、<code>ebx.properties</code> メイン構成ファイル、または両方のメカニズムの組み合わせを使用して、ログメッセージを構成できるようになりました。 詳細については、ログの構成を参照してください。
下位互換性 - リンクフィールドのプロパティの継承	リンクされたフィールドは、そのプロパティのほとんどをターゲットノードから継承するようになりました。プロパティとファセットの継承の結果、リンクされたフィールドへのファセットの追加は廃止されました。警告メッセージがスローされ、ファセットは無視されます。さらに、検証はターゲットノードで行われ、リンクフィールドにはターゲットノードの値のみが表示されるため、リンクフィールドの検証は常に無効になります。 詳細については、リンクされたフィールドを参照してください。
クエリでのプログラムによるアクセスルールの考慮	クエリにセッションを設定して実行すると、セッションで定義されたプログラムによるアクセスルールが結果で考慮されるようになりました。これは、以前のバージョンで、すでに考慮されていたスクリプト化アクセスルールに追加されるものです。

Kubernetes プラットフォームのサポート	EBX® Container Edition は Red Hat® OpenShift®と互換性を持つようになりました。Helm チャートのサンプルが、GitHub から提供されています。 詳細については、Kubernetes を参照してください。
Java EE 展開 - 展開	Cookie の管理はデフォルトで有効になります。 詳細については、FrontServlet 用アプリケーションサーバーの構成を参照してください。
既知の問題	6.1.2 既知の問題セクションを参照してください。
修正済みの問題	6.1.2 修正済みの問題セクションを参照してください。

6.1.1

6.1.1 は、2023 年 7 月 10 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。

EBX® の SQL	EBX® SQL API には、単一パラメーターの FK_AS_STRING ビルトイン関数が含まれるようになりました。これにより、3 つのパラメーターを使用するバージョンに間違った引数を指定するという問題が回避されます。場合によっては、引き続き 3 つのパラメーターを使用するバージョンを使用する必要があります。詳細については、EBX® SQL 関数を参照してください。
REST 例外処理	REST Toolkit の例外 JSON 形式 は、ビルトイン RESTful サービスの例外 JSON 形式と一致するように更新されました (バージョン 5 と同じ)。

関連付けの検索	クイック検索に関連付けフィールドが含まれるようになりました。さらに、高度な検索を実行するときに、「テキスト検索」演算子に関連付けに使用できます。検索は、関連するレコードのラベルで使用される各フィールドに対して実行されます。 パフォーマンスを向上させるには、新しいプロパティ <code>osd:defaultView/hiddenInQuickSearch="true false"</code> を使用して、クイック検索ツールからフィールドと関連付けを除外します。 詳細については、デフォルトビューを参照してください。
複雑な検索フィルター	複数の検索条件を論理演算子でグループ化して、複雑なフィルターを作成できるようになりました。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。詳細については、複雑なフィルターの作成を参照してください。
ステージング	次の新しいステージングエレメントタイプが追加されました。 • 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 起動時のステージングアーカイブの自動インポート機能も追加されました。詳細については、ステージングを参照してください。
権限：アクセス権限の構成	データセットの権限構成画面は、新しいグリッド構成サービスに置き換えられました。このサービスにより、ユーザーは1つ

	の画面ですべてのデータセットの権限を設定および確認できるようになります。
権限：履歴へのアクセスと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が設定されている場合に、テーブルの履歴にアクセスするために許可されたプロファイルのリストを構成できるようになりました。詳細については、 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 を参照してください。
下位互換性 – 無効なコンテキストでの値の取得	以前のバージョンでは、指定されたノードが現在のレコードと同じテーブルに属していない場合、 <code>ReadContext.get(SchemaNode)</code> メソッドは正しく動作しませんでした。現在は、指定されたノードの整合性がチェックされ、現在のレコードと同じテーブルに属していない場合は <code>IllegalArgumentException</code> がスローされます。
ビジネスオブジェクト	TIBCO EBX® Match and Merge Add-on が登録されていない場合でも、ビジネスオブジェクト拡張機能を DMA で使用できるようになりました。 詳細については、 ビジネスオブジェクト 、 <code>ビジネスオブジェクトモデル</code> 、および <code>MainEntity</code> を参照してください。
修正済みの問題	6.1.1 修正済みの問題 セクションを参照してください。

Java API

Java API は、次のトピックで改善されました。

- `データスペース検証レポート`

データスペースの検証レポートを表す新しいインターフェイスが追加されました。指定されたポリシーに従って、データスペースに含まれるすべてのデータセットの検証レポートを収集します。また、検証結果を利用するためのいくつかの方法も提供します。詳細については、`DataspaceValidationReport` を参照してください。

- `AdaptationHome`

データスペースを検証するために 2 つの新しいメソッドが追加されました。

詳細については、`AdaptationHome.getValidationReport` および `AdaptationHome.getValidationReport` を参照してください。

注意:

このデータスペース内のすべてのデータセットは、指定されたポリシーに従って検証されます。したがって、データスペースの検証には長い時間がかかる可能性があります。

データスペースが検証フレームワークによって使用されていることを識別するための新しいメソッドが追加されました。

`AdaptationHome.isusedByValidation` を参照してください。

- `DependenciesDefinitionContext`

検証の依存関係を指定するための 5 つの新しいメソッドが追加されました。

詳細については、`DependenciesDefinitionContext.addDependencyToModify`、`DependenciesDefinitionContext.addDependencyToInsertAndDelete`、`DependenciesDefinitionContext.addDependencyToInsertDeleteAndModify`、および `DependenciesDefinitionContext.addDependencyToInsertDeleteAndModifyInOtherInstance` を参照してください。

注意：* 6.1.0** は 2023 年 3 月 22 日にビルドされました。*

監査証跡

「監査証跡」機能が拡張されました。現在この機能は、比較および検証サービスへのユーザーアクセスを処理します。属性「service-mode」は、「User service」イベントの特定のケースで記録されるようになりました。詳細については、[監査証跡](#)を参照してください

タイルビュー

タイルを使用してデータを表示するカスタムビューを作成できるようになりました。タイルはグリッドに配置され、タイトル (必須)、サブタイトル、画像、説明など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、ビューに適用する並べ替え基準とフィルター基準を定義できます。ツールバーも追加できます。詳細については、[タイルビュー](#)を参照してください。

チームアップ

チームアップ機能は、データのコラボレーションと連携のための新しい方法を導入します。ビジネスユーザーは、レコードにコメントを残したり、レコードを評価したり、レコードにタグを割り当てたりすることができます。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。詳細については、[コラボレーション機能 \(チームアップ\)](#) および [コラボレーション機能 \(チームアップ\)](#) の概要を参照してください。

階層

このリリースには、次の新しい階層機能と更新された機能が含まれています。

- 列挙型、ブール値型、日付型に加えて、サポートされている列挙可能なデータ型には文字列型と整数型も含まれるようになりました。これらの新しい依存関係の制限については、[階層の制限事項](#)を参照してください。
- 単一のテーブルに含まれる列に基づいて、Group By 階層ビューを作成できるようになりました。列は階層の次元を設定します。このタイプの階層に使用されるフィールドは、外部キーまたは列挙可能タイプである必要があります。詳細については、[階層によるグループ化](#)を参照してください。
- メンバーの親が 1 つのレベルをスキップする不規則な階層のビューを生成するために、レベルのスキップが利用できるようになりました。不規則な階層表示を有効にするには、新しいプロパティ `階層ビュー構成のグループ` で使用できます。すべての外部キーがターゲットディメンションテーブルに存在する必要があることに注意してください。詳細については、[不規則な階層](#)を参照してください。
- 日付データ型のフィールドを、時間ベースの階層ディメンションとして定義できるようになりました。階層ビューの作成中に最後の画面に到達したら、画面上部の日付フィールドを選択し、メニューを使用して、ディメンションの表示方法を選択します。詳細については、[時間ベースの階層ディメンション](#)を参照してください。

- 再帰的なルートリーフの削除は、このリリースで再び有効になりました。再帰的なケースでは、検索フィルターが適用されている場合、または が有効になっている場合、このオプションはルートリーフが表示されないようにします。

レコードメタデータ

レコードのメタデータがテーブルの構造に自動的に追加されるようになりました。メタデータノードを使用すると、レコードに関する作成者、作成時間、最終変更者、最終変更時間、および一意の識別子 (UUID) の情報を取得できます。このメタデータは現在データモデルの一部であるため、次のことができます。

- 検索基準の一部としてメタデータフィールドを使用して、レコードを検索するには、クイック検索機能を使用します。
- XPath または SQL を使用して、API でクエリを実行します。
- 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。

詳細については、[レコードメタデータ](#)、`SchemaMetadataConstants`、`SchemaNode.getNodeChildren` を参照してください。

ステージング

新しいステージング機能を使用すると、EBX® リポジトリ内の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。詳細については、[ステージング](#)を参照してください。

ワークフロー

ワークフローにいくつかの改善が加えられました。

ワークフローランチャー

ワークフローランチャービルトインサービスは、次のように改善されました。

- ワークフローランチャーをアクティブ化するとき、複数のテーブルを同時に選択できるようになりました。
- レコードの選択が必要なワークフローを起動または表示するプロセスが簡素化されました。ユーザーは、テーブルビュー

	の行に表示されるツールバーからワークフローを表示および起動できるようになりました。これにより、ワークフローを起動する前にレコードを開いたり、[アクション] メニューを使用してワークフローを見つけたりする必要がなくなります。詳細については、データセクションのワークフローランチャーを参照してください。
ワークフローモデルエディターの改善	- 新しいステップを作成するときに、検証されていないレイアウトの変更が保存されていない場合に、ポップアップが表示されるようになりました。 - リンクの配置をカスタマイズし、より一般的には、新しいコネクターを使用してステップの配置をカスタマイズします。 - レイアウトの未検証の変更が保存されていない場合、ポップアップが表示されるようになりました。新しい自動配置モードである「縦方向レイアウト」が利用可能です。
新しいビルトインスクリプトタスク	- レコード文字列値の更新 - レコードのフィールド値の取得 (文字列) - データスペースラベルの更新 - ワークフローラベルの更新

パースペクティブ

EBX® への接続時に、新しいパースペクティブモードをデフォルトとして設定できるようになりました。次の 2 つのモードから選択します。

- **お気に入りのパースペクティブ**：接続時にお気に入りのパースペクティブを適用します。これがデフォルトのモードです。

- 最後に使用したパースペクティブ (このリリースの新機能)：接続時に最後にアクセスしたパースペクティブを適用します。

[パースペクティブモード](#)も参照してください。

データモデリング

データモデリングには、次の新機能が含まれています。

- [エンティティ名](#)
- [検索戦略](#)

エンティティ名

デフォルトでは、データモデルで名前が明示的に定義されていない場合、テーブルのエンティティ名が自動的に作成されるようになりました。エンティティ名を使用すると、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直接参照できます。これは、SQL リクエストのエイリアスと WSDL 操作のサフィックスを定義することによって実現されます。この新しい動作は、既存のデータモデルに影響を与える可能性があります。

[下位互換性](#)セクションに記述されているように、この新しい動作により、データモデルのコンパイル時にエラーが発生する可能性があることに注意してください。

検索戦略

レーベンシュタイン、Soundex、Double Metaphone、および NGram 検索戦略を文字列フィールドで定義できるようになりました。以前は、これらの戦略は EBX® Information Search Add-on が展開された場合にのみ利用可能でした。データモデルがすでにそれらを使用している場合は、それ以上のアクションは必要ありません。

詳細については、[文字列の高度な戦略](#)を参照してください。

データ検証

レコードレベルチェックでテーブルに制約を作成する場合、レコードの作成ステータスに応じて動作を指定できるようになりました。インジケータを使用して、現在のレコードが作成されているかどうかを確認できるようになりました。現在のレコードが作成されている場合、メソッド

`isRecordCreation()` は `true` を返します。詳細については、

`ValueContextForValidationOnRecord.isRecordCreation` を参照してください。

リクエストとクエリの最適化

いくつかの機能のパフォーマンスが向上しています。特に、次の点が向上しています。

- `リクエスト` の実行時に、リクエストからクエリへの変換をキャッシュするリクエストキャッシュが導入されました。詳細については、[リクエストキャッシュの有効化](#)を参照してください。
- `RequestResult` サイズ関連のメソッドは、複数回実行するとパフォーマンスが向上します。
- 主キーの値に基づくリクエストは、より最適化された `RequestResult` の実装を使用して実行されます。

Java API

Java API は、次のトピックで改善されました。

反復可能な <code>RequestResult</code>	<code>RequestResult</code> は <code>Iterable<Adaptation></code> を拡張するようになり、簡素化された構文を使用してトラバースできるようになりました。 詳細については、<code>RequestResult</code> を参照してください。
<code>リポジトリ</code>	新しいメソッド <code>Repository.getDefaultWhenFullyInitialized()</code> は、リポジトリが完全に初期化され、SOAP、REST ビルトイン、REST Toolkit リクエスト、および UI を処理できるようになるまで、呼び出しスレッドをブロックします。 詳細については、<code>Repository.getDefaultWhenFullyInitialized</code> を参照してください。
<code>SchemaLocation</code>	パッケージ化された埋め込みデータモデルのプロパティを取得するために、いくつかの

	メソッドが <code>SchemaLocation</code> に追加されました。 詳細については、 <code>SchemaLocation.isPublication</code>、 <code>SchemaLocation.getPublicationName</code>、 および <code>SchemaLocation.getPathInModule</code> を参照してください。
<code>UICustomViewContext</code>	新しい Java インターフェイス <code>UICustomViewContext</code> により、以下を取得できます。 • 特定のユーザーに推奨されるビュー。 • 特定のユーザーのお気に入りのビュー。 • 現在適用されているビュー。 詳細については、<code>UICustomViewContext</code> を参照してください。
<code>ValidationReportItemSubject</code>	<code>ValidationReportItemSubject</code> に 2 つの新しいメソッドが追加され、検証アイテムのサブジェクトタイプがチェックされます。 • 詳細については、 <code>ValidationReportItemSubject.isDatasetOrRecord</code> を参照してください。 • 詳細については、 <code>ValidationReportItemSubject.isTableSubject</code> を参照してください。

データサービス

次のセクションでは、データサービスの新機能と改善点について説明します。

- REST Toolkit API
- ビルトイン RESTful サービス

- ビルトイン SOAP 操作

REST Toolkit API

OpenAPI 操作は、「api」カテゴリを REST Toolkit リソースの生成ドキュメントに拡張するようになりました。詳細については、[OpenAPI](#) を参照してください。

ビルトイン RESTful サービス

ビルトイン RESTful サービスは、次の領域で改善されました。

URL エンコーディング	REST サービスの URL エンコーディングが改善されました。現在、RFC-3986 with percent encoding (section 2.1) の URI 仕様に準拠しています。 詳細については、下位互換性を参照してください。
レコードメタデータ	Select 操作が改善され、新しい <code>includeMetadata</code> パラメーターを使用してレコードメタデータが返されるようになりました。システムメタデータは、作成者、作成時間、更新者、および更新時間メタデータフィールドを持つ <code>filter</code> および <code>sort</code> パラメーターで使えるようになりました。結果として、次のようになります。 • <code>includeMeta</code> は廃止され、<code>includeMetamodel</code> パラメーターに置き換えられました。 • <code>includeTechnicals</code> パラメーターは廃止され、「system」値を持つ <code>includeMetadata</code> パラメーターに置き換えられました。

	• <code>checkNotChangedSinceLastUpdateDate</code> パラメーターは廃止され、Delete 操作の <code>checkNotChangedSinceLastUpdateTime</code> パラメーターに置き換えられました。 Update 操作と Delete 操作は、拡張形式と同じメカニズムを持つ JSON コンパクト形式の楽観ロックをサポートするようになりました。詳細については、データモデリングのレコードメタデータ、<code>includeMetadata</code> パラメーター、<code>includeMetamodel</code> パラメーター、および <code>checkNotChangedSinceLastUpdateTime</code> パラメーターを参照してください。
新しい一括更新および Delete 操作	これらの操作は、XPath 述語式および/または XPath 式を含むことができる公開されたテーブルビューを組み合わせることができます。更新操作は、一致する各レコードに 1 つ以上の変更を適用します。 詳細については、Insert 操作および Delete 操作を参照してください。
ヘルス REST ビルトイン	新しいヘルスカテゴリオペレーションは、EBX® サーバーの監視に役立ちます。 詳細については、ヘルスオペレーションを参照してください。
軽量 JSON 拡張形式	複数回の列挙フィールドのセレクター URL は、各オカレンスではなく、フィールドの下に移動されます。この変更により、Select 操作の JSON コンテンツが軽くなります。

ビルトイン SOAP 操作

ビルトイン SOAP 操作は、次の領域で改善されました。

レコードメタデータ

Select 操作が改善され、新しい

`includesMetadata` パラメーターを使用してレコードメタデータが返されるようになりました。システムメタデータは、作成者、作成時間、更新者、および更新時間のフィールドを持つ述語パラメーターで使用できるようになりました。結果として、

`includesTechnicalData` は廃止され、「system」値を持つ `includesMetadata` パラメーターに置き換えられました。

Update 操作が改善され、フィールドが無視された場合でも、Select 操作によって返されたすべてのデータコンテンツが取得されるようになりました。たとえば、`ebx-`

`metadata/system/update_time` を除いて、関連付け値とレコードメタデータを取得して、[楽観ロック](#)機能を確実に成功させることができます。

詳細については、`includesMetadata` パラメーター、Update 操作、レコードメタデータを参照してください。

EBX® メイン構成ファイル

メイン構成ファイルは、次のトピックで改善されました。

JNDI を使用しない SMTP の構成

Java Naming and Directory Interface (JNDI) を無視する場合、メール送信のタイムアウトをメイン構成ファイルから構成できるようになりました。

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。

EBX® Container Edition

Container Edition は、次のトピックで改善されました。

Java SE 開発キット	JDK は LTS バージョン 17 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。
SMTP タイムアウトの構成	メール送信のタイムアウトを構成できるようになりました。 詳細については、 メール接続 を参照してください。

このリリースの制限事項

階層の制限事項

新しい依存関係の許可されたディメンションには、次のことが適用されます。

- リストフィールドは、ネストされた列の依存関係内では許可されていません。
- 日付リストフィールドは時間ディメンションフィールドとして使用できません。

クエリの制限事項

これらの機能のパフォーマンスはまだ最適化されていません。

- 参照されるテーブルの主キーの外側のフィールドがこのパターンに含まれている場合、カスタム表示パターンを持つ外部キーフィールドの並べ替えを使用したページ化されたビュー (次/前のページ) のナビゲーション。
- 文字列ベースの操作を使用した外部キーフィールドでの高度な検索フィルタリング (例：開始、含む、一致など)。
- リンクされたフィールドでの並べ替えを使用した、ページ化されたビュー (次/前のページ) のナビゲーション。

次のケースは、Lucene インデックスに依存していません。このため、2 つの結果が生じます。1 つ目は、パフォーマンスが最適化されていないことです。2 つ目は、クイック検索の動作が全文検索ではなく、「contains」のように動作することです。

- ローカル依存関係のない計算フィールド
- 継承されたフィールド
- 継承されたデータセットのすべてのフィールド
- テーブル履歴ビューのすべてのフィールド

インデックスの最適化の恩恵を受けていないより具体的なケースについては、[次のセクション](#)で説明しています。

値のラベル付けの制限事項

値のラベル付け機能を使用した検索と並べ替えには、[値のラベル付けセクション](#)で説明されているいくつかの制限事項があります。

データ検証の制限事項

- プロシージャ内のデータセットまたはテーブルを検証することはできません。

リンクフィールドの制限事項

- 列挙型であるノードをターゲットにすることは、まだ完全にはサポートされていません。現在、表示と検索には、列挙ラベルではなく、未加工の値のみが使用できます。この制限が解除されるまで、回避策は、ターゲットノードとリンクされたフィールド自体で同じ列挙を宣言することです。

バージョンアップグレード

このリリースノートで提供される情報に加えて、[6.1 Java API の変更](#)の章では、バージョン 6.0 のリリース以降に公開された新しい Java API の完全なリストを提供します。

コンパイルおよびビルド環境

コンパイルおよびビルド環境では、`ebx.jar` の代わりに、次の新しい開発 JAR ライブラリを使用する必要があります。

- `ebx-api.jar` - EBX® の Java API 用。
- `ebx-resources-api.jar` - サードパーティの Java API 用。

注意:

`ebx.jar` の Java 型は難読化されなくなりました。

注意:

実行時には、開発 JAR ライブラリを展開しないでください。引き続き `ebx.jar` ライブラリのみを使用してください。

サポートされる環境の変更

サポートされる環境の章が更新され、以下に詳述する必要なアップグレードが反映されました。

ブラウザのサポート	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 11 または 17 のいずれかです。Java 8 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) がサポートされています。
Oracle サポート	Oracle 12c 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 です。
PostgreSQL サポート	PostgreSQL 10 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 11 です。
Microsoft サポート	SQL Server 2012 SP4 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 2014 です。

既存の EBX® 5 環境の移行

既存の EBX® 5 環境から実行可能な EBX® 6 を入手するには、[専用の移行ガイド](#)に従ってください。

既存の EBX® 6 環境からのアップグレード

Lucene ライブラリのバージョンが 8.11.0 から 9.4.0 に上がり、インデックス形式が変更されました。その結果、インデックスを保持するディレクトリは、製品のアップグレード中に自動的に削除されます。したがって、インデックスの再構築では、各テーブルへの最初のアクセスで遅延が発生することが予想されます。検索動作にも若干の変更が見られます。

データセット作成オプションの削除

データセットの作成サービスで、[外部データモデル](#)からデータセットを作成するオプションが削除されました。データモデルアシスタントでファイルをインポートすることは引き続き可能です。

下位互換性

URL エンコーディング

REST サービスと GUI の URL エンコーディングが改善されました。現在、[RFC-3986 with percent encoding \(section 2.1\)](#) の URI 仕様に準拠しています。パーセントエンコーディングは、エンコーディングが必要な文字が含まれている可能性があるデータスペースキーまたは主キーが含まれている場合に、URL のパスおよびクエリパラメーターの値とキーをエンコードするために使用されます。たとえば、URL のクエリパラメーター値の「&」は、`A&B` の `A%26B` のようにエンコードされます。

スラッシュとバックスラッシュ文字をサポートするには、サーバー構成を確認する必要があります。

詳細については、[インストールに関する注意事項](#)を参照してください。

エンティティ名	このリリースでは、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するようになりました。テーブルの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でなければならないため、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。 詳細については、エンティティマッピングおよび <code>SchemaNode.getTableEntityName</code> を参照してください。
JSON 拡張形式の変更	セレクター URL は、Select 操作を使用して複数回出現する列挙型フィールドに移動されました。詳細については、軽量 JSON 拡張形式を参照してください。
REST Toolkit の例外処理	REST Toolkit の例外 JSON 形式が変更され、バージョン 6.0.x で使用されている形式と互換性がなくなりました。 詳細については、6.1.1 の概要を参照してください。
REST Toolkit での 10 進数のシリアル化	JSON シリアル化ライブラリがアップグレードされ、新しいバージョンでは <code>java.util.BigDecimal</code> が <code>JSON 番号</code> ではなく <code>JSON 文字列</code> にシリアル化されます。カスタムシリアライザーを使用して、この動作をオーバーライドできます。
クエリ内の句の最大数	Lucene ライブラリのアップグレードにより、クエリで許可される句の最大数を計算する方法が変更されました。以前は、非常に多数の句を使用することができました。こ

	これらのクエリは例外をスローするようになりました。条件は簡単に満たされません。これが発生した場合、レーベンシュタイン検索戦略によることが考えられます。これは、戦略をカスタマイズしてあいまいさを減らすことで回避できます。たとえば、不変のプレフィックスを追加するか、最大編集距離を減らします。
ローリングログファイル	6.0.6 から 6.0.10 (付属) では、ローリングファイル名が <code>xxxx.log.yyyy-MM-dd</code> (または <code>xxxx.log.yyyy-MM-dd-i</code>) から <code>xxxx.MM-dd-yyyy.log</code> (または <code>xxxx.MM-dd-yyyy-i.log</code>) に変更されました。デフォルトでは、前者の形式 (<code>xxxx.log.yyyy-MM-dd</code> または <code>xxxx.log.yyyy-MM-dd-i</code>) が使用されます。ローリング形式は、プロパティ <code>ebx.log4j.appender.ebxFile.rollingFormat</code> を設定することにより、<code>ebx.properties</code> ファイルで指定できます。このプロパティが指定されている場合、ファイル名は <code>xxxxxx.rollingFormat.log</code> の形式になり、ファイル名の末尾に拡張子が追加されません。6.0.6 以降と同じファイル名形式を構成するには、値を <code>%d{yyyy-MM-dd}-%i</code> にする必要があります。このプロパティは、Java SimpleDateFormat と互換性のある日付/時刻パターンおよび/または整数カウンターを表す <code>%i</code> の両方を受け入れます。整数カウンターは、バックアップのしきい値と組み合わせて使用され、現在のファイルサイズがしきい値に達すると増分され、新しいファイルが作成されます。 6.0.6 以降、 <code>ebx.log4j.appender.ebxFile.backup.Threshold</code> プロパティは、毎日のロールオーバーのバツ

クアッパファイルの最大サイズ (メガバイト単位、バックアップファイルの数ではなく) を定義します。

6.1.2 既知の問題

ユーザーインターフェイス

- **CP-24388** タイルビューが、コンボボックスの高度なセレクトターとして使用できません。

コンボボックスの高度なセレクトターとしてタイルビューが指定されている場合、タイルビューのカードに ボタンが表示されません。

6.1.2 修正済みの問題

コアエンジン

- **CP-19208** `ebx.log4j.appender.ebxFile.backup.Threshold` プロパティのドキュメントを更新する必要があります。

ドキュメントでは、`ebx.log4j.appender.ebxFile.backup.Threshold` が保持されるバックアップファイルの数を定義すると指定しています。ただし、このプロパティは実際にはログファイルのサイズ制限を設定します。

- **CP-21633** 列挙ノードをターゲットとするリンクフィールドの動作を更新する必要があります。

リンクされたフィールドが列挙ノードをターゲットにしている場合、期待される動作は、ターゲットのテーブルと同じように表示および検索可能です。現在、列挙ラベルの代わりに未加工の値が使用されます。

- **CP-22439** `SQLChecker` は、エラーに関する詳細なコンテキストを提供する必要があります。

サポートケースでのデバッグを容易にするために、スタックトレースをコンパイルレポートに追加する必要があります。

- **CP-23515** `UIHttpManagerComponent` は、述語に改行がある場合、誤った結果を返します。
- **CP-25570** クラスターセンターでは子テーブルの除外が無視されます。

- **CP-25588** 削除されたデータセットへの外部キーを定義するテーブルの履歴にアクセスすると、エラーが発生します。
- **CP-25697** 名前戦略による検索と日本語を改善する必要があります。

漢字やひらがなでは適切な結果が返されない場合があります。

- **CP-25805** 間違ったフィルター関数名を使用すると、REST ビルトイン選択がステータスコード 200 を返します。
- **CP-25874** フィルタリングされた再帰階層で検索するときにパフォーマンスの問題が発生します。
- **CP-25910** 最も内側のサブクエリが最も外側のテーブルを参照する 2 つのネストされたサブクエリを含むクエリは、間違った結果を返すか失敗します。
- **CP-25935** クエリはセッションのプログラムによるアクセスルールを適用する必要があります。
- **CP-25943** 特定の条件下で、インデックスのコミットされた状態が失われる可能性があります。
- **CP-25945** ロールバック時に、リポジトリ内で何も変更されていない場合、ローカルトランザクションコンテキストがロールバックされません。
- **CP-25981** ロールバックにより、以前にコミットされたトランザクションのインデックスリーダーが閉じる可能性があります。
- **CP-26036** 長い文字列値に対して名前検索戦略を使用すると、インデックス作成が失敗します。
- **CP-26061** テーブルリビジョンとインデックスの共有状態が矛盾しています。
- **CP-26076** 同じトランザクション内でレコードが作成および削除された場合、最後のインデックスサーチャーが更新されないことがあります。

データサービス

- **CP-25741** REST ビルトインの Select が、「述語にパスがありません」というメッセージとともに HTTP ステータスコード 400 を返します。
- **CP-25968** フォーマットが正しくない場合、HTTP ステータスコード 500 が返されます。

ユーザーが入力した JSON 本文のフィールドでフォーマットに誤りがある場合、HTTP コード 500 が返されます。

データワークフロー

- **CP-25782** プロセスビューに正しい状態が表示されません。

プロセスビューのグラフに、サブワークフロー呼び出しステップを持つワークフローの正しい状態が表示されません。

- **CP-25996** ワークフローを終了しようとする、 `NullPointerException` が発生します。

ドキュメント

- **CP-26063** `ibmjsse.jar` ライブラリのドキュメントに含まれるダウンロードリンクが正しくありません。

階層

- **CP-23280** 100 レベルを超える再帰階層を拡張すると失敗します。
- **CP-24211** 不規則な階層で外部キー未設定ノードを展開すると、エラーメッセージが表示されます。
- **CP-24958** レベルのテーブルが別のデータセットにある場合、階層レベルフィルターが表示されません。

チームアップ

- **CP-25808** プロファイルのチームアップコンボボックスにビルトインの管理者ロールが表示されません。

ユーザーインターフェイス

- **CP-25593** 関連付けられたデータセットから非表示のテーブルを表示しようとしても、そのテーブルが表示されません。
- **CP-25795** 別のユーザーサービスにリダイレクトし、終了ポリシーが定義されている場合、空白のページが表示されます。
- **CP-25832** 階層内で、「検索」アクションが強調表示された項目までスクロールしません。
- **CP-25834** テーブルにレコード権限ルールが定義されている場合、ユーザーは フィールドにプロファイルを複数回追加できます。
- **CP-25836** データセットの ページでエレメントが正しく配置されていません。
- **CP-25841** アバター内のテキストが正しく中央に配置されません。

- **CP-25885** テーブル間をすばやく切り替えると権限が拒否されます。

この問題は、ユーザーがビルトインの管理者の場合ではなく、ロールが使用されている場合に発生します。

- **CP-25953** カスタムログイン URL を定義すると、無限リダイレクトが発生します。
- **CP-26041** ユーザーがドキュメントの編集に戻ると、ビュー作成ページの フィールドが誤って表示されます。

ユーザーと権限

- **CP-24692** 読み取り専用アクセス権を持つユーザーが タブの グループに値を保存すると、エラーが発生します。
- **CP-26077** データモデルのインポートにレコード権限が構成されている場合、メモリの問題が発生します。

6.1.1 修正済みの問題

クラウド

- **CP-23527** データワークフローの下のテーブルの階層ビューでノードラベルが正しく表示されません。

Container Edition

- **CP-25175** TIBCO EBX® Container Edition インストーラーで、AMD64 プラットフォームイメージ以外のイメージがビルドされる場合があります。

ARM64 プラットフォームでインストーラーを実行すると、生成されるイメージは ARM64 になります。ZIP パッケージに AMD64 サフィックスを含めることはできません。

コアエンジン

- **CP-16736** 無関係な列での並べ替えを無効にする検索戦略を修正しました。

「プロセスビュー」、「アクション」、または説明値などの列での並べ替えは役に立ちません。

- **CP-16856** バイナリデータのコピー時にデータベース間の移行が失敗する場合があります。

- **CP-18406** グループ内の複数値フィールドを検索すると、検索結果が正しく強調表示されません。
- **CP-18408** ターゲットを絞った検索でキーワードが正しく強調表示されません。
- **CP-18435** 相互再帰主キーを持つスキーマを公開中に `StackOverflowError` がスローされます。
- **CP-18487** フィールドが複合グループの下にある場合、`Result.getHighlights()` API は空の `RecordHighlights` を返します。
- **CP-18524** 検索結果が、フィールドで指定された条件を満たす場合に正しく強調表示されません。
- **CP-19033** 「削除されたデータスペース/スナップショット」管理テーブル内のレコードを削除できないようにする必要があります。

これを行うと、削除されたデータスペースの履歴を GUI から削除できなくなります。

- **CP-22630** タグを含む値を強調表示すると例外がスローされます。
- **CP-23388** 大文字と小文字を区別しない「単語全体を含む」フィルターが正しく機能しません。

単語に特殊文字が含まれている場合、「単語全体を含む」フィルターを使用してフィルターされたビューを適用すると、間違った結果が返されます。

- **CP-23400** 列挙型フィールドで「%」文字で終わるキーワードを検索すると、結果が強調表示されません。
- **CP-23928** アーカイブのインポート時に表示されるエラーメッセージが不明瞭です。
- **CP-25314** `LinkedField` プロパティは、管理者以外のユーザーには表示されません。
- **CP-25315** フィールドタイプを Date から DateTime に変換するときにエラーが発生します。
- **CP-25336** 再帰式を含む計算ルールを評価するとエラーが発生します。
- **CP-25383** `CnyCanopyProducerCallable` の最後のレコードが処理されません。
- **CP-25420** 子データスペースの作成時に `NullPointerException` が発生します。
- **CP-25460** `ServiceKey` クラスの初期化が原因で、起動時にデッドロックが発生します。
- **CP-25505** `ProcedureContext.setTriggerActivation(false)` が期待どおりに動作しません。

プロシージャコンテキストを使用して同じテーブル (親の子) に別のレコードを作成する場合、`setTriggerActivation(false)` メソッドはトリガーを無効にせず、子レコードに対して、`handleBeforeCreate` メソッドが呼び出されます。

- **CP-25640** バッファサイズ管理が十分に堅牢ではありません。

サイズ変更方法の改善、負のサイズ割り当ての防止、境界の場合に明確な例外を追加することが期待されます。

- **CP-25641** `com.onwbp.adaptation.ReadContext.get(SchemaNode)` は入力をチェックする必要があります。

ノードの有効性をアップストリームでチェックし、必要に応じて例外を発生させる必要があります。

- **CP-25696** NGram 検索戦略を使用してひらがなおよび漢字の文字セットを検索すると、予想したレコードが返されません。
- **CP-25762** パラメーターを含む XPath フィルターで `ClassCastException` がスローされます。

データモデリング

- **CP-21989** ユーザーがテキスト検索演算子を含む検証ルールにアクセスすると、空のワークスペースが表示されます。

ユーザーがテキスト検索演算子を含む検証ルールにアクセスするか、このルールに対して「条件の変更」サービスを実行すると、空のワークスペースが表示されます。

データサービス

- **CP-22805** レコードの選択に使用される大きな述語式で `StackOverflow` がスローされます。
- **CP-25418** REST API を使用する場合、値関数フィールドが存在するとビルトインの Insert 操作が正常に完了しません。
- **CP-25797** 本文に 1000 件を超えるレコードが含まれている場合、REST API のビルトイン Insert 操作は 400 Bad Request コードを返します。

データサービス -REST Toolkit

- **CP-25609** `swagger-ui.css` ファイルが見つかりません。

REST ドキュメントにアクセスする場合、Swagger UI が使用できません。

データ検証

- **CP-25650** `ValueContext.getValue` が絶対パスで呼び出された場合、`TableRefFilter` から `PathAccessException` がスローされます。

ドキュメント

- **CP-25491** アドオンモジュールの削除に関連するドキュメントが不完全です。

「TIBCO EBX® アドオンの展開と登録」の章の「アドオンモジュールの削除」セクションに手順がありません。

階層

- **CP-17172** 不完全な 3 項結合リンクディメンションを持つ階層ビューを適用すると、ノードが表示されません。
- **CP-21464** 階層作成画面で「ラベル」フィールドが正しく表示されません。

階層作成画面では、「ラベル」フィールドに対象テーブルのみが表示されます。

- **CP-21607** 「アクティベーション」テーブルの階層ビューのノードに間違ったラベルが表示されます。
- **CP-21751** 階層ビューでレコードを移動した後も、レコードが選択されたままになります。
- **CP-23604** レベル述語が設定されている場合、整理された再帰階層でパフォーマンスの問題が発生します。
- **CP-24795** 階層が整理モードの場合、リクエストのパフォーマンスの問題が発生します。
- **CP-24964** Java 列挙ラベルを階層ビューでディメンションとして使用すると、そのラベルが表示されません。
- **CP-25417** 検索アクションの再帰階層にパフォーマンスの問題があります。
- **CP-25456** 階層の表示中にディメンションのイントロスペクションが原因でパフォーマンスの問題が発生します。

ステージング

- **CP-25921** ステージングドメインの「管理」カテゴリに、アルファベット順に表示されていないオプションが 2 つあります。

ユーザーインターフェイス

- **CP-1371** メンバーサイクルが存在する場合、「階層内検索」サービスが正しく動作しません。
- **CP-17443** カスタムレイアウトを定義した後、タブにエラーメッセージが表示されません。
- **CP-24144** クイック検索を適用すると、子データセットの階層ビューの表示に一貫性がなくなります。
- **CP-25262** 別のユーザーサービスにリダイレクトすると、空白のページが表示されます。
- **CP-25589** ワークフローが終了するか、ユーザーサービスでキャンセルされた後、「ユーザーがログアウトしました」ページが表示されます。

これは、5.9 から 6.0.13 または 6.1.0 にアップグレードした後に発生します。

- **CP-25703** 列挙型フィールドを表示すると、カスタムフォームのプレビューでエラーが発生します。

これは、静的列挙セットまたは未定義のラベルを持つ列挙値のプロパティが表示されるときに発生します。

6.1.0 修正済みの問題

Container Edition

- **CP-24408** 停止した EBX Container Edition コンテナを再起動すると失敗する場合があります。

コアエンジン

- **CP-18256** 外部キーで大規模なテーブルをソートすると、非常に時間がかかります。

大量のデータを含むテーブルを並べ替えると、予想よりも時間がかかります。

- **CP-18919** ファイルサイズがゼロの場合、EBX ログが削除されません。
- **CP-20968** 「最後の変更を適用」サービスを適用すると、読み込み画面が長時間表示されます。

これは、100,000 件のレコードを含む子データセットでサービスを実行した場合に発生します。

- **CP-21133** Jaro-Winkler 戦略がパラメーターを受け入れなくする必要があります。
- **CP-23002** 割り当てられた検索戦略に Jaro-Winkler 検索を適用すると、

`NullPointerException` が発生します。

グローバル検索を実行し、フィールドの割り当て済み検索戦略が [デフォルトの検索テンプレート] に設定され、Jaro-Winkler 戦略を使用する場合、例外が発生します。

- **CP-23572** 「present in」 演算子が、HTML データ型の列挙フィールドで使用できません。
- **CP-24178** `AdaptationHome.getTerminationDate()` が、開いているデータスペースで null を返しません。
- **CP-24189** 複合データ型の関係とクラスターフィールドが一致すると、クラスタリングでエラーが発生します。

照合で関係を使用し、クラスターフィールドが複合データ型である場合、クラスタリングはエラーをスローします。

- **CP-24199** データモデルのコンパイル中はデータセットにアクセスできません。
- **CP-24257** ロギングカテゴリのレベルが変数によって定義されると、`NullPointerException` が発生します。
- **CP-24290** パースペクティブが複製されている場合、パースペクティブアーカイブのインポート中にエラーが発生します。

複製されたパースペクティブを含むアーカイブをインポートすると、エラーが発生します。

- **CP-24333** アドオンライセンススレッドがサーバーのシャットダウンをブロックする可能性があります。
- **CP-24345** 重大なエラーで作成されたレコード、テーブル、またはデータセットをエクスポートすると `NullPointerException` が発生します。

重大なエラーの状態で作成されたレコード、テーブル、またはデータセットをエクスポートすると、`NullPointerException` が発生します。

- **CP-24346** ウィジェットで `SchemaFacetTableRef.getLinkedRecords()` を呼び出すと `ClassCastException` が発生します。

ウィジェットで `SchemaFacetTableRef.getLinkedRecords()` を呼び出すと、`ClassCastException` が発生します。

- **CP-24379** 検証レポートのクリーニングステップでリポジトリの削除操作が失敗します。

検証レポートのクリーニングステップで、リポジトリの削除操作が失敗します。

- **CP-24405** 外部キーの `UIDropDownList` がエレメントを表示しません。
- **CP-24449** メタデータが実際には存在しないテーブルを参照している場合、EBX 6 への移行が失敗します。

メタデータが実際には存在しないテーブルを参照している場合、EBX 6 への移行が失敗します。

- **CP-24579** 子データスペースでインデックスを再構築するとエラーが発生します。
- **CP-24627** 新しい列を追加すると、履歴テーブルの既存のレコードにデフォルト値が設定されます。

履歴化されたテーブルに新しい列を追加すると、そのデフォルト値が履歴内の既存のレコードに追加されます。

- **CP-24634** エラーにより、テーブルの内容にアクセスできません。

これには、「`java.lang.IllegalStateException: Cf0ccurrence with ref (...,...) could not fetched from table zza7[<tableName> of content type=VALUES in dataset <datasetName> in dataspace DataspaceName]: record have been deleted a transaction`」のようなメッセージのエラーが含まれます。

- **CP-24657** レコードの更新中に「`java.lang.IllegalStateException: This index is read-only`」というエラーが発生します。

レコードの更新時に `IllegalStateException` エラーが発生することがあります。

- **CP-24706** 最大値セットフィールドをより大きな値に編集しても、新しいレコードを作成するときの自動インクリメントには影響しません。
- **CP-24733** クエリパラメーターを準備するときに、サブクエリが正しく識別されません。
- **CP-24763** `com.orchestranetworks.service.directory.EncryptPassword` クラスがありません。

メイン構成ファイルのドキュメントに記載されている

`com.orchestranetworks.service.directory.EncryptPassword` クラスが製品パッケージにありません。

- **CP-24895** コミットされていないトランザクションからの値により、テーブルサイズの見積もりが破損する可能性があります。
- **CP-24917** > > のテーブルに高度な検索を適用すると、エラーが発生します。<emphasis></emphasis>
- **CP-24944** `SchemaDocumentation` を拡張すると、表示されたリンクをクリックできなくなります。
- **CP-24997** デフォルトテンプレートの検索方法がありません。

複数の検索戦略が定義された外部キーノードを含むデータモデルをコンパイルすると、例外がスローされる場合があります。

- **CP-24999** `LockObtainFailedException`: この仮想マシンが保持するロック エラーが発生します。

インデックスの再構築中に問題が発生した場合、再構築の実行に使用された `IndexWriter` が適切に閉じられず、スロットがロックされたままになり、後で再利用できなくなります。

- **CP-25032** コミット時のインデックス更新中に、ログエラーが発生する可能性があります。

これにより、コミット後にキャッシュされたインデックスとテーブルコンテンツ間の同期が中断される可能性があります。

- **CP-25233** 一括更新でリストを含むレコードを複製すると、主キー違反が発生します。
- **CP-25270** レコードの作成時に `LxiDuplicateKeyException` が発生します。
- **CP-25405** `XPxParametrizedLiteral` 値が内部フォーマットの場合、`toXPathExpression` 内で `ClassCastException` が発生する可能性があります。

データモデリング

- **CP-21255** データモデルに循環依存関係がある場合、データモデルの `SqlChecker` エラー (エラーのある 1 つのデータモデルを含む) が発生します。
- **CP-22225** テーブル参照の非推奨のプロパティが使用できません。

非推奨のプロパティ `displayKey` または `labelPaths` を使用してテーブル参照を宣言すると、データモデルがコンパイルされません。

データサービス

- **CP-2130** REST ビルトインの `Select` が、複数オカレンスグループを持つテーブルからデータを返しませんが、複数オカレンスグループが複数オカレンスグループ内にネストされている場合、REST 要求は内部グループからデータを返しませんが、

複数オカレンスグループが複数オカレンスグループ内にネストされている場合、REST 要求は内部グループからデータを返しませんが、

- **CP-17619** SOAP ヘッダーのオプションパラメーターは、プレフィックスなしのデフォルトの名前空間を使用すると機能しません。

SOAP 操作を呼び出すときに、プレフィックスなしでデフォルトの名前空間を使用すると、ヘッダーにパラメーターを追加しても機能しません。

- **CP-20709** ContentHolderForInput は、「content」という名前のオブジェクト属性を自動的に削除します。

`ContentHolder` をデータ入力に使用すると、「content」という名前のオブジェクト属性がデータモデルの一部であっても自動的に削除されます。

- **CP-21814** `:count` 構文を使用した関連付けフィールドでの REST カウント操作は、404 を返します。
- **CP-21830** ドキュメントには、REST ビルトインの JSON コンパクト形式を使用したプロシージャコンテキストの制限について言及されていません。
- **CP-24606** リネージ WSDL 構成は、UI で選択されたコンテキストを考慮しません。

リネージ WSDL 構成では、選択された UI コンテキストが考慮されます。

- **CP-24653** REST ビルトインの一時的な操作は、関連付けフィールドと値関数ノードを無視する必要があります。
- **CP-24664** 一部の有効化されたパラメーターを使用した選択操作で、SOAP 本文の検証エラーが発生します。

選択操作で `exportCredentials` が `true` 値で有効になっている場合、XML 本文の応答が無効です。

- **CP-24673** SOAP 操作を使用してブランチが存在しない場合に、NPE が発生します。

SOAP 操作の使用時に、定義されたブランチが存在しない場合、`NullPointerException` が発生します。

- **CP-24722** 関連付けフィールドは、REST ビルトインの Import/Update 操作を使用して無視する必要があります。

REST ビルトインの Import/Update 操作を使用する場合は、関連付けフィールドを無視します。

- **CP-24962** XML のインポート時に `PathAccessException` が発生します。

グループのリストの直後にある空のグループがレコードに含まれている場合、例外によりインポートが失敗します。

- **CP-25013** REST ビルトインのセレクトフィールドは、読み取り専用フィールドに存在することはできません。

データ検証

- **CP-7461** 「dd/MMM/yyyy」のような形式で日付を入力することができません。

フォーマットポリシーで月の 3 文字フォーマットが指定されている場合、日付を入力することはできません。

- **CP-18807** ユーザーサービスで、ページの再表示時に例外がスローされる場合があります。

User Service API を使用して、レコードを作成するためのフォームをカスタマイズできます。レコードに外部キーが含まれている場合、ユーザーサービスが再表示されたときに、予期しない例外がスローされることがあります。

- **CP-21584** `UserService` で、
`ConstraintOnTableWithRecordLevelCheck#checkRecord` にエラーメッセージが追加されている場合、レコードフォームの送信がブロックされません。
- **CP-23502** レコード検証レポートが、レコードフォームに表示されるタブごとに計算されません。

レコードのフォームのヘッダーには、関連するレコードの最新の検証レポートが表示されます。フォームの各タブには、独自のヘッダーがあります。結果として、検証レポートは、フォームに表示されるタブの数だけ計算されます。テーブルに多くの制約が含まれている場合、これによりパフォーマンスの問題が発生する可能性があります。

- **CP-24171** `CalciteException` テーブル参照ノードを使用してテーブル参照フィルターでテーブルを検証するときに、テーブルが見つかりません。

テーブル参照フィルターが、別のデータセットにあるテーブルへのテーブル参照であるノードを使用すると、例外がスローされます。

- **CP-24192** テーブル参照フィルター検証のために calcite によって生成されたコードで `NullPointerException` が発生します。

テーブル参照フィルターの検証のために Calcite によって生成されたコードで `NullPointerException` が発生します。

- **CP-24336** `xs:pattern` ファセットが日付の XML 文字列表現に適用されません。

`xs:pattern` ファセットは、XML 文字列表現ではなく、Java Date オブジェクトの文字列表現をチェックします。

- **CP-24483** 永続化された検証レポートの更新中に、`NullPointerException` が発生します。

永続化された検証レポートの更新中に `NullPointerException` が発生します。

- **CP-24526** 永続化された検証レポートにアクセスするとパフォーマンスが低下します。

永続化された検証レポートにアクセスすると、パフォーマンスが影響を受けます。

- **CP-24777** 機能データセットの削除が中止された後に検証データスペースにアクセスすると、`IllegalStateException` が発生します。
- **CP-24904** テーブル参照の検証が `ClassCastException` で失敗します。

これは、`PvrActionForTableRef` が間違った `FK_AS_STRING` パラメーターでクエリプランを生成した場合に発生します。

- **CP-24923** コンテキストテーブル参照フィルターの検証中に `ClassCastException` が発生します。
- **CP-24933** 多値フィールドを持つコンテキストテーブル参照フィルターが、正しくない結果を返します。
- **CP-25019** `ValidationReportItemSubjectForTable.getRecords()` の結果に `null` 値が含まれています。

データビュー - 階層

- **CP-16830** 階層サイクルの検出は、再帰的な 3 項依存関係では機能しません。
- **CP-17169** 特定のプロパティ「再帰的な孤立リーフを整理」を実行できません。

親のない再帰メンバーがある場合、孤立リーフを整理するオプションを実行できません。

- **CP-17204** ノードレベルのアイコンとそのテキストが、階層ビューの作成画面で整列されていません。

階層ビューの作成画面で、ノードレベルのアイコンとテキストが整列しません。

- **CP-18439** マルチリンク階層で検索が見つかりません。

マルチリンク階層では検索結果が見つかりません。

- **CP-20401** 一部の子ノードが階層ビューで重複しています。

階層ビューが適用されると、一部の子ノードが 2 回表示されます。

- **CP-20448** 列挙可能なレベルフィルターが階層に対して機能しません。

ユーザーが階層ビューを作成してフィルターを設定すると、誤った結果が表示されます。

- **CP-21838** 階層の「すべて展開」が 2000 の表示メンバーに制限されなくなりました。

階層展開は、2000 メンバーの表示に制限されなくなりました。

- **CP-21840** 3 項再帰階層での検索が失敗します。

3 項再帰ディメンション階層の検索ツールが正しく機能しません。

- **CP-21882** 日付/列挙フィールドに基づく階層ビューの最後のページで、ボタンがアクティブになります。

日付または列挙型のデータ型を持つフィールドに基づく階層の最後のページに移動した後、ナビゲーションボタンがアクティブなままになります。

- **CP-21892** `member.isLooping()` は、主にワークフローで `stackOverflow` を引き起こす可能性があります。

`member.isLooping()` を使用すると、主にワークフローで `StackOverflowError` が発生する可能性があります。

- **CP-21900** 階層ビューの作成時に、レベルの子ノードが完全には表示されません。

ユーザーが複数のレベルを持つ階層ビューを作成して適用すると、その子ノードが完全には表示されません。

- **CP-22403** ユーザーが階層ビューで再帰レコードを複数回選択/選択解除すると、選択されたレコードの数がツールバーに正しく表示されません。

再帰レコードを複数回選択および選択解除すると、選択されたレコードの数が階層ビューのツールバーに正しく表示されません。

- **CP-22429** 整理モードを有効にした後、順序付けフィールドモデルを持つ 3 項の階層ビューのノードが切り替えられます。

3 項テーブル階層ビューで整理モードを有効にした後、ノードの順序が入れ替わります。

- **CP-22483** 階層ビューで、フィルターで除外する必要があるにもかかわらず、一部のレコードが表示されます。

一部のレコードは、フィルターで除外する必要がある場合に階層ビューに表示されます。

- **CP-22881** 階層ブランチの選択は、不規則な階層では機能しません。

階層ブランチの選択は、不規則な階層では機能しません。

- **CP-23010** 予期しないノードが階層ビューに表示されます。

予期しないノードが階層ビューに表示され、選択するとエラーが発生します。

- **CP-24281** 「再帰的な孤立リーフを整理」オプションを再実装します。

オプションを再実装する必要があります。<emphasis role="bold"></emphasis>

- **CP-24443** 「整理モードの有効化/無効化」が正しく設定されていません。

整理モードオプションが正しく設定されていません。

- **CP-24558** 列挙可能なフィールドではページネーションが機能しません。
- **CP-24646** 非表示のレコードを階層表示に使用するべきではありません。
- **CP-24675** サイクルメンバーが、3 項依存階層に残ります。

サイクルメンバーを含む 3 項依存関係階層では、サイクルメンバーは検索後もそのまま残ります。有効なブランチにありません。

- **CP-24746** 階層 REST API 孤立メンバーが適切に管理されていません。

REST API では、階層の孤立メンバーが適切に管理されません。

データワークフロー

- **CP-19283** ランチャーのラベルを更新しても、アクティベーションレコードのドキュメントが更新されません。

ランチャーアクティベーションラベルは、そのランチャーのラベルから継承するか、カスタマイズすることができます。このラベルが継承されている場合、ランチャーのラベルが変更されても更新されず、ユーザーはこれに気付かない可能性があります。新しい値を適用するために再度保存する必要があることをユーザーに通知する情報やヒントは表示されません。

- **CP-22631** ワークフローダイアグラムエディターの アイコンを修正します。

ワークフローダイアグラムエディターの アイコンが正しく表示されません。<emphasis role="bold"></emphasis>

- **CP-24148** ワークフローステップでプロファイルフィールドをクリアする方法がありません。

ワークフローステップでプロファイルフィールドをクリアする方法がありません。

- **CP-24578** ワークフローの複雑なオブジェクトがデフォルトで展開される必要があります。
- **CP-24724** リンクを別のステップにドラッグして確認をキャンセルすると、リンクの終了点が変わります。
- **CP-24769** レコードフィールド値の更新スクリプトタスクの XPath パラメーターのドキュメントが正しくありません。
- **CP-25239** モデル構成のデータコンテキストマッピングは、デフォルトで開いている必要があります。

データスペースと履歴

- **CP-24934** データスペース比較アクション中に `NullPointerException` が発生します。

ドキュメント

- **CP-18443** セキュリティドキュメントの Java サンプルを修正します。

セキュリティのベストプラクティスに関するドキュメントの Java サンプルを更新する必要があります。

- **CP-25026** WebSphere AS 8.5 および 9.0 サポートの Java EE プラットフォームが正しくありません。

WebSphere Application Server 8.5 および 9.0 サポートの Java EE プラットフォームが正しくありません。

- **CP-25092** Tomcat のインストールに関する注意事項の Windows パスの二重引用符の使用法が正しくありません。

インフラストラクチャー

- **CP-24586** すべての `handleRepositoryStartup` メソッドは、EBX® へのアクセスを開く前に完了する必要があります。

クエリ

- **CP-24291** フィールドトリマーは、`COUNT()` を使用して `LogicalAggregate` への入力からすべてのフィールドを削除します。

フィールドトリミングステップ中に `ClassCastException` が発生します。

- **CP-24319** `RequestResult.getSize()` の結果が正しくなく、呼び出しごとに異なる可能性があります。

`RequestResult.getSize()` メソッドは、正しくない結果を返す可能性があり、呼び出しごとに異なります。

- **CP-24551** `RelNode` から SQL への変換が遅いため、リクエストログレベルを `DEBUG` に設定するとパフォーマンスの問題が発生します。

リクエストログレベルを `DEBUG` に設定すると、パフォーマンスの問題が発生します。

- **CP-24608** 非機密として設定されていても、複雑なノードの下にある場合、非表示ノードを外部キーフィルターで使用できません。
- **CP-24631** Lucene インデックスが利用できない場合にリソースフィールド (およびページネーション) で並べ替えると、例外がスローされます。

- **CP-24677** ヘルプフェーズ後のプランで、`HepRelVertex` が原因で、SQL クエリが `CannotPlanException` をスローします。

チームアップ

- **CP-24911** 階層ビューでチームアップフィールドを使用して検索を適用すると、エラーが発生します。

ユーザーインターフェイス

- **CP-2041** 関連付けを通じてレコードを作成しているときに、フィールドが予期せず自動的に入力されます。

関連付けを通じてレコードを作成する場合、作成フォームのフィールドは事前に入力されている必要がありますが、読み取り専用ではありません。

- **CP-17509** パースペクティブとブランチを選択する Web コンポーネント URL を使用すると、エラーが発生します。
- **CP-19856** バックアップされたユーザーを選択するページが表示されない場合があります。

カスタムログインを定義するときに、ユーザー選択ページのバックアップが表示されません。

- **CP-20087** ボタンのテキストとその矢印の間で整列の問題が発生します。

インポート画面とタスク作成モдалで、ボタンのテキストと矢印が水平方向に整列しません。<emphasis role="bold"></emphasis><emphasis role="bold"><emphasis>

- **CP-20157** データワークフローでテーブルを検索するときに、適用されたフィルターの数にツールバーが重なります。

ユーザーがタスクを実行して高度な検索を適用すると、適用されたフィルターの数にツールバーが重なって表示されます。

- **CP-21620** [WCAG 2.0 - 4.1.2](#). アクセシビリティチェックツール (NVDA) を使用する場合、ボタンと ボタンは追加情報を提供する必要があります。

追加情報には、そのロール、状態、および削除およびチェックされたエレメントなどの値関連情報と、それに対応する値を含める必要があります。

- **CP-21684** グループが非表示に設定されている場合でも、グループの下テーブルが表示されます。
- **CP-21753** ワークフローモデルの待機タスク作成画面で位置合わせの問題が発生します。

ワークフローモデルの作成画面で、フィールドのアイコンが、とのラジオボタンと垂直方向に整列されていません。

- **CP-22032** データスペース画面で列を選択すると、対応するデータセットが選択されません。

データスペース画面で、データセットの選択動作が正しく機能しません。

- **CP-22041** URL コンピューティングのドキュメントには、リバースプロキシを使用する場合のデータリネージ構成について記載する必要があります。

リバースプロキシを使用する場合 (特に https で)、データリネージプロパティを構成する必要があります。これは、URL コンピューティング構成で言及する必要があります。

- **CP-22275** カスタムウィジェットが定義されている場合、デフォルト値が失われます。

カスタムウィジェットを定義すると、特定の状況でデフォルト値が失われます。

- **CP-22295** でスクロールバーが表示されません。

を呼び出すと、ポップアップにスクロールバーが表示されません。

- **CP-22703** ペインの高さを上げると、表示の問題が発生します。

ペインのサイズを変更すると、階層ビューとタイルビューで表示の問題が発生します。

- **CP-22970** デフォルトのラベルがワークフローランチャーレコードにありません。
- **CP-23788** 「present in」 演算子を使用して高度な検索を適用すると、ドロップダウンリストフィールドの値が重複することがあります。

「present in」 演算子を使用して高度な検索を適用すると、ドロップダウンリストフィールドの値が重複します。

- **CP-24131** 強調表示されたバーがデータモデルとワークフローセクションで非表示になります。
- **CP-24668** パースペクティブのデータセットで宣言されたユーザーサービスの確認メッセージが表示されません。
- **CP-24771** UI でテーブルをロードするときに、下部の水平スクロールバーが表示されません。
- **CP-25000** エラーがクリックできません。
- **CP-25166** レコードのラベルが空の場合、階層リンクとそのドキュメントペインにプレースホルダーが表示される必要があります。

ユーザーと権限

- **CP-21226** FK が非表示の場合、リンクされたフィールドでフィルタリングすると例外が発生します。

外部キーが非表示の場合、リンクされたフィールドをフィルタリングすると例外がスローされます。

- **CP-24489** パースペクティブの所有者を設定するドロップダウンリストに、ビルトインの管理者ロールを持つプロファイルのみが表示されます。