TIBCO EBX®
Product Documentation

Version 5.9.20
August 2022

Important Information

Table of contents

User Guide

Reference Manual

Administration Guide

Security Guide

Developer Guide

User Guide

Introduction

CHAPTER 1

How TIBCO EBX works

This chapter contains the following topics:

1.1 Product overview

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

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

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

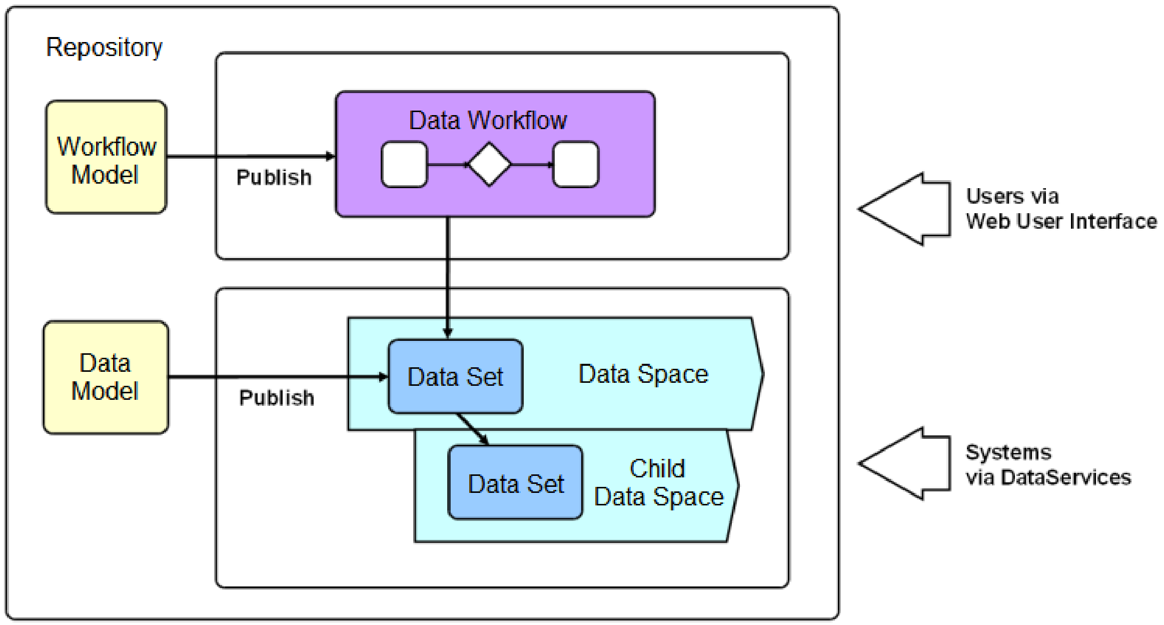
An MDM project using EBX starts with the creation of a *data model*. This is where tables, fields, links and business rules related to the master data are defined. Examples of modeled data include product catalogs, financial hierarchies, lists of suppliers or simple reference tables.

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

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

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

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

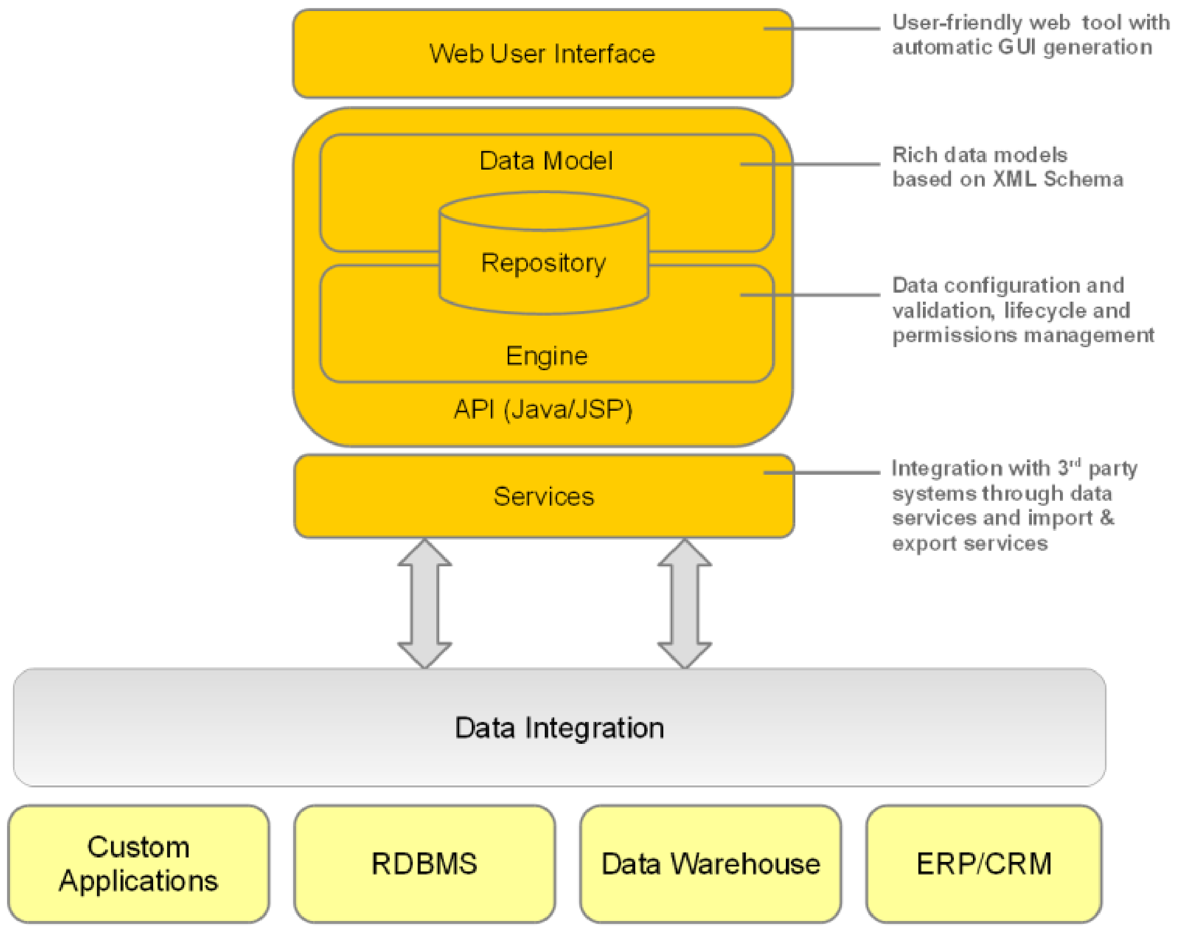


See also

- *Data modeling* [p 24]
- *Datasets* [p 26]
- *Dataspaces* [p 28]
- *Workflow modeling* [p 29]
- *Data workflows* [p 30]
- *Data services* [p 31]

1.2 EBX architecture

The following diagram illustrates the EBX architecture.


Web User Interface
User-friendly web tool with automatic GUI generation
Data Model
Repository
Engine
API (Java/JSP)
Rich data models based on XML Schema
Data configuration and validation, lifecycle and permissions management
Services
Integration with 3rd party systems through data services and import & export services
Data Integration
Custom Applications
RDBMS
Data Warehouse
ERP/CRM

CHAPTER **2**

Using the user interface

This chapter contains the following topics:

2.1 Overview

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

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

The advanced perspective is accessible by default.

See also*UI administration* *[p 383]*

2.2 Advanced perspective

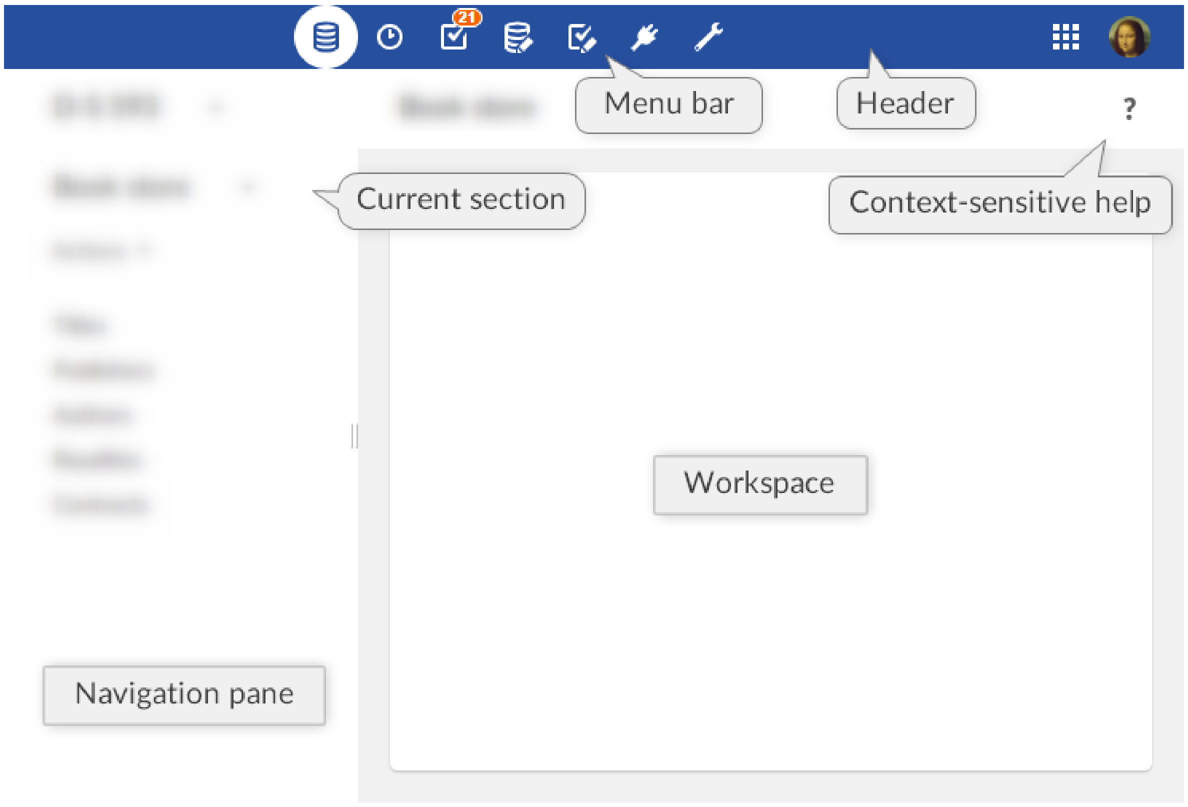
By default, the EBX advanced perspective is available to all users, but its access can be restricted to selected profiles. The view is separated into several general areas, referred to as the following in the documentation:

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

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

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

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



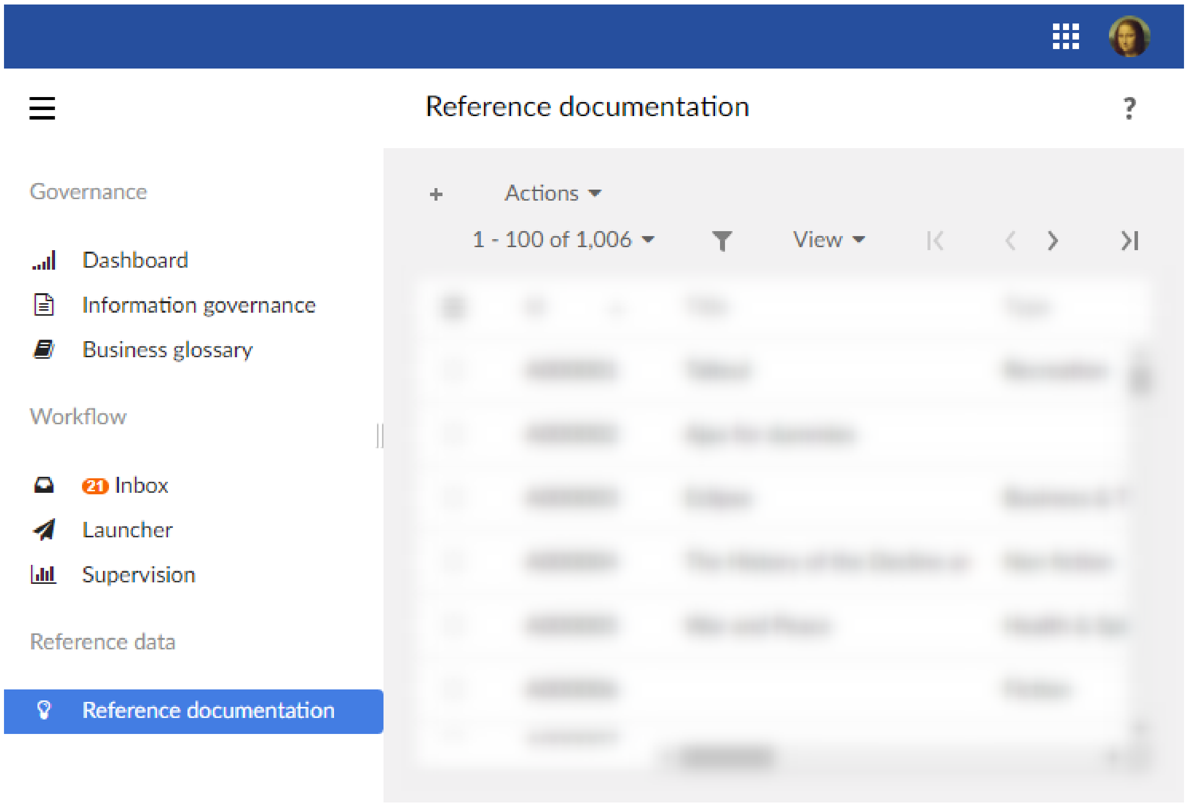
2.3 Perspectives

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

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

Perspectives are configured by authorized users. For more information on how to configure a perspective, see perspective administration [p 384].

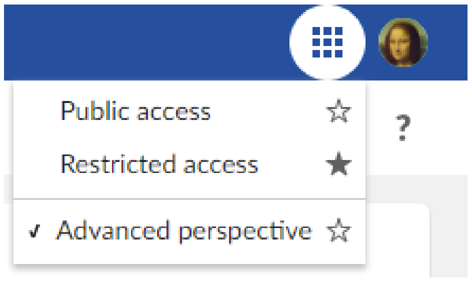
Example of a hierarchical menu:



Favorite perspectives

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

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



See also*Recommended perspectives* *[p 395]*

2.4 User pane

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

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

Attention

The logout button is located on the user pane.

Avatar

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

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

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

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

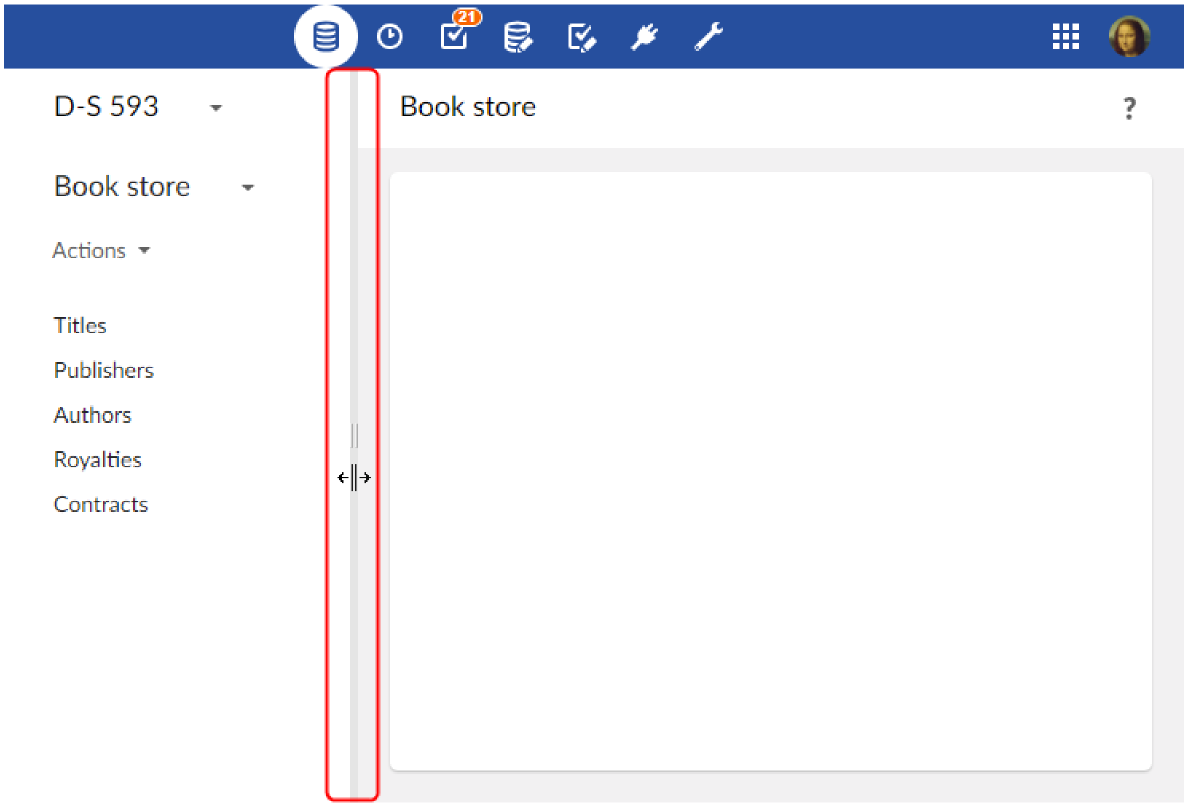
Density

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

2.5 User interface features

Resetting the navigation pane width

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



2.6 Where to find EBX help

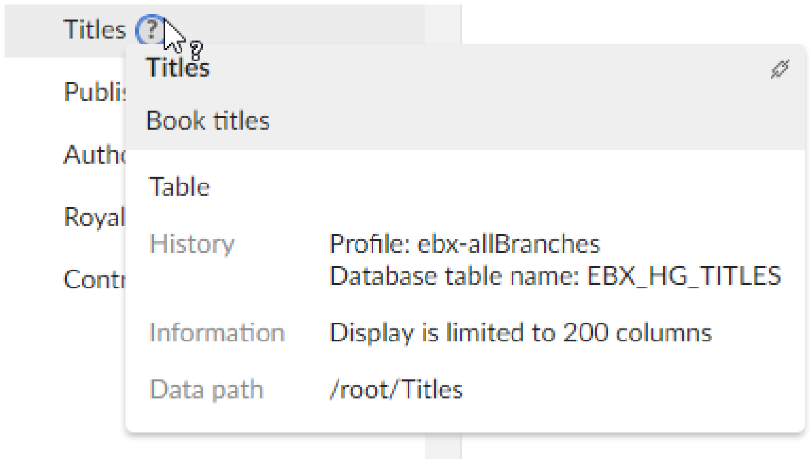
In addition to the full standalone product documentation accessible via the user pane [p 19], help is accessible in various forms within the interface.

Context-sensitive help

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

Contextual help on elements

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



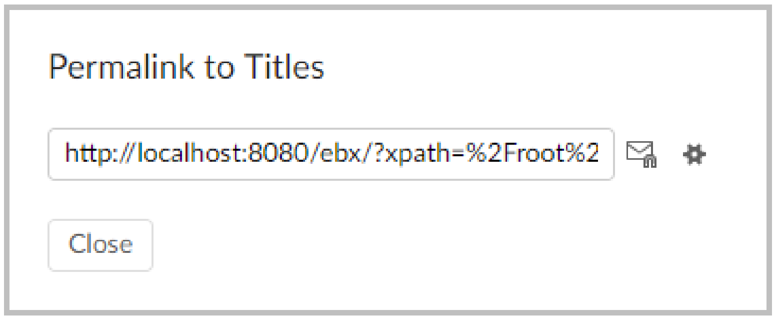
Titles

Book titles

Table

History	Profile: ebx-allBranches Database table name: EBX_HG_TITLES
Information	Display is limited to 200 columns
Data path	/root/Titles

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



Permalink to Titles

http://localhost:8080/ebx/?xpath=%2Froot%2

CHAPTER 3 Glossary

This chapter contains the following topics:

3.1 Governance

repository

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

See also dataspace [p 28].

profile

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

See also user [p 23], role [p 24].

Related Java API `Profile`API.

user

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

See also user and roles directory [p 24], profile [p 23].

Related concept User and roles directory [p 399].

Related Java API `UserReference`API.

role

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

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

See also user and roles directory [p 24], profile [p 23].

Related concept User and roles directory [p 399].

Related Java API `Role`API.

administrator

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

user and roles directory

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

See also user [p 23], role [p 24].

Related concept User and roles directory [p 399].

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

user session

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

Related concept User and roles directory [p 399].

Related Java API `Session`API.

3.2 Data modeling

Main documentation section Data models [p 34]

data model

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

See also dataset [p 26].

Related concept Data models [p 34].

field

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

By default, fields are denoted by the icon ▭.

See also record [p 26], group [p 25], table (in data model) [p 25], validation rule [p 26], inheritance [p 27].

Related concepts Structure elements properties [p 53], Controls on data fields [p 67].

Related Java API `SchemaNode`API.

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

primary key

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

Primary keys are denoted by the icon 🔑.

Related concept Tables definition [p 493].

foreign key

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

Foreign keys are denoted by the icon 🔗.

See also primary key [p 25].

Related concept Foreign key [p 498].

table (in data model)

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

Tables are represented by the icon ▦.

See also record [p 26], primary key [p 25], reusable type [p 26].

group

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

Groups are represented by the icon 🗀.

See also reusable type [p 26].

Related Java API `SchemaNode`API.

reusable type

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

validation rule

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

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

data model assistant (DMA)

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

See also Data models [p 34].

3.3 Datasets

Main documentation section Datasets [p 108]

record

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

See also table (in dataset) [p 26], primary key [p 25].

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

table (in dataset)

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

Tables are represented by the icon ▦.

See also record [p 26], primary key [p 25].

dataset

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

Datasets are represented by the icon 🗎.

See also table (in dataset) [p 26], field [p 25], group [p 25], views [p 27].

Related concept Datasets [p 108].

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

inheritance

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

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

Inheritance from the parent dataset is represented by the icon .

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

Inherited fields are represented by the icon .

Related concept Inheritance and value resolution [p 270].

views

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

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

See also

Views [p 115]

Hierarchies [p 117]

recommended view

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

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

Related concept Recommended views [p 118].

favorite view

When displaying a table, the user can choose to define the current as their favorite view through the 'Manage views' sub-menu.

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

Related concept Manage views [p 119].

3.4 Data management life cycle

Main documentation section Dataspaces [p 90]

dataspace

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

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

See also inheritance [p 27], repository [p 23], dataspace merge [p 28].

Related concept Dataspaces [p 90].

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

reference dataspace

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

See also dataspace [p 28], dataspace merge [p 28], repository [p 23].

dataspace merge

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

Related concept Merge [p 98].

snapshot

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

Snapshots are represented by the icon 📷.

Related concept Snapshot [p 103]

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

3.5 History

Main documentation section History [p 251]

historization

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

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

See also table history view [p 29], transaction history view [p 29], history profile [p 29].

history profile

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

See also history profile [p 29].

table history view

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

Related technical reference History [p 251].

transaction history view

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

Related technical reference History [p 251].

3.6 Workflow modeling

Main documentation section Workflow models [p 132]

workflow model

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

Related concept Workflow models [p 132].

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

Workflow models are represented by the icon .

script task

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

Script tasks are represented by the icon .

See also workflow model [p 29].

user task

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

User tasks are represented by the icon .

See also workflow model [p 29].

workflow condition

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

Workflow conditions are represented by the icon .

sub-workflow invocation

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

wait task

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

data context

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

3.7 Data workflows

Main documentation section Data workflows [p 160]

workflow publication

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

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

data workflow

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

See also workflow model [p 29].

Related concept Data workflows [p 160].

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

work list

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

work item

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

Allocated work items are represented by the icon ◇.

See also user task [p 30].

token

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

3.8 Data services

Main documentation section Data services [p 180]

data service

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

Data services offer:

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

lineage

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

Related concept: Generating a WSDL for lineage [p 185].

3.9 Cross-domain

node

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

- In the workflow model [p 29] context, a node is a workflow step or condition.
- In the data model [p 24] context, a node is a group, a table or a field.
- In the hierarchy [p 27] context, a node represents a value of a dimension.
- In an adaptation tree [p 27], a node is a dataset.
- In a dataset [p 26], a node is the node of the data model evaluated in the context of the dataset or the record.

Data models

CHAPTER **4**

Introduction to data models

This chapter contains the following topics:

4.1 **Overview**

What is a data model?

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

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

Basic concepts used in data modeling

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

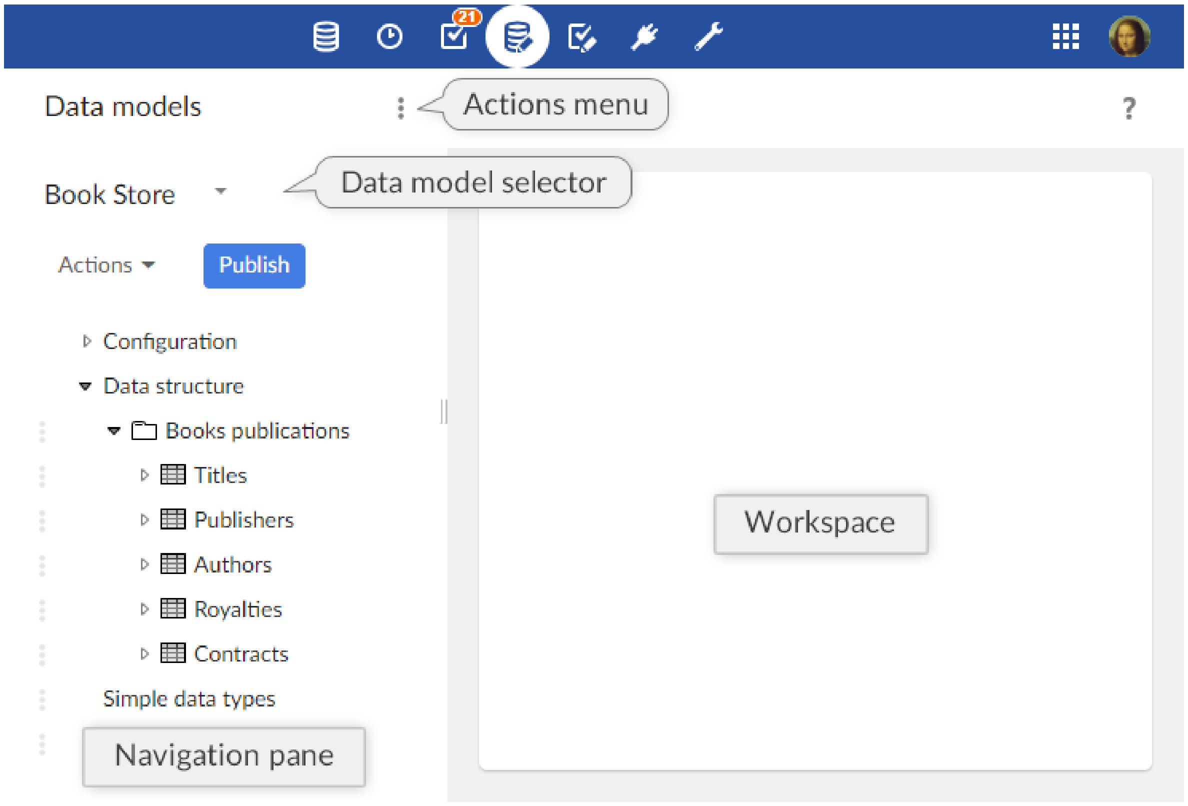
- field [p 25]
- primary key [p 25]
- foreign key [p 25]
- table (in data model) [p 25]
- group [p 25]
- reusable type [p 26]
- validation rule [p 26]

4.2 Using the Data Models area user interface

Navigating within the Data Model Assistant

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

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



The navigation pane is organized into the following sections:

Configuration	The technical configuration of the data model.
Global properties	Defines the global properties of the data model.
Included data models	Defines the data models included in the current model. All types defined in included data models can be reused in the current model.
Java bindings	The bindings specify what Java types have to be generated from the model.
Component library	Defines the Java components available in the model. These provide programmatic features that will be available for the model, such as programmatic constraints, functions, and UI beans.
Toolbars	The toolbars available to use in the data model.
Ajax components	Defines the available Ajax components in the model.
User services	Declares the user services using the API available before release 5.8.0. From release 5.8.0, it is advised to use the new UserService API (these services are directly registered through the Java API, hence no declaration is required for them in the data model assistant)..
Add-ons	Specifies which add-ons are used by the data model. These add-ons will have the capacity to enrich the current data model after the publication by adding properties and constraints to the elements of the data model.
Data services	Specifies the WSDL operations' suffixes that allow to refer to a table in the data service operations using a unique name instead of its path.
Replications	This table defines the replication units of the data model. A replication unit allows the replication of a source table in the relational database, so that external systems can access this data by means of plain SQL requests and views.
Data structure	The structure of the data model. Defines the relationship between the elements of the data model and provides access to the definition of each element.

Simple data types	Simple reusable types defined in the current data model.
Complex data types	Complex reusable types defined in the current data model.
Included simple data types	Simple reusable types defined in an included external data model.
Included complex data types	Complex reusable types defined in an included external data model.

See also

Implementing the data model structure [p 47]

Configuring the data model [p 41]

Reusable types [p 49]

Data model element icons

field [p 25]

primary key [p 25]

foreign key [p 25]

table [p 25]

group [p 25]

Related concepts

Dataspaces [p 90]

Datasets [p 108]

CHAPTER 5

Creating a data model

This chapter contains the following topics:

5.1 Creating a new data model

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

5.2 Selecting a data model type

If you are a user with the 'Administrator' role, you must choose the type of data model you are creating, *semantic* or *relational*, in the first step of the data model creation wizard.

Semantic models

Semantic models enable the full use of data management features, such as life cycle management using dataspaces, provided by TIBCO EBX. This is the default type of data model.

Relational models

Relational models are used when the tables created from a data model will eventually be mapped to a relational database management system (RDBMS). The primary benefit of using a relational model is the ability to query the tables created from the data model using external SQL requests. However, employing a relational model results in the loss of functionalities in EBX such as inheritance, multi-valued fields, and the advanced life cycle management provided by dataspaces.

> **Note**
>
> A relational data model can only be used by a single dataset, and the dataspace containing the dataset must also be declared as being relational.

See also

Relational mode *[p 245]*

Dataspaces *[p 90]*

CHAPTER 6

Configuring the data model

This chapter contains the following topics:

6.1 Information associated with a data model

To view and edit the owner and documentation of your data model, select 'Information' from the **data model 'Actions'** [p 35] menu for your data model in the navigation pane.

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

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

6.2 Permissions

To define the user permissions on your data model, select 'Permissions' from the **data model 'Actions'** [p 35] menu for your data model in the navigation pane.

The configuration of the permissions of a data model are identical to the options for the permissions of a dataset, as explained in Permissions [p 123].

6.3 **Data model properties**

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

Module name	Defines the module that contains the resources that will be used by this data model. This is also the target module used by the data model publication if publishing to a module.
Module path	Physical location of the module on the server's file system.
Sources locations	The source paths used when configuring Java components in the 'Component library'. If a path is relative, it will be resolved using the 'Module path' as the base path.
Publication mode	Whether to publish the data model as an XML Schema Document within a module or as a publication completely embedded in the TIBCO EBX repository. Embedded data models offer additional functionality such as versioning and rollback of publications. See Publication modes [p 85] for more information. **Model path in module**: Defines the target file for the data model generation. It must start with '/'.
Dataset inheritance	Specifies whether dataset inheritance is enabled for this data model. Dataset inheritance is disabled by default. See Dataset inheritance [p 127] for more information.
Documentation	Documentation of the data model defined by a Java class. This Java class can programmatically specify labels and descriptions for the elements of the data model. The labels and descriptions defined in this Java class are displayed in associated datasets in preference to the ones defined locally on an element. See Dynamic labels and descriptions [p 532] for more information.
Special extensions	Access permissions defined by programmatic rules in a Java class.
Disable auto-increment checks	Specifies whether to disable if the check of an auto-incremented field value in associated datasets regarding to the "max value" found in the table being updated. See Auto-incremented values [p 529] for more information.

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

6.4 Included data models

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

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

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

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

See also*Including external data models* *[p 491]*

6.5Data services

It is possible to refer to tables in Data Service operations using unique names instead of their paths by defining suffixes for WSDL operations. A WSDL suffix is the association between a table path and a name.

To define a WSDL suffix through the user interface, create a new record in the 'Data services' table under the data model configuration in the navigation pane. A record of this table defines the following properties:

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

See also*Data services* *[p 545]*

6.6 **Replication of data to relational tables**

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

To define a replication unit through the user interface, create a new record in the 'Replications' table under the data model configuration in the navigation pane. Each replication unit record is associated with a particular dataset in a given dataspace. A single replication unit can cover multiple tables, as long as they are in the same dataset. A replication unit defines the following information:

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

See also*Replication* *[p 259]*

6.7 **Add-ons used by the data model**

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

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

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

CHAPTER 7

Implementing the data model structure

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

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

This chapter contains the following topics:

7.1 Common actions and properties

Adding elements to the data model

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

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

Add a new element relative to any existing element in the data structure by clicking the down arrow ▾ to the right of the existing entry, and selecting an element creation option from the menu. Depending on whether the existing element is a field, group, or table, you have the choice of creating the new element as a child of the existing element, or before or after the existing element at the same level. You can then follow the element creation wizard to create the new element.

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

Names, labels, descriptions, and information

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

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

Deleting elements of the data model

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

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

Duplicating existing elements

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

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

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

Moving elements

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

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

7.2 Reusable types

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

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

Defining a reusable type

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

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

Using a reusable type

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

Including data types defined in other data models

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

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

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

See Included data models [p 43] for more information.

7.3 Data model element creation details

Creating fields

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

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

Creating tables

While creating a table, you have the option to create the new table based on an existing reusable type. See Reusable types [p 49] for more information.

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

Creating groups

While creating a group, you have the option to create the new group based on an existing reusable type. See Reusable types [p 49] for more information.

Creating primary key fields

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

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

Creating or defining foreign key fields

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

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

Creating associations

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

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

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

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

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

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

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

7.4 Modifying existing elements

Removing a field from the primary key

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

See primary key [p 25] in the glossary.

CHAPTER **8**

Properties of data model elements

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

See also*Data validation controls on elements* *[p 67]*

This chapter contains the following topics:

8.1 **Basic element properties**

Common basic properties

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

Information	Additional non-internationalized information associated with the element.
Minimum number of values	Minimum number of values for an element. As primary keys cannot be multi-valued, they must have this property set to '1' or 'undefined'. The minimum number of values is automatically set to '0' when defining the field as a selection node.
Maximum number of values	Maximum number of values for an element. When set to a value greater than '1', the element becomes multi-valued. As primary keys cannot be multi-valued, they must have this property set to '1' or 'undefined'. For tables, the maximum number of values is automatically set to 'unbounded' upon creation. The maximum number of values is automatically set to '0' when defining the field as a selection node.
Validation rules	This property is available for tables and fields in tables except `Password` fields, reusable types, fields in complex reusable types, and selection nodes. Used to define powerful and complex validation rules with the help of the provided XPath 1.0 criteria editor. See Criteria editor [p 293] for more information. This can be useful if the validation of the value depends on complex criteria or on the value of other fields. It is also possible to indicate that a rule defines a verification for a value that is mandatory under certain circumstances. In this case a value is mandatory if the rule is not satisfied. See Constraint on 'null' values [p 521] for more information. Using the associated wizard, you can define localized labels for the validation rule, as well as define a localized message with severity to be displayed if the criteria is not met. When defining the severity of the validation message it is possible to indicate whether an input that would violate a validation rule will be rejected or not when submitting a form. The error management policy is only available on validation rules defined on a field and when the severity is set to 'error'. If the validation rule must remain valid, then

	any input that would violate the rule will be rejected and the values will remain unchanged. If errors are allowed, then any input that would violate the rule will be accepted and the values will change. If not specified, the validation rule always blocks errors upon the form submission by default. If a validation rule is defined on a table, it will be considered as a 'constraint on table' and each record of the table will be evaluated against it at runtime. See Constraints on table [p 521] for more information.

Basic properties for fields

The following basic properties are specific to fields:

Default value	Default value assigned to this field. In new data creation forms, the default value appears automatically in the user input field. The default value must comply with the defined type of the field. See Default value [p 537] for more information.
Conversion error message	Internationalized messages to display to users when they enter a value that is invalid for the data type of this field.
Computation rule	This property is available for fields in tables, except in reusable types. Defines a rule for computing the value of the field using the provided XPath 1.0 editor. See criteria editor [p 293] This can be useful if the value depends on other values in the same record, but does not require a programmatic computation. The following limitations exist for computation rules: • Computation rules can only be defined on simple fields inside a table. • Computation rules cannot be defined on fields of type `OResource` or `Password`. • Computation rules cannot be defined on selection nodes and primary key fields. • Computation rules cannot be defined when accessing an element from the validation report.

8.2 Advanced element properties

Common advanced properties

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

Default view and tools > Visibility	Specifies whether or not this element is shown in the default view of a dataset, in the text search of a dataset or in the data service "select" operation. • Model-driven view Specifies whether or not the current element is shown in the default tabular view of a table, the default record form of a table, and in the default view of a dataset if the current element is a table. Default dataset view, tabular view and default record form generated from the structure of the data model. If the current element is inside a table, then setting the property to 'Hidden' will hide the element from the default tabular view and default record form of the table without having to define specific access permissions. Current element will still be displayed in the view configuration wizard to be able to create a custom view that displays this element. If the current element is a table, then setting the property to 'Hidden' will hide the table from the default view of a dataset without having to define specific access permissions. This property is ignored if it is set on an element that is neither a table nor in a table. • All views Specifies whether or not the current element is shown in all views of a table in a dataset. Setting the property to 'Hidden in all views' will hide the element in all views of the table, whether tabular (default tabular view included) or hierarchical, without having to define specific access permissions. The current element will also be hidden in the view configuration wizard. That is, it won't be possible to create a custom view that will display this element. This property is ignored if it is set on an element that is not in a table. This property is not applied on forms. That is, setting the property to 'Hidden in all views' will not hide the element in a record form but only in views. • Search tools Specifies whether or not the current element is shown in a dataset search tool. Setting the property to 'Hidden in all searches' will hide the element both in the text and typed search tools of a dataset. Setting the property to

	'Hidden only in text search' will only hide the element in the text search tool. If this property is not set, the element will be displayed in the text search tool by default. This property is ignored if it is set on an element that is not in a table. • Data services Specifies whether or not the current element is shown in the data service select operation. Setting the property to 'Excluded from Data Services' will hide the element in the data service "select" operation. This property is ignored if it is set on an element that is not in a table. See Default view [p 539] in the Developer Guide.
Default view and tools > Widget	Defines the widget to be used. A widget is an input component that is displayed in forms in associated datasets. If undefined, a default widget is displayed in associated datasets according to the type and properties of the current element. It is possible to use a built-in widget or a custom widget. A custom widget is defined using a Java API to allow the development of rich user interface components for fields or groups. Built-in and custom widgets cannot be defined on a table or an association. It is forbidden to define both a custom widget and a UI bean. It is forbidden to define on a foreign key field both a custom widget and a combo-box selector. See `UIWidgetFactory`API for more information.
Default view and tools > Combo-box selector	Specifies the name of the published view that will be used in the combo-box selection of the foreign key. A selection button will be displayed at the bottom right corner of the drop-down list. When defining a foreign key, this feature allows accessing an advanced selection view through the 'Selector' button that opens the advanced selection view, from where sorting and searching options can be used. If no published view is defined, the advanced selection view will be disabled. If the path of the referenced table is absolute then only the published views corresponding to this table will be displayed. If the path of the referenced table is relative then all the published views associated with the data model containing the target table will be displayed. This property can only be set if no custom widget is defined. See Defining a view for the combo box selector of a foreign key [p 541] in the Developer Guide.
UI bean	**Attention**

	From version TIBCO EBX 5.8.0, it is recommended to use widgets instead of UI Beans. Widgets provide more features than UI Beans, and no further evolution will be made on UI beans. See widget [p 57] for more information. This property is available for all elements except tables and associations. Specifies a Java class to customize the user interface associated with this element in a dataset. A UI bean can display the element differently and/or modify its value by extending the `UIBeanEditor`API class in the Java API.
Transformation on export	This property is available for fields and for groups that are terminal nodes. Specifies a Java class that defines transformation operations to be performed when exporting an associated dataset as an archive. The input of the transformation is the value of this element. See `NodeDataTransformer`API for more information.
Access properties	Defines the access mode for the current element, that is, if its data can be read and/or written. • 'Read & Write' corresponds to the mode `RW` in the data model XSD. • 'Read only' corresponds to the mode `R-` in the data model XSD. • 'Not a dataset node' corresponds to the mode `CC` in the data model XSD. • 'Non-terminal node' corresponds to the mode `--` in the data model XSD. See Access properties [p 537] in the Developer Guide.
Comparison mode	Defines the comparison mode associated with the element, which controls how its differences are detected in a dataset. • 'Default' means the element is visible when comparing associated data. • 'Ignored' implies that no changes will be detected when comparing two versions of modified content (records or datasets). During a merge, the data values of ignored elements are not merged even if the content has been modified. However, values of ignored data sets or records being created during the operation are merged. During an archive import, values of ignored elements are not imported when the content has been modified.

	However, values of ignored datasets or records being created during the operation are imported. See Comparison mode [p 542] in the Developer Guide.
Apply last modifications policy	Defines if this element must be excluded from the service allowing to apply the last modifications that have been performed on a record to the other records of the same table. • 'Default' means that the last modification on this element can be applied to other records. • 'Ignored' implies that the last modification on this element cannot be applied to other records. This element will not be visible in the apply last modifications service. See Apply last modifications policy [p 542] in the Developer Guide.
Node category	Defines a category for this element. Categories allow controlling the visibility of data in a dataset to users. A node with the category 'Hidden' is hidden by default. Restriction: category specifications other than 'Hidden' do not apply to table record nodes. See Categories [p 543] in the Developer Guide.

Advanced properties for fields

The following advanced properties are specific to fields.

Check null input

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

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

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

See Constraints, triggers and functions [p 525] in the Developer Guide.

Trim whitespaces

Trim white spaces

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

See Whitespace handling upon user input [p 526] in the Developer Guide.

UI bean

See Common advanced properties [p 57].

Function (computed value)

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

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

Disable validation

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

Transformation on export

See Common advanced properties [p 58].

Access properties

See Common advanced properties [p 58].

Auto-increment

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

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

Auto-incremented values have the following behavior:

- The computation and allocation of the field value are performed whenever a new record is inserted and the field value is yet undefined.

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

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

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

See Auto-incremented values [p 529] in the Developer Guide.

Default view

See Common advanced properties [p 56].

Node category

See Common advanced properties [p 59].

Inherited field

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

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

See inheritance [p 27] in the glossary.

For more information, see also Inherited fields [p 272].

Advanced properties for tables

The following advanced properties are specific to tables.

Table

Primary key	A list of fields in the table that compose the table's primary key. You can add or remove primary key fields here, as in the 'Data structure' view. Each primary key field is denoted by its absolute XPath notation that starts under the table's root element. If there are several elements in the primary key, the list is white-space delimited. For example, "/name /startDate".
Presentation	Specifies how records are displayed in the user interface of this table in a dataset.
Presentation **> Record labeling**	Defines the fields to provide the default and localized labels for records in the table. Can also specify a Java class to set the label programmatically, or set the label in a hierarchy. This Java class must implement either the `UILabelRenderer`API interface or the `UILabelRendererForHierarchy`API interface. **Attention:** Access rights defined on associated datasets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
Presentation **> Default rendering for groups in forms**	Specifies the default display rendering mode of the groups contained in this table. If nothing is defined here, the default policy set in the Administration area will be used to display groups. See Record form: rendering mode for nodes [p 390] in the Administration Guide. **Enabled rendering for groups** Specifies a display rendering mode to be enabled for groups in the table in addition to the modes 'expanded' and 'collapsed', which are always available. Tabs must be enabled on the table to have the option to display groups as tabs. Similarly, links must be enabled to have the option to display groups as links. **Default rendering for groups** Specifies the default display rendering mode to use for the groups contained in this table. If a group does not specify a default mode then the default mode defined for this table will be used. Links must be enabled to define the default rendering mode as 'Link'. Select a rendering mode according to network and browser performance. Link mode is lighter

	as its content is not displayed on the same page, whereas the expanded and collapsed modes are heavier. **Note:** When the tabs view is enabled on a table, any groups that would otherwise have used links are automatically converted to collapsed mode. This is done to avoid the inherent display layout complexities that would arise from links and tabs coexisting in the same user interface.
***Presentation* > Specific rendering of forms**	Defines a specific rendering for customizing the record form in a dataset. See `UIForm`API and `UserServiceRecordFormFactory`API for more information.
Toolbars	Defines the toolbars to use in this table. Toolbars can be edited in the *Configuration* > *Toolbars* section. **Tabular view top:** Defines the toolbar to use on top of the default table view. **Tabular view row:** Defines the toolbar to use on each row of the default table view. **Record top:** Defines the toolbar to use in the record form. **Hierarchy top:** Defines the toolbar to use in the default hierarchy view of the table. See Toolbars [p 75] for more information.
History	Specifies when historization is to be performed, and the level of guarantee requested. The available history profiles can be edited in **Administration > History and logs**. See History configuration in the repository [p 251] for more information.
Indexes	Defines the fields to index in the table. Indexing speeds up table access for requests on the indexed fields. No two indexes can contain the exact same fields. **Index name**: Unique name for this index. **Fields to index**: The fields to index, each represented by an absolute XPath notation starting under the table `root` element.
Specific filters	Defines record display filters on the table.
Actions	Specifies the actions that are allowed on the table in associated datasets. By default, all actions are allowed unless specific access rights are defined in a dataset.

Uniqueness constraints

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

Triggers

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

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

See Triggers [p 528] in the Developer Guide.

Access properties

See Common advanced properties [p 58].

Default view

See Common advanced properties [p 56].

Node category

See Common advanced properties [p 59].

Advanced properties for groups

The following advanced properties are specific to groups.

Value container class (JavaBean)

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

UI bean

See Common advanced properties [p 57].

Transformation on export

See Common advanced properties [p 58].

Access properties

See Common advanced properties [p 58].

Default view

Visibility	See Common advanced properties [p 56].
Rendering in forms	Defines the rendering mode of this group. If this property is not set, then the default view for groups specified by the container table will be used. 'Tab' and 'Link' are each only available when the container table enables it. ***Tab position*** This attribute specifies the position of the tab with respect to all the tabs defined in the model. This position is used for determining tab order. If a position is not specified, the tab will be displayed according to the position of the group in the data model.

Node category

See Common advanced properties [p 59].

Related concepts*Data validation controls on elements [p 67]*

CHAPTER **9**

Data validation controls on elements

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

See also*Properties of data model elements* *[p 53]*

This chapter contains the following topics:

9.1 Simple content validation

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

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

See XML schema supported facets [p 513].

9.2 Advanced content validation

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

See also *Dynamic constraints* [p 516]

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

Fixed length [dynamic]	Defines a field to provide the exact number of characters required for this field.
Minimum length [dynamic]	Defines a field to provide the minimum number of characters allowed for this field.
Maximum length [dynamic]	Defines a field to provide the maximum number of characters allowed for this field.
Excluded values	Defines a list of values that are not allowed for this field.
Excluded segment	Defines an inclusive range of values that are not allowed for this field. **Minimum excluded value:** Lowest value not allowed for this field. **Maximum excluded value:** Highest value not allowed for this field.
Specific constraint (component)	Specifies one or more Java classes that implement the `Constraint`API interface of the Java API. See Programmatic constraints [p 520] for more information.
Specific enumeration (component)	Specifies a Java class to define an enumeration. The class must define an ordered list of values by implementing the `ConstraintEnumeration`API interface of the Java API.
Enumeration filled by another node	Defines the possible values of this enumeration using a reference to another list or enumeration element.
Dataspace set configuration	Define the dataspaces that can be referenced by a field of the type Dataspace identifier (osd:dataspaceKey). If a configuration is not set, then only opened branches can be referenced by this field by default. • Includes Specifies the dataspaces that can be referenced by this field. **Pattern:** Specifies a pattern that filters dataspaces. The pattern is checked against the name of the dataspaces. **Type:** Specifies the type of dataspaces that can be referenced by this field. If not defined, this restriction is applied to branches. **Include descendants:** Specifies if children or descendants of the dataspaces that match the specified pattern are included in the set. If not defined, this restriction is not applied to child dataspaces. If "None"

then neither children nor descendants of the dataspaces that match the specified pattern are included. If "All descendants" then all descendants of the dataspaces that match the specified pattern are included. If "All descendant branches" then all descendant branches of the dataspaces that match the specified pattern are included. If "All descendant snapshots" then all descendant snapshots of the dataspaces that match the specified pattern are included. If "Child branches" then only direct branches of the dataspaces that match the specified pattern are included. If the current dataspace is a version, includes the branches that are the direct children of this version; if the current dataspace is a branch, includes the branches that are the direct children of the versions which are children of this branch. If "Child snapshots" then only direct snapshots of the dataspaces that match the specified pattern are included. If the current dataspace is a branch, includes the snapshots that are the direct children of this branch; if the current dataspace is a version, includes the versions that are the direct children of the branches which are children of this version.

- Excludes

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

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

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

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

	a branch, includes the branches that are the direct children of the versions which are children of this branch. If "Child snapshots" then only direct snapshots of the dataspaces that match the specified pattern are included. If the current dataspace is a branch, includes the snapshots that are the direct children of this branch; if the current dataspace is a version, includes the versions that are the direct children of the branches which are children of this version. • Dataspace filter Specifies a filter to accept or reject dataspaces in the context of a dataset or a record. This filter is only used in the dedicated input component that is associated to this field. That is, this filter is not used when validating this field. A specific constraint can be used to perform specific controls on this field. A filter is defined by a Java class that implements the `DataspaceSetFilter`API interface of the Java API.
Dataset set configuration	Define the datasets that can be referenced by a field of the type Dataset identifier (osd:datasetName). • Includes Specifies the datasets that can be referenced by this field. **Pattern:**Specifies a pattern that filters datasets. The pattern is checked against the name of the datasets. **Include descendants:** Specifies if children or descendants of the datasets that match the specified pattern are included in the set. • Excludes Specifies the datasets that cannot be referenced by this field. Excludes are ignored if no includes are defined. **Pattern:** Specifies a pattern that filters datasets. The pattern is checked against the name of the datasets. **Include descendants:** Specifies if children or descendants of the datasets that match the specified pattern are included in the set. • Filter Specifies a filter to accept or reject datasets in the context of a dataset or record. This filter is only used in the dedicated input component that is associated to this field. That is, this filter is not used when validating this field. A specific constraint can be used to perform specific controls on this field. A filter is defined by

a Java class that implements the `DatasetSetFilter`API interface of the Java API.

Validation properties

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

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

Related concepts*Properties of data model elements* *[p 53]*

CHAPTER 10 Toolbars

This chapter contains the following topics:

10.1 Definition

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

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

Name	Toolbar's name. The name of the toolbar must be unique in the context of the data model. That is, it is not allowed to create several toolbars with the same name.
Label and description	Internationalized labels and descriptions to be displayed to end users.
Default template	Allows to create a toolbar with the structure of a default toolbar.
Locations	Specifies the locations where the toolbar can be used in associated datasets.

Defining the structure of a toolbar

A toolbar can define the following elements:

- Action button [p 77]
- Menu button [p 78]
- Separator [p 78]
- Menu group [p 79]

- Action menu item [p 80]
- Sub menu item [p 81]

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

Action button

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

Target	Defines if the service is executed in the current context or in a web component. A service executed in a web component has no access to the current selection, but it is possible to specify a selection and specific parameters.
Service	Defines the service that will be executed when the user clicks on the button. It is possible to select a built-in service, or a user service defined in a module or in the current data model. If the 'Web component' target is selected, the service will have to be declared as available as a web component for toolbars. **See also** `WebComponentDeclarationContext.setAvailableAsToolbarAction` API
Modal size	Size of the modal window containing the web component.
Resizable	Indicates if the modal window can be resized.
Parameters	Service parameters. Those can be specified only if the target is set to 'Web component'.
Label and description	Internationalized labels and descriptions to be displayed to end users.
Layout	Defines how this element will be displayed in datasets using the toolbar. It is possible to display: the icon only, the text only, text with the icon to the left or text with the icon to the right.
Icon	Icon to display. It is possible to use an icon to choose from a set of suggested icons, or to refer to an icon using a URL.
Is highlighted	Indicates if the button should be highlighted by default.

Note

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

Menu button

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

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

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

Separator

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

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

Menu group

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

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

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

- Menu button

- Sub menu item

Action menu item

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

Label and description	Internationalized labels and descriptions to be displayed to end users.
Target	Defines if the service is executed in the current context or in a web component. A service executed in a web component has no access to the current selection, but it is possible to specify a selection and specific parameters.
Service	Defines the service that will be executed when the user clicks on the button. It is possible to select a built-in service, or a user service defined in a module or in the current data model. If the 'Web component' target is selected, the service will have to be declared as available as a web component for toolbars. **See also** `WebComponentDeclarationContext.setAvailableAsToolbarAction` API
Modal size	Size of the modal window containing the web component.
Resizable	Indicates if the modal window can be resized.
Parameters	Service parameters. Those can be specified only if the target is set to 'Web component'.

Note

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

Sub menu item

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

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

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

Deleting elements

All the elements of a toolbar can be deleted from it by using the arrow ▾ located to the right of the element to be deleted.

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

Duplicating existing elements

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

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

Moving elements

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

Associate with existing tables

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

Exporting the toolbars

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

> **Note**
>
> A selection of toolbars can be exported by selecting in the 'Toolbars' section the toolbars to be exported and then by selecting the *XML export* option available in the *Actions* menu. The toolbars can also be exported by using the data model export service. It can be found in the **Data model 'Actions'** [p 35] menu in the navigation pane.

See also*XML Schema Document (XSD) import and export* *[p 83]*

Importing toolbars

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

> **Note**
>
> The toolbars can also be imported by using the data model import service accessible via the **Data model 'Actions'** [p 35] menu in the navigation pane.

See also*XML Schema Document (XSD) import and export* *[p 83]*

See also *Use of toolbars* *[p 63]*

CHAPTER **11**

Working with an existing data model

Once your data model has been created, you can perform a number of actions that are available from the **data model 'Actions'** [p 35] menu in the workspace.

This chapter contains the following topics:

11.1 Validating a data model

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

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

See Validation [p 300] for detailed information on incremental data validation.

11.2 XML Schema Document (XSD) import and export

TIBCO EBX includes built-in data model services to import from and export to XML Schema Document (XSD) files. XSD imports and exports can be performed from the **data model 'Actions'** [p 35] menu of the target data model in the navigation pane. An XSD import or export is always performed in the context of a single data model. That is, during imports, the structure of the target data model is completely replaced with the content of the imported XSD. Similarly, during exports, the entire data model is included in the XSD file.

When importing an XSD file, the file must be well-formed and must comply with EBX validation rules. If this document declares resources that are located in a module, the module must also be declared in the configuration of the data model. If the module has not been declared, you will not be able to import the XSD file. See Data model properties [p 42] for more information on declaring modules.

To perform an import select 'Import XSD' from the **data model 'Actions'** [p 35] menu of the data model into which you are importing.

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

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

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

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

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

11.3 Duplicating a data model

To duplicate a data model, select 'Duplicate' from the **data model 'Actions'** [p 35] menu for that data model. You must give the new data model a name that is unique in the repository.

11.4 Deleting a data model

To delete a data model, select 'Delete' from the **data model 'Actions'** [p 35] menu for that data model. When you delete a data model, all of its existing publications will remain and continue to be accessible to their associated datasets. If you recreate a new data model with the same name as one that was previously deleted, the new data model will be reassociated with all the existing publications in the repository. At publication time of the new data model, you will have the opportunity to confirm the replacement of an existing publication.

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

See Publishing data models [p 85] for more information on the publication process.

CHAPTER 12

Publishing a data model

This chapter contains the following topics:

12.1 About publications

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

> **Note**
>
> The **Publish** button is only displayed to users who have permission to publish the data model. See Data model permissions [p 41] for more information.

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

12.2 Publication modes

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

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

12.3 Embedded publication mode

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

See Viewing and creating publications [p 86] for more information on the use of different publications.

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

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

> **Note**
>
> Snapshots, which are static archives of the state of the data model, must not be confused with data model *versions*, which act instead as parallel evolving branches of the data model. See Versioning embedded data models [p 87] for more information on data model versions.

Viewing and creating publications

To access the publications that exist for the current data model, select 'Manage publications' from its **data model 'Actions'** [p 35] menu in the navigation pane. From there, you can view the details of the publications and create new publications.

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

To create a new publication, select 'Manage publications' from the **data model 'Actions'** [p 35] menu of the data model in the navigation pane, then click the **Create publication** button. The name you give to the publication must unique in the repository.

CHAPTER 13

Versioning a data model

This chapter contains the following topics:

13.1 About versions

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

13.2 Accessing versions

To see the existing versions of your data model, select 'Manage versions' from the **data model 'Actions'** [p 35] menu of the data model.

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

13.3 Working with versions

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

Access data model version	Go to the corresponding version of the data model.
Create version	Creates a new version based on the contents of the selected version. The new version is added as a child of the selected version, though its contents bear no relation to those of its parent version after creation.
Set as default version	Sets the selected version as the default version opened when users access the data model.
Export archive	Exports the selected data model version to an archive containing the version's content, including its permissions and information. The exported archive is located in the archives directory, which is accessible to repository administrators. Exporting to an existing archive name will overwrite the existing file. See Archives directory [p 376] for more information.
Import archive	Imports the content of an archive into the selected version. The archive to import must contain a data model with the same name as the data model associated with the version.

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

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

13.4 Known limitations on data model versioning

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

Dataspaces

CHAPTER 14

Introduction to dataspaces

This chapter contains the following topics:

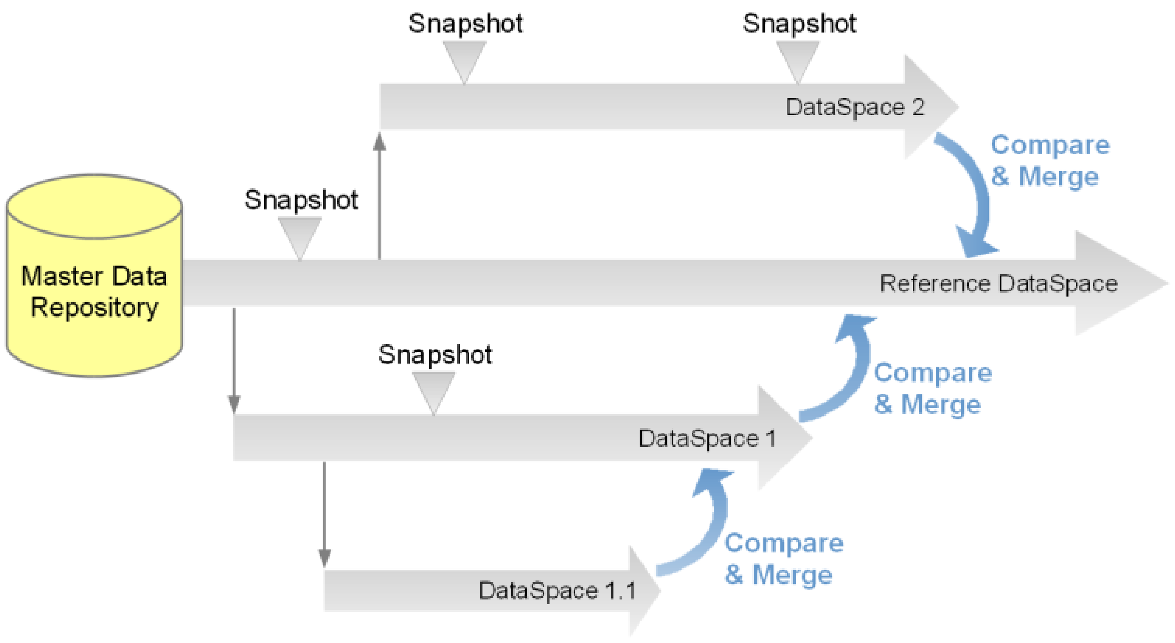
14.1 Overview

What is a dataspace?

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

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

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



Basic concepts related to dataspaces

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

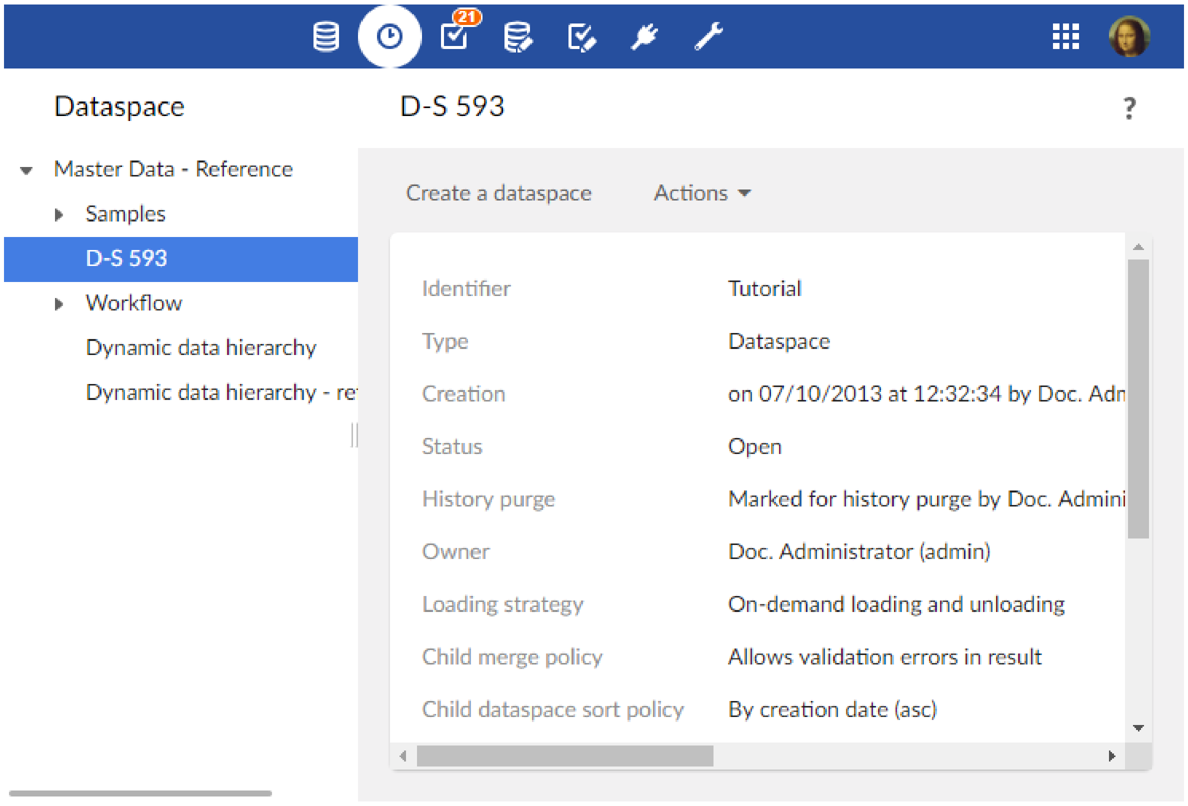
- dataspace [p 28]
- snapshot [p 28]
- dataset [p 26]
- dataspace merge [p 28]
- reference dataspace [p 28]

14.2 **Using the Dataspaces area user interface**

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

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

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



Identifier	Tutorial
Type	Dataspace
Creation	on 07/10/2013 at 12:32:34 by Doc. Adn
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)

See also

Creating a dataspace *[p 93]*

Snapshots *[p 103]*

Related concepts*Datasets* *[p 108]*

CHAPTER **15**

Creating a dataspace

This chapter contains the following topics:

15.1 Overview

By default, dataspaces in TIBCO EBX are in *semantic mode*. This mode offers full-featured data life cycle management.

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

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

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

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

See also*Relational mode* *[p 94]*

15.2 Properties

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

Identifier	Unique identifier for the dataspace.
Owner	Owner of the dataspace, who is, by default, allowed to modify its information and permissions. The owner does not necessarily have to be the creator of the dataspace.
Label	Localized label and description associated with the dataspace.
Relational mode	Whether or not this dataspace is in relational mode. This option only exists when creating a new dataspace from the reference dataspace.

15.3 Relational mode

Dataspaces in relational mode can only be created from the reference dataspace. They offer limited functionality compared to dataspaces in semantic mode. For instance, dataspaces in relational mode do not handle snapshots or support child dataspaces.

Relational mode indicates that the tables of data in this dataspace are stored directly in an RDBMS.

See also*Relational mode* *[p 245]*

CHAPTER 16

Working with existing dataspaces

This chapter contains the following topics:

16.1 Dataspace information

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

Documentation	Localized labels and descriptions associated with the dataspace.
Loading strategy	*Only administrators can modify this setting.* • **On demand loading and unloading:** The main advantage of this strategy is the ability to free memory when needed. Its disadvantage is the performance cost associated with a resource being accessed for the first time since server startup, or accessed after having been unloaded. This is the default mode. • **Forced loading:** This mode is recommended for dataspaces and snapshots used heavily or demanding in terms of response time. • **Forced loading and prevalidation:** This mode is recommended for dataspaces and snapshots used heavily or demanding in terms of response time, and where the validation process can be time-intensive. **Note:** Whenever the loading strategy is changed, you must restart the server for the new setting to take effect.
Child merge policy	This merge policy only applies to user-initiated merge processes; it does not apply to programmatic merges, for example, those performed by workflow script tasks. The available merge policies are: • **Allows validation errors in result:** Child dataspaces can be merged regardless of the validation result. This is the default policy. • **Pre-validating merge:** A child dataspace can only be merged if the result would be valid.
Current Owner	Owner of the dataspace, who is, by default, allowed to modify its information and permissions. The owner does not necessarily have to be the creator of the dataspace.
Child dataspace sort policy	Defines the display order of child dataspaces in dataspace trees. If not defined, the policy of the parent dataspace is applied. Default is 'by label'.

Change owner	Whether the current owner of the dataspace is allowed to give ownership to another profile by modifying its 'Current owner' property. If the value is 'Forbidden', only an administrator can modify the owner.
Change permissions	Whether the current owner of the dataspace is allowed to modify its permissions. If the value is 'Forbidden', only an administrator can modify the permissions of the dataspace.

16.2 **Dataspace permissions**

General permissions

Dataspace id	The dataspace to which the permissions will apply.
Profile selection	The profile to which the rule applies.
Restriction policy	Whether these permissions restrict the permissions assigned to a given user through policies defined for other profiles. See Restriction policy [p 283].
Dataspace access	The global access permission on the dataspace. **Read-only** • Can see the dataspace and its snapshots, as well as child dataspaces, according to their permissions. • Can see the contents of the dataspace depending on their permissions; cannot make modifications. **Write** • Can see the dataspace and its snapshots, as well as child dataspaces, according to their permissions. • Can modify the contents of the dataspace depending on their permissions. **Hidden** • Cannot see the dataspace nor its snapshots directly. • From a child dataspace, the current dataspace can be seen but not selected. • Cannot access the contents of the dataspace. • Cannot perform any actions on the dataspace.

Allowable actions

Users can be allowed to perform the following actions:

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

16.3 **Merging a dataspace**

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

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

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

Initiating a merge

To merge a dataspace into its parent dataspace:

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

Reviewing and accepting changes

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

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

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

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

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

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

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

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

Types of modifications

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

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

Types of conflicts

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

Conflicts are categorized as follows:

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

Finalizing a merge

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

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

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

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

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

Setting the child merge policy of a dataspace

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

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

See also*Child merge policy* *[p 100]*

Abandoning a merge

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

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

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

16.4 Comparing a dataspace

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

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

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

See also*Compare contents* *[p 211]*

16.5 Validating a dataspace

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

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

16.6 Dataspace archives

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

Exporting

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

> **Note**
>
> See Archives directory [p 376] in the Administration Guide for more information.

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

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

Importing

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

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

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

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

16.7 Closing a dataspace

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

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

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

See also*Closing unused dataspaces and snapshots* *[p 378]*

CHAPTER **17**

Snapshots

This chapter contains the following topics:

17.1 Overview of snapshots

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

See also*Snapshot* *[p 28]*

17.2 Creating a snapshot

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

The following information is required:

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

17.3 **Viewing snapshot contents**

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

17.4 **Snapshot information**

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

Documentation	Localized labels and descriptions associated with the snapshot.
Loading strategy	*Only administrators can modify this setting.* See Loading strategy [p 96].
Current Owner	Owner of the snapshot, who is, by default, allowed to modify its information and permissions. The owner does not necessarily have to be the creator of the snapshot.
Change owner	Whether the current owner of the snapshot is allowed to give ownership to another profile by modifying its 'Current owner' property. If the value is 'Forbidden', only an administrator can modify the owner.

17.5 **Comparing a snapshot**

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

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

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

See also*Compare contents* *[p 211]*

17.6 **Validating a snapshot**

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

Note

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

17.7 **Export**

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

> **Note**
>
> See Archives directory [p 376] in the Administration Guide for more information.

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

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

17.8 **Closing a snapshot**

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

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

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

See also*Closing unused dataspaces and snapshots* *[p 378]*

Datasets

CHAPTER **18**

Introduction to datasets

This chapter contains the following topics:

18.1 **Overview**

What is a dataset?

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

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

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

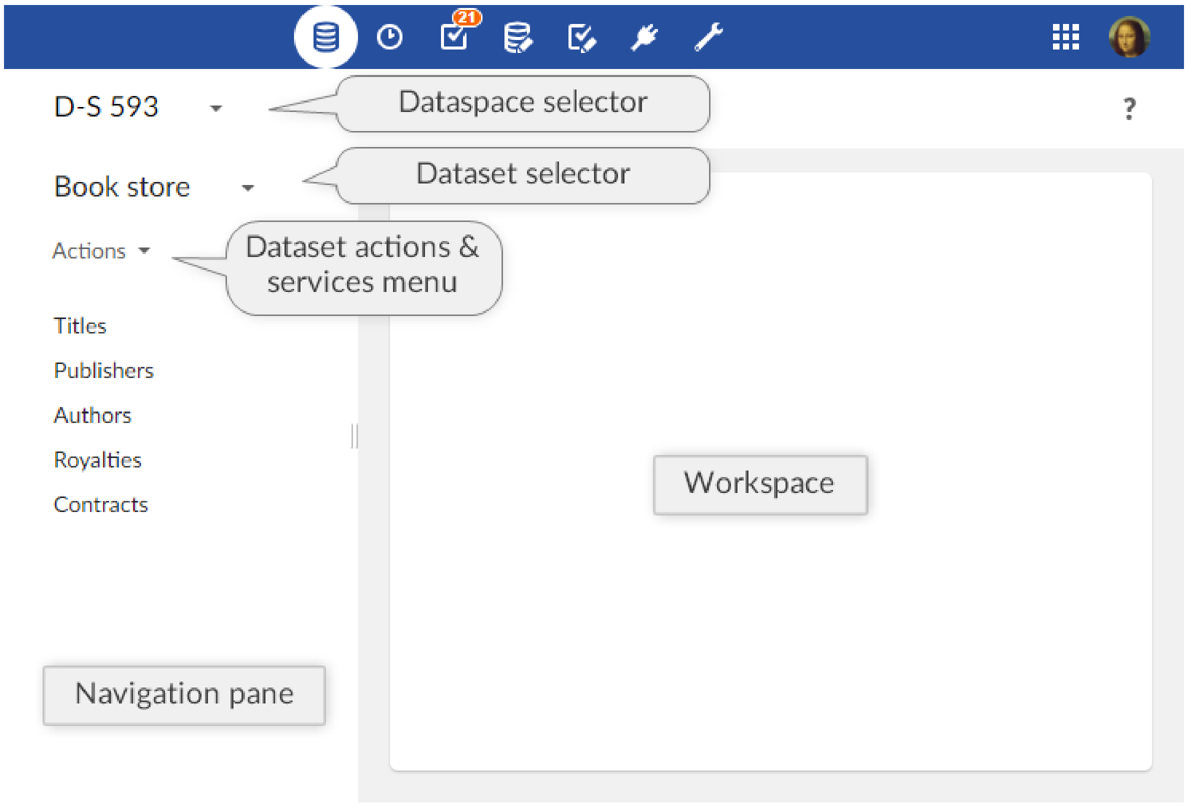
Basic concepts related to datasets

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

- dataspace [p 28]
- dataset [p 26]
- record [p 26]
- field [p 25]
- primary key [p 25]
- foreign key [p 25]
- table (in dataset) [p 26]
- group [p 25]

18.2 Using the Data user interface

Datasets can be created, accessed and modified in the **Data** area using the Advanced perspective [p 17] or from a specifically configured perspective. Only authorized users can access these interfaces.



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

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

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

See also

Creating a dataset [p 111]

Searching and filtering data [p 114]

Working with records in the user interface [p 121]

Inheritance [p 27]

Related concepts

Data model [p 34]

Dataspace [p 90]

CHAPTER **19**

Creating a dataset

This chapter contains the following topics:

19.1 Creating a root dataset

To create a new root dataset, that is, one that does not inherit from a parent dataset, select the '**Select dataset** [p 109]' ▾ menu in the navigation pane, click the '**Create a dataset**' button in the pop-up, and follow through the wizard.

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

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

- A *packaged data model* is a data model that is located within a module, which is a web application.
- An *embedded data model* is a data model that is managed entirely within the TIBCO EBX repository.
- An *external data model* is one that is stored outside of the repository and is referenced using its URI.

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

Attention

Table contents are not copied when duplicating a dataset.

19.2 **Creating an inheriting child dataset**

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

To create a child dataset, select the '**Select dataset** [p 109]' ▼ menu in the navigation pane, then click the + button next to the desired parent dataset.

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

See also*Dataset inheritance [p 127]*

CHAPTER 20
Viewing table data

TIBCO EBX offers a customization mechanism for tables via the 'Views' feature. A view allows specifying which columns should be displayed as well as the display order. Views can be managed by profile thanks to the recommended views [p 118] concept.

This chapter contains the following topics:

20.1 'View' menu

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

Views are managed in a dedicated sub-menu: 'Manage views' [p 119].

Views can also be grouped. An administrator has to define groups beforehand in 'Views configuration' under the 'Groups of views' table. The end-user can then set a view as belonging to a group through the field 'View group' upon creation or modification of the view. See 'View description' [p 116] for more information.

20.2 Sorting data

To simply sort a single column in a table, click on the column title. The first click will sort by ascending order and a second click will reverse the sorting.

Note that the default order is by ascending primary key.

For more advanced sorting, custom sort criteria allow specifying the display order of records.

To define custom sort criteria, click on the 'Select and sort' button in the workspace.

Each sort criterion is defined by a column name and a sort direction, that is, ascending or descending. Use the 'Move left' or 'Move right' arrows to add or remove a criterion from the 'Sorted' table. When a criterion is highlighted, you can set its sort direction by clicking on the 'ASC' or 'DESC' button to the right.

To change the priority of a sort criterion, highlight it in the list, then use the up and down arrow buttons to move it.

To remove a custom sort order that is currently applied, select *View > Reset*.

20.3 **Searching and filtering data**

The feature for searching and filtering records is accessible via the icon 🔍 in the workspace.

When criteria are defined for a search or filter, a checkbox appears in the title bar of the pane to apply the filter. When unchecked, the search or filter is not applied.

> **Note**
>
> Applying a view resets and removes all currently applied searches and filters.

Search

In simple mode, the 'Search' tool allows adding type-contextual search criteria to one or more fields. Operators relevant to the type of a given field are proposed when adding a criterion.

By enabling the advanced mode, it is possible to build sub-blocks containing criteria for more complex logical operations to be involved in the search results computation.

> **Note**
>
> In advanced mode, the criteria with operators "matches" or "matches (case sensitive)" follow the standard regular expression syntax from Java.

See also*Regex pattern*

Text search

The text search is intended for plain-text searches on one or more fields. The text search does not take into account the types of the fields being searched for.

- If the entered text contains one or more words without wildcard characters (`*` or `?`), matching fields must contain all specified words. Words between quotes, for example "aa bb", are considered to be a single word.
- Standard wildcard characters are available: `*` (any text) or `?` (any character). For performance reasons, only one of these characters can be entered in each search.
- Wildcard characters themselves can be searched for by escaping them with the character '\'. For example '*' will search for the asterisk character.

Examples:

- `aa bb`: field contains 'aa' and 'bb'.
- `aa "bb cc"`: field contains 'aa' and 'bb cc'.
- `aa*`: field label starts with 'aa'.
- `*bb`: field label ends with 'bb'.
- `aa*bb`: field label starts with 'aa' and ends with 'bb'.

- `aa?`: field label starts with 'aa' and is 3 chars long.
- `?bb`: field label ends with 'bb' and is 3 chars long.
- `aa?bb`: field label starts with 'aa' and ends with 'bb' and is 5 chars long.
- `aa\*bb`: field contains 'aa*bb' as is.

For large tables, it is recommended to select only one field, and for cases where the field type is not a string, to try to match the format type. For example:

- boolean: Yes, No
- date: 01/01/2000
- numeric: 100000 or 100,000
- enumerated value: Red, Blue...

The text search can be made case sensitive, that is distinguishing between upper and lower case, by checking the 'Case sensitive' checkbox.

Validation messages filter

The validation messages filter allows viewing records according to their status as of the last validation performed. Available levels are: 'Errors', 'Warnings', or 'Information'.

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

Custom table searches

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

See also*Specifying UI filters on a table* *[p 190]*

20.4 Views

It is possible to customize the display of tables in EBX according to the target user. There are two types of views: tabular [p 116] and hierarchical [p 117].

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

Two types of views can be created:

- 'Simple tabular view': A table view to sort and filter the displayed records.
- 'Hierarchical view': A tree view that links data in different tables based on their relationships.

View description

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

Documentation	Localized label and description associated with the view.
Owner	Name of the owner of the view. This user can manage and modify it. (Only available for administrators and dataset owners)
Share with	Other profiles allowed to use this view from the 'View' menu. **Note** Requires a permission, see Views permissions [p 397].
View mode	Simple tabular view or hierarchical view.
View group	Group to which this view belongs (if any).

Simple tabular views

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

Displayed columns	Specifies the columns that will be displayed in the table.
Sorted columns	Specifies the sort order of records in the table. See Sorting data [p 113].
Filter	Defines filters for the records to be displayed in the table. See Criteria editor [p 293].
Pagination limit	Forces a limit to the number of visible records.
Grid edit	If enabled, users of this view can switch to grid edit, so that they can edit records directly from the tabular view.
Disable create and duplicate	If yes, users of this view cannot create nor duplicate records from the grid edit.

Hierarchical views

A hierarchy is a tree-based representation of data that allows emphasizing relationships between tables. It can be structured on several relationship levels called dimension levels. Furthermore, filter criteria can be applied in order to define which records will be displayed in the view.

Hierarchy dimension

A dimension defines dependencies in a hierarchy. For example, a dimension can be specified to display products by category. You can include multiple dimension levels in a single view.

Hierarchical view configuration options

This form allows configuring the hierarchical view options.

Display records in a new window	If 'Yes', a new window will be opened with the record. Otherwise, it will be displayed in a new page of the same window.
Prune hierarchy	If 'Yes', hierarchy nodes that have no children and do not belong to the target table will not be displayed.
Display orphans	If 'Yes', hierarchy nodes without a parent will be displayed.
Display root node	If 'No', the root node of the hierarchy will not be displayed in the view.
Root node label	Localized label of the hierarchy root node.
Toolbar on top of hierarchy	Allows to set the toolbar on top of the hierarchy.
Display non-matching children	In a recursive case, when a search filter is applied, allows the display of non-matching children of a matching node during a search.
Remove recursive root leaves	In a recursive case, when a search filter is applied or if the mode is 'pruned', removes from the display the root leaves.
Detect cycle	Allow cycle detection and display in a recursive case, the oldest node record will be chosen as the cycle root. Limitation: does not work in search or pruned mode.

Labels

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

Filter

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

See also*Criteria editor* *[p 293]*

Ordering field

In order to enable specifying the position of nodes in a hierarchical view, you must designate an eligible ordering field defined in the table on which the hierarchical view is applied. An ordering field must have the 'Integer' data type and have a 'Hidden' default view mode in its advanced properties in the data model definition.

Except when the ordering field is in 'read-only' mode or when the hierarchy is filtered, any field can be repositioned.

By default, if no ordering field is specified, child nodes are sorted alphabetically by label.

Attention

Do not designate a field that is meant to contain data as an ordering node, as the data will be overwritten by the hierarchical view.

Actions on hierarchy nodes

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

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

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

View sharing

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

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

View publication

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

A published view is then available to all users via Web components, workflow user tasks, data services and perspectives. To publish a view, go to *View* > *Manage views* > *name of the view* > *Publish*.

20.5 **Views management**

Manage recommended views

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

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

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

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

Manage views

The 'Manage views' sub-menu offers the following actions:

Define this view as my favorite	Only available when the currently displayed view is NOT the recommended view. The favorite view will be automatically applied when accessing the table.
Define recommended view as my favorite	Only available when a favorite view has already been defined. This will remove the user's current favorite view. A recommended view, similarly to a favorite view, will be automatically applied when accessing the table. **This menu item is not displayed if no favorite view has been defined.**

20.6 Grid edit

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

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

See also[*Grid edit*](#) *[p 116]*

Copy/paste

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

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

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

20.7 History

The history feature allows tracking changes on master data.

The history feature must have been previously enabled at the data model level. See [Advanced properties for tables](#) [p 61] for more information.

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

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

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

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

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

See also*History* *[p 251]*

CHAPTER 21

Editing data

This chapter contains the following topics:

21.1 Working with records in the user interface

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

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

Creating a record

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

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

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

Updating an existing record

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

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

Duplicating a record

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

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

Deleting records

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

Comparing two records

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

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

21.2 Importing and exporting data

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

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

See also

CSV Services *[p 221]*

XML Services *[p 215]*

21.3 Restore from history

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

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

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

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

> **Note**
>
> This feature has limitations linked to the limitations of the history feature:
>
> - the 'restore from history' feature is not available on tables containing lists that are not supported by history. See Data model limitations [p 256].
> - computed values, encrypted values and fields on which history has been disabled are ignored when restoring a record from history, since these fields are not historized.

See also*History* *[p 251]*

CHAPTER 22

Working with existing datasets

This chapter contains the following topics:

22.1 Validating a dataset

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

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

See Validation [p 300] for detailed information about incremental data validation.

22.2 Duplicating a dataset

To duplicate an existing dataset, select it from the '**Select dataset** [p 109]' ▾ menu in the navigation pane, then select **Actions** > **Duplicate**.

22.3 Deactivating a dataset

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

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

22.4 Managing dataset permissions

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

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

See also*Profile* *[p 23]*

Profile	Defines the profile to which these permissions apply.
Restriction policy	If 'Yes', indicates that when the permissions defined here are more strict than otherwise defined, these permissions are respected. This is contrary to the default where the most permissive rights defined take precedence. See Resolving user-defined rules [p 283].
Dataset actions	Specifies the permissions for actions on the dataset.
Create a child dataset	Indicates whether the profile can create a child dataset. Inheritance also must be activated in the data model.
Duplicate the dataset	Indicates whether the profile can duplicate the dataset.
Delete the dataset	Indicates whether the profile can delete the dataset.
Activate/deactivate the dataset	Indicates whether the profile can modify the *Activated* property in the dataset information. See Deactivating a dataset [p 123].
Create a view	Indicates whether the profile can create views and hierarchies in the dataset.
Tables policy	Specifies the default permissions for all tables. Specific permissions can also be defined for a table by clicking the '+' button.
Create a new record	Indicates whether the profile can create records in the table.
Overwrite inherited record	Indicates whether the profile can override inherited records in the table. This permission is useful when using dataset inheritance.
Occult inherited record	Indicates whether the profile can occult inherited records in the table. This permission is useful when using dataset inheritance.
Delete a record	Indicates whether the profile can delete records in the table.
Values access policy	Specifies the default access permissions for all the nodes of the dataset and allows the definition of permissions for

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

CHAPTER 23

Dataset inheritance

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

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

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

See also*Data model configuration* *[p 42]*

This chapter contains the following topics:

1. Dataset inheritance structure
2. Value inheritance

23.1 Dataset inheritance structure

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

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

23.2 Value inheritance

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

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

The ⮡ button can be used to override a value.

Record inheritance

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

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

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

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

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

See also*Record lookup mechanism* *[p 272]*

Workflow models

CHAPTER **24**

Introduction to workflow models

This chapter contains the following topics:

24.1 **Overview**

What is a workflow model?

Workflows in TIBCO EBX facilitate the collaborative management of data in the repository. A workflow can include human actions on data and automated tasks alike, while supporting notifications on certain events.

The first step of realizing a workflow is to create a *workflow model* that defines the progression of steps, responsibilities of users, as well as other behavior related to the workflow.

Once a workflow model has been defined, it can be validated and published as a *workflow publication*. Data workflows can then be launched from the workflow publication to execute the steps defined in the workflow model.

See also

Workflow model (glossary) [p 29]

Data workflow (glossary) [p 31]

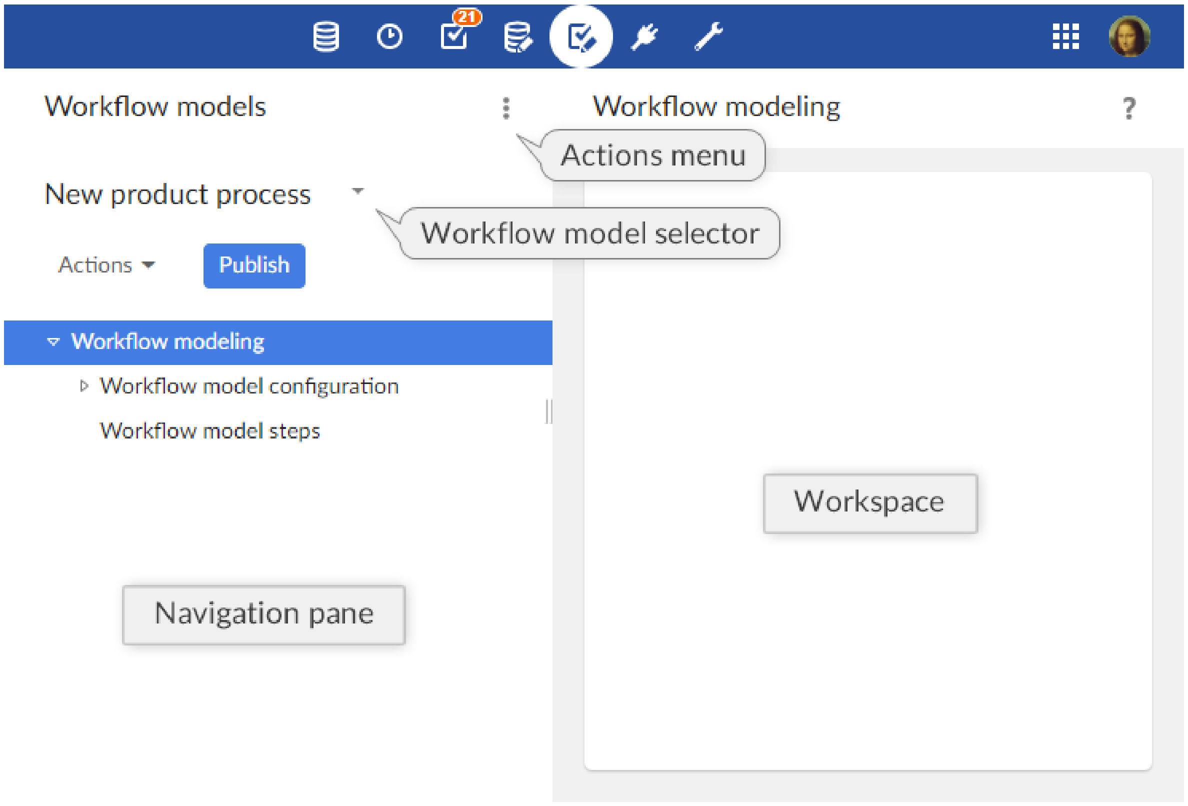
Basic concepts related to workflow models

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

- script task [p 30]
- user task [p 30]
- work item [p 31]
- workflow condition [p 30]
- sub-workflow invocation [p 30]

- wait task [p 30]
- data context [p 30]

24.2 Using the Workflow Models area user interface



Note

This area is available only to authorized users in the 'Advanced perspective'. Only authorized users can access these interfaces.

24.3 Generic message templates

Notification emails can be sent to inform users of specific events during the execution of data workflows.

Generic templates can be defined and reused by any workflow model in the repository. To work with generic templates, select 'Message templates' from the Workflow Models area **Actions** menu.

These templates, which are shared by all workflow models, are included statically at workflow model publication. Thus, in order to take template changes into account, you must update your existing publication by re-publishing the affected models.

Please note that, if you want to export those templates in an archive, you will have to select the dataset "configuration" as it is the one containing the message templates.

When creating a new template, two fields are required:

- **Label & Description**: Specifies the localized labels and descriptions associated with the template.
- **Message**: Specifies the localized subjects and bodies of the message.

The message template can include data context variables, such as `${variable.name}`, which are replaced when notifications are sent. System variables that can be used include:

system.time	System time of the repository.
system.date	System date of the repository.
workflow.lastComment	Last comment on the previous user task. (Note: this variable refers to the last user task, not the current one. Also the current task is the one on which the workflow is positioned, and it also includes the completion notification of a user task).
workflow.lastDecision	Last decisions made on the previous user task. (Note: this variable refers to the last user task, not the current one. Also the current task is the one on which the workflow is positioned, and it also includes the completion notification of a user task).
user.fullName	Full name of the notified user.
user.login	Login of the notified user.
workflow.process.label	Label of the current workflow.
workflow.process.description	Description of the current workflow.
workflow.workItem.label	Label of the current work item.
workflow.workItem.description	Description of the current work item.
workflow.workItem.offeredTo	Role to which the current work item has been offered.
workflow.workItem.allocatedTo	User to whom the current work item has been allocated.
workflow.workItem.link	Link to access the current work item in the work item inbox, using the Web Component API. This link can only be computed if a current work item is defined and if the URL is configured in Workflow-executions, in the email configuration.

workflow.workItem.link.allocateAndStart	Link to access the current work item in the work item inbox, using the Web Component API. If the target work item has not yet been started, it will be automatically allocated to and started by the user clicking the link. This link can only be computed if a current work item is defined and if the URL is configured in Workflow-executions, in the email configuration.
workflow.currentStep.label	Label of the current step.
workflow.currentStep.description	Description of the current step.

Example

Generic template message:

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

Resulting email:

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

24.4 Limitations of workflows

The following functionality is currently unsupported in EBX:

- **Scheduled tasks**, task executed as soon as its turn comes, and whose execution cannot be delayed.
- **Event tasks**, allowing the workflow to move forward upon receiving an event, such as a web service call.
- **Time limitation** on a task duration.

Related concepts*Data workflows* *[p 160]*

CHAPTER 25

Creating and implementing a workflow model

This chapter contains the following topics:

25.1 Creating a workflow model

A new workflow model can be created in the **Workflow Models** area. The only required information at creation is a name that is unique in the repository.

The steps of the workflow model are initialized with a single transition. In order to fully implement the workflow model, you must define the sequence of steps beyond this initial transition.

25.2 Implementing the steps

A workflow model defines steps that correspond to different operations that must be performed on data, and associated conditions. The following types of steps exist:

- User task
- Script task
- Condition
- Sub-workflow invocation
- Wait task

A data context is linked to each data workflow. This data context can be used to define variables that can be used as input and output variables in the steps of the workflow.

Progress strategy of the next step

For each step type (excluding sub-workflows invocations), a property is available to define which progress strategy has to be applied for the next step. Upon step completion, this strategy is evaluated in order to define the navigation when the workflow is executed. By default, the progress strategy is set to 'Display the work items table'. In that case, after the step has been executed, the work items table (work items inbox or monitoring > work items) is automatically displayed, in order to select the following work item.

Another strategy can be selected: 'Automatically open the next step'. This strategy allows the user to keep working on this workflow and to directly execute the next step. If, following to this execution, a work item is reached and the connected user can start it, then the work item is automatically opened (if several work items are reached, the first created is opened). Otherwise, the next step progress strategy is evaluated. If no work item has been reached, the work items table will be displayed.

This strategy is used to execute several steps in a row without going back to the work items inbox.

There are some limitations that will lead to disregard this strategy. In that case, the work items table is automatically displayed. This property will be disregarded when: the next step is a sub-workflow; or the current step is a user task with more than one work item.

In the case of conditions, two other strategies are available: 'If true, automatically open the next step' and 'If false, automatically open the next step'. These strategies allow choosing which strategy will be applied according to the condition result.

Hidden in graphical view

For each step type, a property is available to define which steps must be hidden in the workflow progress view by default.

If this property is enabled, the step will be automatically hidden in the workflow progress view for non-administrator users (neither built-in administrator nor workflow administrator). Hidden steps can be displayed on demand.

25.3 **User tasks**

User tasks are steps that involve actions to be performed by human users. Their labels and descriptions can be localized.

Mode

For backward compatibility reasons, two user task modes are available: the default mode and the legacy mode.

By default, a user task generates a single work item. This mode offers more features, such as offering a work item to a list of profiles or directly displaying the avatars in the workflow progress view.

In the legacy mode, a user task can generate several work items.

By default, the user task creation service is hidden in legacy mode. To display it, a property should be enabled in the `ebx.properties` file. For more information, see Disabling user task legacy mode [p 360].

List of profiles

The definition of the profiles of a user task may vary depending on the user task mode.

[Default] Offered to the following profiles

The defined profiles are the roles or the users to whom the user task is being offered. When executing the user task, a single work item is generated. If a single user is defined, the work item is automatically assigned to this user. If a role is defined, the work item is offered to the members of the role. If several users and roles are defined, the work item is offered simultaneously to these users and to the members of these roles.

[Legacy mode] Participants

The participants are the roles or the users to whom the user task is intended. By default, when executing the user task, a work item is generated by profile. If a profile refers to a user instead of a role, the work item is directly allocated to that user. If a profile is a role, the work item is offered to the members of the role.

For more information

See also*Extension* *[p 141]*

Service

TIBCO EBX includes the following built-in services:

- Access a dataspace
- Access data (default service)
- Access the dataspace merge view
- Compare contents
- Create a new record
- Duplicate a record.
- Export data to a CSV file
- Export data to an XML file
- Import data from a CSV file
- Import data from an XML file
- Merge a dataspace
- Validate a dataspace, a snapshot or a dataset

See also*EBX built-in services* *[p 201]*

Configuration

Main options > Enable reject

By default, only the *accept* action is offered to the user when saving a decision.

It is possible to also allow users to reject the work item by setting this field to 'Yes'.

Main options > Enable confirmation request

By default, a confirmation request is displayed after user task execution when the user saves the decision by clicking the 'Accept' or 'Reject' button.

To disable this confirmation prompt, set this field to 'Yes'.

Main options > Enable comments

By default, comments are enabled. When a work item is open, a 'Comments' button is displayed and allows the user to enter a comment.

It is possible to hide this 'Comments' button by setting this property to *No*.

Main options > Comments required

By default, it is optional to submit a comment associated with a work item.

It is possible to require the user to enter a comment before saving the decision by setting this field to the desired validation criteria. Comments can be set to be always required, required only if the work item has been accepted, or required only if the work item has been rejected.

Main options > Customized labels

When the user task is run, the user can accept or reject the work item by clicking the corresponding button. In the workflow model, it is possible for a user task to define a customized label and confirmation message for these two buttons. This can be used to adapt the buttons to a more specific context.

[Legacy mode] Termination > Task termination criteria

A single user task could be assigned to multiple *participants* and thus generate multiple work items during workflow execution. When defining a user task in the workflow model, you can select one of the predefined methods for determining whether the user task is finished, based on the statuses of its component work items. When the user task's exit requirement has been satisfied, the data workflow will move on to the next step defined in its model.

For example, for the case of a user task where a product record needs to be approved, you could designate three potential participants. The task termination criteria can specify whether the product record needs to be approved by all three users, or only the first user to respond.

The default user task termination criteria is 'When all work items have been accepted.'

> **Note**
>
> If you specify a service extension overriding the method `UserTask.handleWorkItemCompletion` to handle the determination of the user task's completion, it is up to the developer of the extension to verify from within their code that the task termination criteria defined through the user interface has been met. See `UserTask.handleWorkItemCompletion`API in the JavaDoc for more information

[Legacy mode] Termination > Reject tolerance

By default, if a user rejects a work item during workflow execution, the user task is placed into an error state and the workflow progress is halted. When the user task is in an error state, a workflow administrator must intervene by replaying the step where the error occurred in order to continue the workflow execution.

In order to change this default behavior, it is possible to define a certain number of work item rejections to tolerate. While within the limit of tolerated rejections, no error will occur and it is the task termination criteria that determines when to end the user task.

The following task termination criteria automatically tolerate all rejections:

- 'When all work items have been either accepted or rejected'
- 'Either when all work items have been accepted, or as soon as one work item has been rejected'

Extension

A custom class can be specified in order for the task to behave dynamically in the context of a given data workflow. For example, this can be used to create work items or complete user tasks differently than the default behavior.

The specified rule is a JavaBean that must extend the `UserTask`API class.

Attention

If a rule is specified and the `handleWorkItemCompletion` method is overridden, the completion strategy is no longer automatically checked. The developer must check for completion within this method.

Notification

A notification email can be sent to users when specific events occur. For each event, you can specify a content template.

It is possible to define a monitor profile that will receive all emails that are sent in relation to the user task.

See also*Generic message templates* *[p 133]*

Reminder

Reminder emails for outstanding offered or allocated work items can be periodically sent to the concerned users. The recipients of the reminder are the users to whom the work item is offered or allocated, as well as the recipients on copy.

The content of the reminder emails is determined by the current state of the work item. That is, if the work item is offered, the notification will use the "Offered work items" template; if the work item is allocated, the notification will use the "Allocated work items" template.

Deadline

Each user task can have a completion deadline. If this date passes and associated works items are not completed, a notification email is sent to the concerned users. This same notification email will then be sent daily until the task is completed.

There are two deadline types:

- *Absolute deadline*: A calendar date.
- *Relative deadline*: A duration in hours, days or months. The duration is evaluated based on the reference date, which is the beginning of the user task or the beginning of the workflow.

25.4 Script tasks

Script tasks are automatic tasks that are performed without human user involvement.

Two types of script tasks exist, which, once defined, can be used in workflow model steps:

Library script task	EBX includes a number of built-in library script tasks, which can be used as-is. Any additional library script tasks must be declared in a `module.xml` file. A library script task must define its label, description and parameters. When a user selects a library script task for a step in a workflow model, its associated parameters are displayed dynamically.
Specific script task	Specifies a Java class that performs custom actions. The associated class must belong to the same module as the workflow model. Its labels and descriptions are not displayed dynamically to users in workflow models.

Packaging TIBCO EBX modules [p 459]

Library script tasks

EBX includes the following built-in library script tasks:

- Create a dataspace
- Create a snapshot
- Merge a dataspace
- Import an archive
- Close a dataspace
- Set a data context variable
- Send an email
- Delete records (Note: this script can remove several records)

Library script tasks are classes that extend the class `ScriptTaskBean`API. Besides the built-in library script tasks, additional library script tasks can be defined for use in workflow models. Their labels and descriptions can be localized.

The method `ScriptTaskBean.executeScript`API is called when the data workflow reaches the corresponding step.

Attention

The method `ScriptTaskBean.executeScript`API must not create any threads because the data workflow moves on as soon as the method is executed. Each operation in this method must therefore be synchronous.

See the example [p 550].

It is possible to dynamically set variables of the library script task if its implementation follows the Java Bean specification. Variables must be declared as parameters of the bean of the library script task in `module.xml`. The workflow data context is not accessible from a Java bean.

> **Note**
>
> Some built-in library script tasks are marked as "deprecated" because they are not compatible with internationalization. It is recommended to use the new script tasks that are compatible with internationalization.

Specific script tasks

Specific script tasks are classes that extend the class Sample of ScriptTask [p 550].

The method `ScriptTask.executeScript`API is called when the data workflow reaches the corresponding step.

> **Attention**
>
> The method `ScriptTask.executeScript`API must not create any threads because the data workflow moves on as soon as the method is executed. Each operation in this method must therefore be synchronous.

See the example [p 550].

It is not possible to dynamically set the variables of the bean for specific script tasks. However, the workflow data context is accessible from the Java bean.

25.5 Conditions

Conditions are decision steps in workflows.

Two types of conditions exist, which, once defined, can be used in workflow model steps:

Library condition	EBX includes a number of built-in library conditions, which can be used as-is. Any additional library script tasks must be declared in a `module.xml` file. A library condition must define its label, description and parameters. When a user selects a library condition for a step in a workflow model, its associated parameters are displayed dynamically.
Specific condition	Specifies a Java class that implements a custom condition. The associated class must belong to the same module as the workflow model. Its labels and descriptions are not displayed dynamically to users in workflow models.

Packaging TIBCO EBX modules [p 459]

Library conditions

EBX includes the following built-in library conditions:

- Dataspace modified?
- Data valid?
- Last user task accepted?
- Value null or empty?
- Values equals?

Library conditions are classes that extend the class `ConditionBean`API. Besides the built-in library conditions, additional library conditions can be defined for use in workflow models. Their labels and descriptions can be localized.

See the example [p 553].

It is possible to dynamically set variables of the library condition if its implementation follows the Java Bean specification. Variables must be declared as parameters of the bean of the library condition in `module.xml`. The workflow data context is not accessible from a Java bean.

Specific conditions

Specific conditions are classes that extend the class `Condition`API.

See the example [p 552].

It is not possible to dynamically set the variables of the bean for specific conditions. However, the workflow data context is accessible from the Java bean.

25.6 Sub-workflow invocations

Sub-workflow invocation steps in workflow models put the current workflow into a waiting state and invoke one or more workflows.

It is possible to include another workflow model definition in the current workflow by invoking it alone in a sub-workflow invocation step.

If multiple sub-workflows are invoked by a single step, they are run concurrently, in parallel. All sub-workflows must be terminated before the original workflow continues onto the next step. The label and description of each sub-workflow can be localized.

Two types of sub-workflow invocations exist:

Static	Defines one or more sub-workflows to be invoked each time the step is reached in a data workflow. For each sub-workflow, it is possible to set its localized labels and descriptions, as well as the input and output variable mappings in its data context. This mode is useful when the sub-workflows to be launched and the output mappings are predetermined.
Dynamic	Specifies a Java class that implements a custom sub-workflow invocation. All workflows that could be potentially invoked as sub-workflows by the code must be declared as dependencies. The workflow data context is directly accessible from the Java bean. Dynamic sub-workflow invocations must be declared in a `module.xml` file. This mode is useful when the launch of sub-workflows is conditional (for example, if it depends on a data context variable), or when the output mapping depends on the execution of the sub-workflows.

25.7 Wait tasks

Wait task steps in workflow models put the current workflow into a waiting state until a specific event is received.

When a wait task is reached, the workflow engine generates a unique resume identifier associated with the wait task. This identifier will be required to resume the wait task, and as a consequence the associated workflow.

A wait task specifies which profile is authorized to resume the wait task; and a Java class that implements a wait task bean: `WaitTaskBean`API.

The workflow data context is directly accessible from the Java bean.

Wait task beans must be declared in a `module.xml` file.

First, the wait task bean is called when the workflow starts waiting. At this time, the generated resume identifier is available to call a web service for example. Then, the wait task bean is called when the wait task is resumed. In this way, the data context may be updated according to the received parameters.

> **Note**
>
> The built-in administrator always has the right to resume a workflow.

25.8 **Visualizing the workflow diagram**

About

Once a workflow model is defined, one can view the model in a BPMN-like diagram.

This service is available with a dedicated button on the toolbar of the hierarchical view. The icon is the following: .

This service provides a view with limited edition capabilities: it is only possible to modify existing steps, but links edition and creation of steps still need to be done through the hierarchical view. This diagram can help modelers have a clear view of the workflow model they are designing.

Please also note that, although the diagram is derived from BPMN standards, it is not a strict representation of BPMN since EBX workflow concepts are slightly different.

Saving the layout

It is possible to save the modified layout. Please note that this is not a user-based save: it will be shared by all the users.

Actions

Export as PNG	Creates a PNG image.
Export as SVG	Creates an SVG image.
Export as PDF	Creates a one-page PDF

View

Layout > Default layout	Applies the default layout to the diagram.
Grid > Show/Hide grid	Shows the grid if the grid is not visible, hides it otherwise.

Buttons

Save layout	Saves the current layout.
Save layout and close	Saves the current layout and closes the service.
Revert	Reverts changes and reloads a previously saved layout.
Close	Closes the service.

Additional features

The diagram view offers useful additional features

Undo last action	CTRL + Z
Zoom in/Zoom out.	Mouse middle button then mouse wheel / CTRL then mouse wheel.
Multiple selection	Click on the nodes or links selected holding down the CTRL button / Draw a selection rectangle (you will need to hold down the left click for 1 second before drawing the area).
Customizing links drawing	When clicking on a link, you can either move the segments by dragging the squares which appear on the corners, or separate a specific segment by moving the circle in the middle.
Edit a step	Double clicking on a step will display an edition form.
Overview	A panel is now available with a miniature workflow diagram view which can be used to navigate within it. This panel can be collapsed, expanded and dragged inside the area allocated to the workflow diagram view.

CHAPTER 26

Configuring the workflow model

This chapter contains the following topics:

26.1 Information associated with a workflow model

To view and edit the owner and documentation of your workflow model, select 'Information' from the workflow model 'Actions' [p 133] menu for your workflow model in the navigation pane.

Owner	Specifies the workflow model owner, who will have the rights to edit the workflow model's information and define its permissions.
Localized documentation	Localized labels and descriptions for the workflow model.
Activated	*This property is deprecated.* Whether the workflow model is activated. A workflow model must be activated in order to be able to be published.

26.2 Workflow model properties

Configuration for a workflow model is accessible in the navigation pane under 'Workflow model configuration'.

Module name	Module containing specific Java resources (user task extensions, specific scripts and conditions).
Notification of start	The list of profiles to which to send notifications, based on a template, when a data workflow is launched. See Generic message templates [p 133].
Notification of completion	The list of profiles to which to send notifications, based on a template, when a data workflow is completed. The notification is only sent if the workflow has been completed under normal circumstances, that is, not due to an administration action. See Generic message templates [p 133].
Notification of error	The list of profiles that will receive notifications, based on a template, when a data workflow is in error state. See Generic message templates [p 133].
Priority	By default, each workflow associated with this model will be launched with this priority. Setting a priority is optional. If no priority is defined here, and a default priority is set for the repository, the repository default priority will be used for any associated workflow with no priority assigned. See Work item priorities [p 171] for more information. **Note:** Only users who are defined as workflow administrators will be able to manually modify the priority level of any associated data workflows.
Activate quick launch	By default, when a workflow is launched, the user is prompted to enter a documentation for the new workflow in an intermediate form. This documentation is optional. Setting the 'Activate quick launch' property to 'Yes' allows skipping this documentation step and proceeding directly to the workflow launch.
Automatically open the first step	Allows determining the navigation after a workflow is launched. By default, once a workflow is launched, the current table (workflow launchers or monitoring > publications) is automatically displayed.

	Enabling this property will allow the workflow user to keep working on the launched workflow. If, after the first workflow step is executed, a work item is reached, and this work item can be started by the workflow creator, then the work item is automatically opened (if several work items are reached, the first created is opened). This will save the user from selecting the corresponding work item from the work items inbox. If no work item has been reached, the next step progress strategy is evaluated. If no work item has been opened, the table from which the workflow has been launched is displayed. **Limitation:** This property will be ignored if the first step is a sub-workflow invocation.
Workflow trigger	Component that intercepts the main events of a workflow. This bean must be declared in a `module.xml` file. See the example [p 556].
Permissions	Permissions on actions related to the data workflows associated with the workflow model. This bean must be declared in a `module.xml` file. See the example [p 555].
Programmatic action permissions	Defines a custom component that handles the permissions of the workflow. If set, this overrides all permissions defined in the property 'Permissions'.

26.3 Data context

The data context configuration can be accessed from the navigation pane.

Each workflow has its own data context, thus allowing to have its own local dataspace during its execution. This gives the possibility to store and to vary values that will direct the workflow execution.

The data context is defined by a list of variables. Each variable has the following properties:

Name	Identifier of the variable.
Default value	If defined, the variable will be initialized with this default value.
Input parameter	'Yes' must be checked in order to define this variable as an input parameter.
Output parameter	'Yes' must be checked in order to define this variable as an output parameter. Else, this variable will not be displayed in the list of output parameters, in the task definition interface.

26.4 Custom workflow execution views

The workflow execution views customization can be accessed from the navigation pane.

The customization allows configuring the specific columns of the work items and workflow views (inbox, work items monitoring, active workflows monitoring and completed workflows). For each specific column, it is possible to associate an expression that can contain data context variables that will be evaluated upon display of the workflow.

26.5 Permissions on associated data workflows

Workflow administration	Defines the profile that is allowed to perform administration actions on the workflows. The administration actions include the following: replay a step, resume a workflow, terminate a workflow, disable a publication and unpublish. In order to perform these actions, this profile is automatically granted the "Visualize workflows" permission. The built-in administrator always has the workflow administration rights.
Workflow administration > Replay a step	Defines the profile that is allowed to replay a workflow step. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Active workflows" section is available to replay a step. A profile with the "Workflow administration" permission is automatically allowed to perform this specific action. The built-in administrator always has the rights to replay a step.
Workflow administration > Terminate workflow	Defines the profile that is allowed to terminate and clean a workflow. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Active workflows" section is available to terminate and clean an active workflow. A button in the "Completed workflows" section is available to delete a completed workflow. A profile with the "Workflow administration" permission is automatically allowed to perform this specific action. The built-in administrator always has the rights to terminate a workflow.
Workflow administration > Force a workflow to resume	Defines the profile that is allowed to force resuming a waiting workflow. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Active workflows" section is available to resume a workflow. A profile with the "Workflow administration" permission is automatically allowed to perform this specific action. The built-in administrator always has the right to resume a workflow.
Workflow administration > Disable a publication	Defines the profile that is allowed to disable a workflow publication. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Publications" section is available to disable a publication. It is only

	displayed on active publications. A profile with the "Workflow administration" permission is automatically allowed to perform this specific action. The built-in administrator always has the rights to disable a publication.
Workflow administration > Unpublish	Defines the profile that is allowed to unpublish a workflow publication. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Publications" section is available to unpublish disabled publications only. A profile with the "Workflow administration" permission is automatically allowed to perform this specific action. The built-in administrator always has the unpublish rights.
Allocation management	Defines the profile that is allowed to manage work items allocation. The allocation actions include the following: allocate work items, reallocate work items and deallocate work items. In order to perform these actions, this profile is automatically granted the "Visualize workflows" permission. The built-in administrator always has the allocation management rights.
Allocation management > Allocate work items	Defines the profile that is allowed to allocate work items. In order to perform these actions, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Work items" section is available to allocate a work item. It is only displayed on offered work items. A profile with the "Allocation management" permission is automatically allowed to perform this specific action. The built-in administrator always has the work items allocation rights.
Allocation management > Reallocate work items	Defines the profile that is allowed to reallocate work items. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Work items" section is available to reallocate a work item. It is only displayed on allocated work items. A profile with the "Allocation management" permission is automatically allowed to perform this specific action. The built-in administrator always has the work items reallocation rights.
Allocation management > Deallocate work items	Defines the profile that is allowed to deallocate work items. In order to perform this action, this profile is automatically granted the "Visualize workflows" permission. A button in the "Monitoring > Work items" section is available to deallocate a work item. It is only displayed on allocated work items. A profile with the "Allocation management" permission is automatically allowed to perform this specific

	action. The built-in administrator always has the work items deallocation rights.
Launch workflows	Defines the profile that is allowed to manually launch new workflows. This permission allows launching workflows from the active publications of the "Workflow launchers" section. The built-in administrator always has the launch workflows rights.
Visualize workflows	Defines the profile that is allowed to visualize workflows. By default, the end-user can only see work items that have been offered or allocated to him in the "Inbox" section. This permission also allows visualizing the publications, workflows and work items associated with this workflow model in the "Monitoring" and "Completed workflows" sections. This profile is automatically granted the "Visualize completed workflows" permission. The built-in administrator always has the visualize workflows rights.
Visualize workflows > The workflow creator can visualize it	If enabled, the workflow creator has the permission to view the workflows he has launched. This restricted permission grants access to the workflows he launched and to the associated work items in the "Monitoring > Active workflows", "Monitoring > Work items" and "Completed workflows" sections. The default value is 'No'.
Visualize workflows > Visualize completed workflows	Defines the profile that is allowed to visualize completed workflows. This permission allows visualizing completed workflows in the "Completed workflows" section and accessing their history. A profile with the "Visualize workflows" permission is automatically allowed to perform this action. The built-in administrator always has the visualize completed workflows rights.

> **Note**
>
> A user who has no specific privileges assigned can only see work items associated with this workflow that are offered or allocated to that user.

See also*Workflow administration* *[p 175]*

26.6 Workflow model snapshots

The history of workflow model snapshots can be managed from **Actions** > **History**.

The history table displays all snapshots which contain the selected workflow model and indicates if a workflow model is published. For each snapshot, the **Actions** button allows you to export or view the corresponding workflow model.

26.7 Deleting a workflow model

Workflow model can be deleted, however any associated publications remain accessible in the Data Workflows area. If a new workflow model is created with the same name as a deleted workflow model, publishing will prompt to replace the old publication.

See also*Publishing workflow models* *[p 157]*

CHAPTER 27

Publishing workflow models

This chapter contains the following topics:

27.1 About workflow publications

Once a workflow model is defined, it must be published in order to enable authorized users to launch associated data workflows. This is done by clicking the **Publish** button in the navigation pane.

If no sub-workflow invocation steps are included in the current workflow model, you have the option of publishing other workflow models at the same time on the publication page. If the current workflow model contains sub-workflow invocation steps, it must be published alone.

Workflow models can be published several times. A publication is identified by its publication name

27.2 Publishing and workflow model snapshots

When publishing a workflow model, a snapshot is taken of its current state. A label and a description can be specified for the snapshot to be created. The default snapshot label is the date and time of the publication. The default description indicates the user who published the workflow model.

For each workflow model being published, the specified publication name must be unique. If a workflow model has already been published, it is possible to update an existing publication by reusing the same publication name. The names of existing workflow publications associated with a given workflow model are available in a drop-down menu. In the case of a publication update, the old version is no longer available for launching data workflows, however it will be used to terminate existing workflows. The content of different versions can be viewed in the workflow model snapshot history.

See also*Workflow model snapshots* *[p 155]*

27.3 Sub-workflows in publications

When publishing a workflow model containing sub-workflow invocation steps, it is not necessary to separately publish the models of the sub-workflows. From an administration standpoint, the model of the main workflow (the one currently published by a user) and the models of the sub-workflows are published as a single entity.

The system computes the dependencies to workflow models used as sub-workflows, and automatically creates one publication for each dependent model. These technical publications are dedicated to the workflow engine to launch sub-workflows, and are not available in the Workflow Data area.

The multiple publication is not available for a workflow model containing sub-workflow invocation steps. This is why the first step of the publication (selection of workflow models to publish) is not offered in this case.

Republishing the main workflow model automatically updates the invoked sub-workflow models.

Although a sub-workflow model can be published separately as a main workflow model, this will not update the version used by an already published main workflow model using this sub-workflow.

Data workflows

CHAPTER **28**

Introduction to data workflows

This chapter contains the following topics:

28.1 Overview

A data workflow is an executed step-by-step data management process, defined using a workflow model publication. It allows users, as well as automated procedures, to perform actions collaboratively on a set of data. Once a workflow model has been developed and published, the resulting publication can be used to launch a data workflow to execute the defined steps.

Depending on the workflow user permissions defined by the workflow model, a user may perform one or more of the following actions on associated data workflows:

- As a user with default permissions, work on and complete an assigned work item.
- As a user with workflow launching permissions, create a new data workflow from a workflow model publication.
- As a workflow monitor, follow the progress of ongoing data workflows and consult the history of completed data workflows.
- As a manager of work item allocation, modify work item allocations manually for other users and roles.
- As a workflow administrator, perform various administration actions, such as replaying steps, terminating workflows in progress, or rendering publications unavailable for launching data workflows.

See also

Work items *[p 167]*

Launching and monitoring data workflows *[p 173]*

Administration of data workflows *[p 175]*

Permissions on associated data workflows *[p 153]*

Related concepts*Workflow models* *[p 132]*

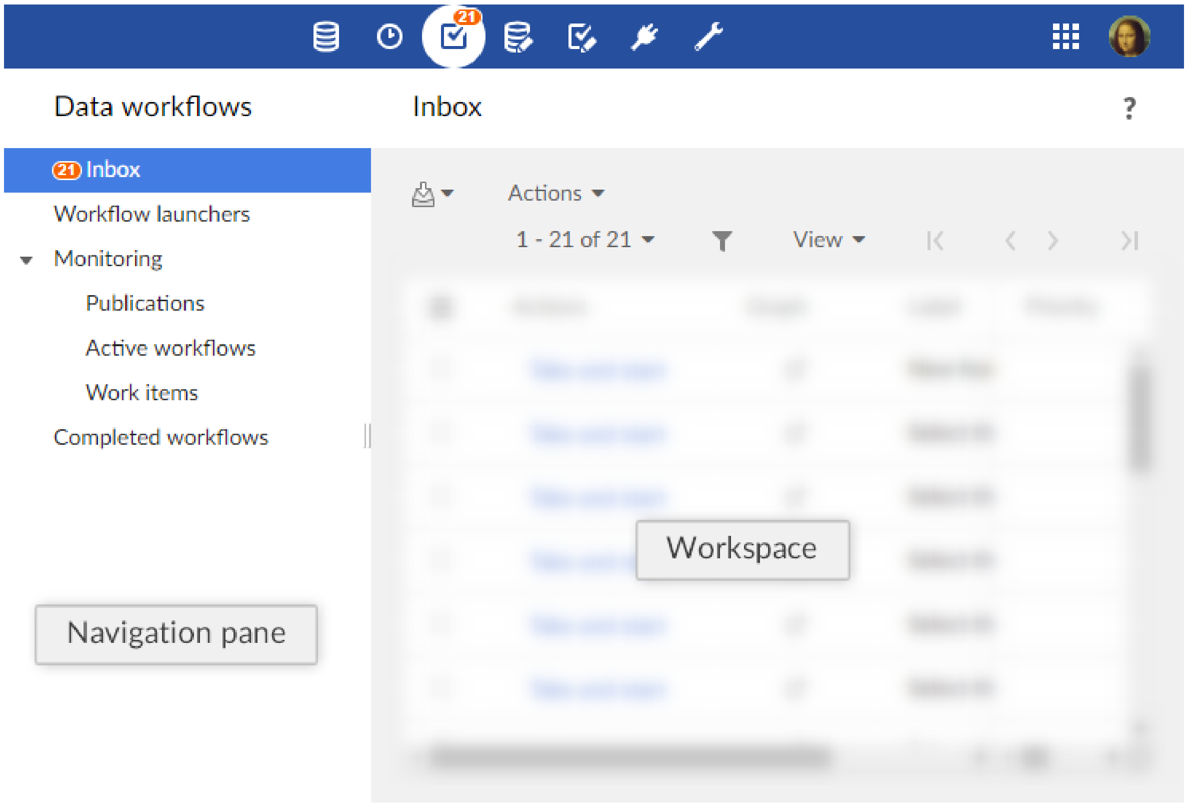
CHAPTER 29

Using the Data Workflows area user interface

This chapter contains the following topics:

29.1 Navigating within the interface

Data workflow functionality is located in the **Data Workflows** area of the TIBCO EBX user interface.



Note

This area is available only to authorized users in the 'Advanced perspective' or from a specifically configured perspective. Only authorized users can access these interfaces.

The navigation pane is organized into several entries. These entries are displayed according to their associated global permission. The different entries are:

Work items inbox	All work items either allocated or offered to you, for which you must perform the defined task.
Workflow launchers	List of workflow model publications from which you are allowed to launch data workflows, according to your user permissions.
Monitoring	Monitoring views on the data workflows for which you have the necessary viewing permissions.
Publications	Publications for which you have the necessary viewing permissions. If you have additional administrative permissions, you can also disable the ability to launch data workflows from specific publications from this view.
Active workflows	Data workflows in the process of execution for which you have the necessary viewing permissions. If you have additional administrative permissions, you can also perform actions such as replaying steps of data workflows, and terminating the execution of data workflows from this view.
Work items	Work items for which you have the necessary viewing permissions. If you have additional administrative permissions, you can also perform actions relevant to work item administration, such as allocating work items to specific users or roles from this view.
Completed workflows	Data workflows that have completed their execution, for which you have the necessary viewing permissions. You can view the history of the executions of the data workflows. If you have additional administrative permissions, you can also clean completed workflows from the repository from this view.

Note

Each section can be accessed through Web Components, for example, for portal integration, or programatically using the `ServiceKey` class in the Java API.

See also

Using TIBCO EBX as a Web Component *[p 193]*

`ServiceKey` API

29.2 Navigation rules

Work items inbox

By default, once a work item has been executed, the work items inbox is displayed.

This behavior can be modified according to the next step progress strategy, which can allow to execute several steps in a row without going back to the work items inbox.

See the progress strategy of a workflow step [p 138] in workflow modeling.

Workflow launchers

By default, once a workflow has been launched, the workflow launchers table is displayed.

This behavior can be modified according to the model configuration, which can allow to directly open the first step without displaying the workflow launchers table.

See the automatic opening of the first workflow step [p 150] in workflow modeling.

29.3 Custom views

It is possible to define views on workflow tables and to benefit from all associated mechanisms (publication included).

Permissions to create and manage workflow table views are the same as the permissions for data table views. It may thus be necessary to change the permissions in the 'Administration' section in order to benefit from this feature, by selecting *Workflow management* > *Workflows*.

See the Views [p 115] for more information.

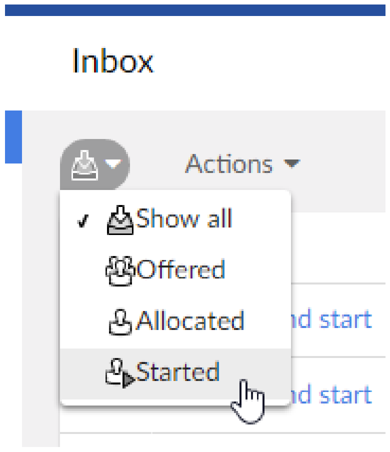
29.4 Specific columns

By default, specific columns are hidden in the views that can benefit from it (inbox, work items monitoring, active workflows monitoring and completed workflows).

A custom view should be created and applied in order to display the specific columns. For each work item or workflow, the matching defined in the associated workflow model is then applied. If an expression is defined for a column and contains data context variables, these variables are evaluated upon display. If the expression contains built-in expressions which depend on the locale, the expression is evaluated in the default locale.

29.5 **Filtering items in views**

In certain tables, such as the 'Work item inbox', you can narrow down the entries displayed in the tables by viewing only the entries of a certain state. In such views, there is a menu from which you can select a state to see its corresponding items.



29.6 **Graphical workflow view**

Whether as a user with a work item to perform, or as a data workflow monitor or administrator, you can view the progress or the history of a data workflow execution by clicking the 'Preview' button that appears in the 'Data workflow' column of tables throughout the data workflows user interface. This opens a pop-up displaying an interactive graphical view of the data workflow's execution. In this view, you can see the overall progress of the execution, as well as click on an individual step to view the details of its information.

If steps have been defined as hidden in the workflow modeling, they are automatically hidden in the workflow progress view for non-administrator users (non built-in administrators and non workflow administrators). A button is available to display hidden steps. The choice of users (show or hide steps) is saved by user, by publication during the user session.

For user tasks performed using the new mode (single work item), the main information about the single work item is directly displayed in the workflow progress view, when applicable: the avatar of the user associated with the work item, and the decision that has been taken for the work item (accepted or rejected).

CHAPTER 30

Work items

This chapter contains the following topics:

30.1 About work items

A work item is a unit of work that must be performed by a human user as a part of a user task. By default, when a workflow model defines a user task, data workflows that are launched from that model's publications will generate an individual work item for each of the participants listed in the user task.

See also*Overview* *[p 563]*

Work item states

When the data workflow spawns a work item for a modelled user task during execution, the work item passes through several possible states: offered, allocated, started, and completed.

Creation of work items

Default mode

By default, a single work item is generated regardless of the list of defined profiles.

By default, if a single user is defined in the list of profiles, the created work item is in the *allocated* state.

By default, in other cases, the created work item is in the *offered* state.

> **Note**
>
> The default behavior described above can be overridden by a programmatic extension defined in the user task. In this case, work items may be generated programmatically and not necessarily based on the user task's list of participants.

Legacy mode

By default, for each user defined as a participant of the user task, the data workflow creates a work item in the *allocated* state.

By default, for each role defined as a participant of the user task, the data workflow creates a work item in the *offered* state.

> **Note**
>
> The default behavior described above can be overridden by a programmatic extension defined in the user task. In this case, work items may be generated programmatically and not necessarily based on the user task's list of participants.

Variations of the work item states

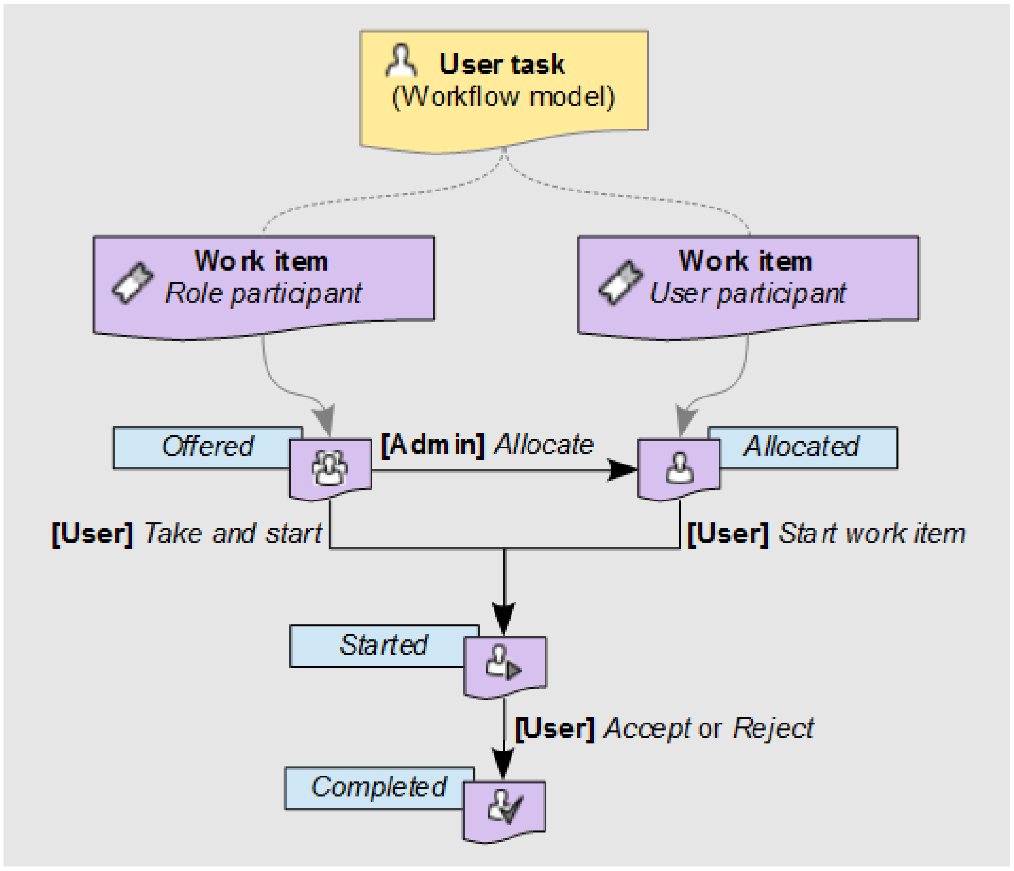
When the work item is in the *allocated* state, the defined user can directly start working on the allocated work item with the 'Take and start' action. The work item's state becomes *started*.

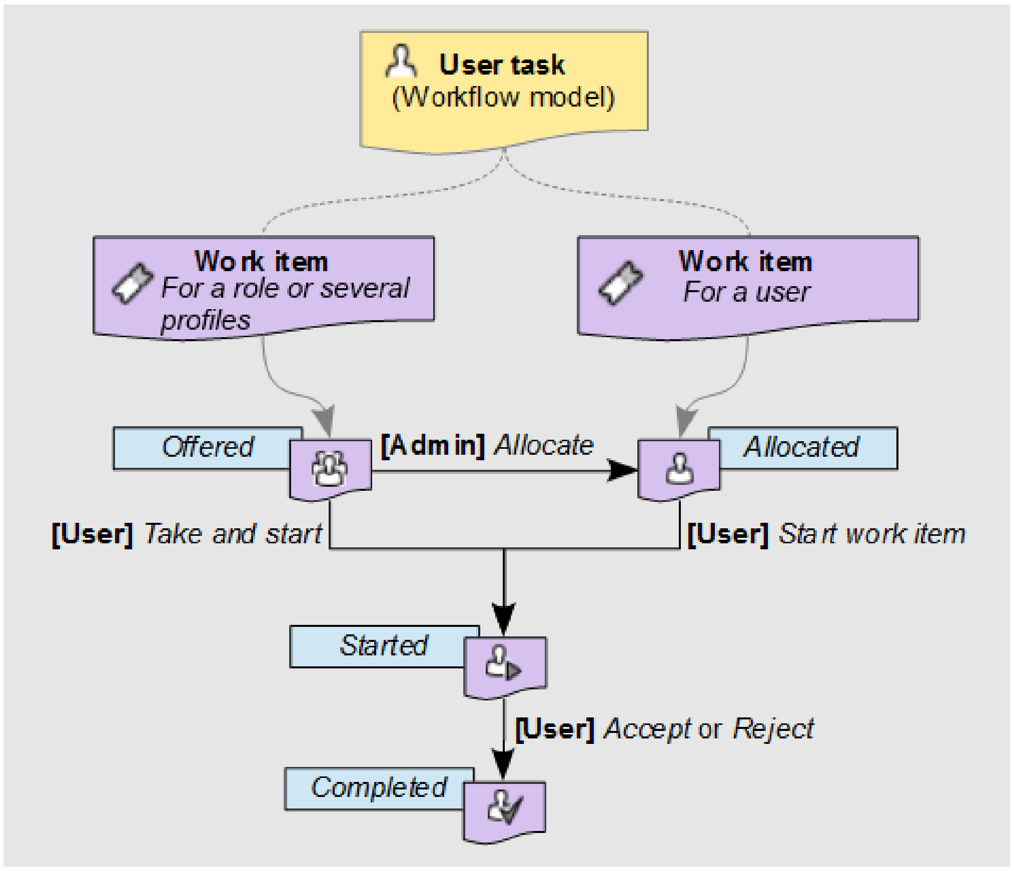
When the work item is in the *offered* state, any user or member of the roles to whom the work item is offered can take the work item with the 'Take and start' action'. The work item's state becomes *started*.

Before a user has claimed the offered work item, a workflow allocation manager can intervene to manually assign the work item to a single user, thus moving the work item to the *allocated* state. Then, when that user begins work on the work item by performing the action 'Start work item', the work item progresses to the *started* state.

Finally, after the user who started the work item has finished the assigned action, the concluding accept or reject action moves the work item to the *completed* state. Once a user completes a work item, the data workflow automatically progresses onto the next step defined in the workflow model.

Diagram of the work item states





30.2 Working on work items as a participant

All work items relevant to you as a user (either offered or allocated to you), appear in your work items inbox. When you begin working on a work item, you can add an associated comment that will be visible to other participants of the data workflow, as well as administrators and monitors of the workflow. As long as you are still working on the work item, you can go back and edit this comment.

After you have performed all the necessary actions assigned for the work item, you must signal its completion by clicking either the **Accept** or **Reject** button. The labels of these two buttons may differ depending on the context of the work item.

To review the current progress of a data workflow for which you have a waiting work item in your work item inbox, click its 'Preview' button in the 'Data workflow' column of the table. A pop-up will show an interactive graphical view of the data workflow up until this point and the upcoming steps. You can view the details of a step by clicking on that step.

Note

If you interrupt the current session in the middle of a started work item, for example by closing the browser or by logging out, the current work item state is preserved. When you return to the work item, it continues from the point where you left off.

30.3 **Work item priorities**

Work items may carry a priority value, which can be useful for sorting and filtering outstanding work items. The priority of a work item is set at the level of its data workflow, rather than being specific to the individual work item itself. Thus, if a data workflow is considered urgent, all its associated open work items are also considered to be urgent. By default, there are six priority levels ranging from 'Very low' to 'Urgent', however the visual representation and naming of the available priority levels depend on the configuration of your TIBCO EBX repository.

See also*user task (glossary)* *[p 30]*

Related concepts*User tasks* *[p 138]*

CHAPTER 31

Launching and monitoring data workflows

This chapter contains the following topics:

31.1 Launching data workflows

If a workflow model has given you the permissions to launch data workflows from its publications, you can create new data workflows from the 'Workflow launchers' entry in the navigation pane. To create a new data workflow from a workflow model publication, click the **Launch** button in the entry of the publication.

You can optionally define localized labels and descriptions for the new data workflow you are launching.

31.2 Monitoring activities

If a workflow model's permissions have configured your user or role for workflow monitoring, you have the ability to follow the progress of data workflows that are currently executing. You can access your monitoring views from the 'Monitoring' section of the navigation panel. If you have additional workflow management permissions, you can also perform the associated actions from these views.

Once the data workflows that you monitor have completed execution, they appear under 'Completed data workflows', where you can consult their execution history.

31.3 Managing work item allocation

If a workflow model defines special allocation management permissions for you or a role that you belong to, you have the ability to manually intervene for work item allocations during the execution of associated data workflows. In this case, you are able to perform one or more of the actions listed below on work items.

Select 'Work items' in the 'Monitoring' section of the navigation pane. The actions that you are able to perform appear in the **Actions** menu of the work item's entry in the table, depending on the current state of the work item.

Allocate	Allocate a work item to a specific user. This action is available for work items in the *offered* state.
Deallocate	Reset a work item in the *allocated* state to the *offered* state.
Reallocate	Modify the user to whom a work item is allocated. This action is available for work items in the *allocated* state.

See also

Work items *[p 167]*

Permissions on associated data workflows *[p 153]*

Related concepts*Workflow models* *[p 132]*

CHAPTER **32**

Administration of data workflows

If you have been given permissions for administration activities associated with data workflows, any relevant publications, active data workflows, and work items, will appear under the entries of the 'Monitoring' section in the navigation panel. From these monitoring views, you can directly perform administrative tasks from the **Actions** menus of the table entries.

> **Note**
>
> When a workflow model gives you administrative rights, you automatically have monitoring permissions on all of the relevant aspects of data workflow execution, such as publications, active data workflows, and work items.

This chapter contains the following topics:

32.1 Overview of data workflow execution

When a data workflow is launched, a *token* that marks the step currently being executed is created and positioned at the start of the workflow. As each step is completed, this token moves on to the next step as defined in the workflow model on whose publication the data workflow is based.

At any given point during the execution of a data workflow, the token is positioned on one of the following:

- a script task, which is run automatically and requires no user interaction. The script task is completed when the defined actions finish running.
- a user task, which spawns one or more work items to be performed manually by users. Each work item is completed by an explicit 'Accept' or 'Reject' action from a user, and the completion of the umbrella user task is determined according to the task termination criteria defined for the user task in the workflow model.
- a condition, which is evaluated automatically in order to determine the next step in the execution of the data workflow.
- a sub-workflows invocation, which launches associated sub-workflows and waits for the termination of the launched sub-workflows.
- a wait task, which pauses the workflow until a specific event is received.

The token can be in the following states:

- **To execute:** The token is the process of progressing to the next step, based on the workflow model.
- **Executing:** The token is positioned on a script task or a condition that is being processed.
- **User:** The token is positioned on a user task and is awaiting a user action.
- **Waiting for sub-workflows:** The token is positioned on a sub-workflow invocation and is awaiting the termination of all launched sub-workflows.
- **Waiting for event:** The token is positioned on a wait task and is waiting for a specific event to be received.
- **Finished:** The token has reached the end of the data workflow.
- **Error:** An error has occurred.

See also*Workflow management [p 409]*

32.2 Data workflow administration actions

Actions on publications

Disabling a workflow publication

To prevent new data workflows from being launched from a given workflow publication, you can disable it. Select the 'Publications' entry from the navigation pane, then select **Actions** > **Disable** in the entry for the publication you want to disable.

Once disabled, the publication will no longer appear in the 'Workflow launchers' view of users, but any data workflows already launched that are in progress will continue executing.

> **Note**
>
> Once a publication has been disabled, it cannot be re-enabled from the Data Workflows area. Only a user with the built-in repository 'Administrator' role can re-enable a disabled publication from the Administration area, although manually editing technical tables is not generally recommended, as it is important to ensure the integrity of workflow operations.

Unpublishing a workflow publication

If a workflow publication is no longer required, you can remove it completely from the views in the Data Workflows area by unpublishing it. To do so,

1. Disable the workflow publication to prevent users from continuing to launch new data workflows from it, as described in Disabling a workflow publication [p 176].
2. Unpublish the workflow publication by selecting **Actions** > **Unpublish** from the workflow publication's entry in the same table of publications.

> **Note**
>
> When you choose to unpublish a workflow publication, you will be prompted to confirm the termination and cleaning of any data workflows in progress that were launched from this workflow publication, and any associated work items. Any data that is lost as a result of forcefully terminating a data workflow cannot be recovered.

Actions on data workflows

From the tables of data workflows, it is possible to perform actions from the **Actions** menu in the record of a given data workflow.

Replaying a step

In the event of an unexpected failure during a step, for example, an access rights issue or unavailable resources, you can "replay" the step as a data workflow administrator. Replaying a step cleans the associated execution environment, including any related work items and sub-workflows, and resets the token to the beginning of the current step.

To replay the current step in a data workflow, select **Actions** > **Replay the step** from the entry of the workflow in the 'Active workflows' table.

Terminating and cleaning an active data workflow

In order to stop and clean a data workflow that is currently in progress, select **Actions** > **Terminate and clean** from the entry of the workflow in the 'Active workflows' table. This will stop the execution of the data workflow and clean the data workflow and all associated work items and sub-workflows.

> **Note**
>
> This action is not available on workflows in the 'Executing' state, and on sub-workflows launched from another workflow.

> **Note**
>
> Workflow history data is not deleted.

Forcing termination of an active data workflow

In order to stop a data workflow that is currently in progress, select **Actions** > **Force termination** from the entry of the workflow in the 'Active workflows' table. This will stop the execution of the data workflow and clean any associated work items and sub-workflows.

> **Note**
>
> This action is available for sub-workflows, and for workflows in error blocked on the last step.

> **Note**
>
> Workflow history data is not deleted.

Forcing resumption of a waiting data workflow

In order to resume a data workflow that is currently waiting for an event, select *Actions* > *Force resumption* from the entry of the workflow in the 'Active workflows' table. This will resume the data workflow. Before doing this action, it is the responsibility of the administrator to update the data context in order to make sure that the data workflow can execute the next steps.

> **Note**
>
> This action is only available for workflows in the 'waiting for event' state.

Cleaning a completed data workflow

When a data workflow has completed its execution, its history is viewable by monitors and administrators of that workflow in the view 'Completed workflows'. To remove the completed workflow, you can perform a clean operation on it. To do so, select **Actions > Clean** from the entry of the workflow in the 'Completed workflows' table.

When cleaned a workflow is no longer visible in the view 'Completed workflows' but its history is still available from the technical administration area.

> **Note**
>
> This action is not available on sub-workflows launched from another workflow.

See also*Workflow management* *[p 409]*

Modifying the priority of a data workflow

After a data workflow has been launched, an administrator of the workflow can alter its priority level. Doing so changes the priority of all existing and future work items created by the data workflow. To change the priority level of a data workflow, select **Actions > Modify priority** from the entry of the workflow in the 'Active workflows' table.

See also*Permissions on associated data workflows* *[p 153]*

Data services

CHAPTER 33

Introduction to data services

This chapter contains the following topics:

33.1 Overview

What is a data service?

A data service [p 31] is:

- a standard Web service that interacts with TIBCO EBX.

  SOAP data services can be dynamically generated based on data models from the 'Data Services' area.
- a REST service that allows interrogating the EBX repository.

  The built-in RESTful service does not require a service interface, it is self-descriptive through the returned metadata.

They can be used to access some of the features available through the user interface.

See also

WSDL/SOAP [p 596]

REST [p 650]

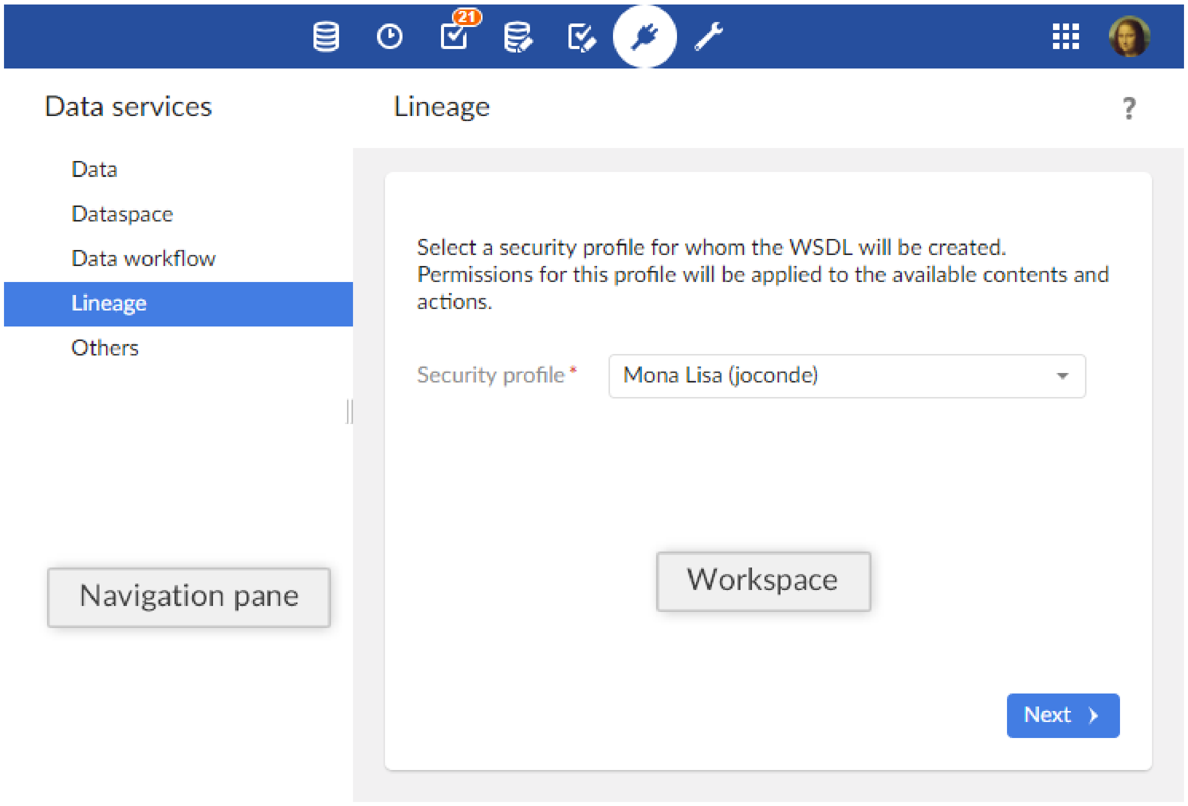
Lineage

Lineage [p 32] is used to establish user permission profiles for non-human users, namely data services. When accessing data using WSDL interfaces, data services use the permission profiles established through lineage.

Glossary

See also*Data services [p 31]*

33.2 Using the Data Services area user interface



Select a security profile for whom the WSDL will be created. Permissions for this profile will be applied to the available contents and actions.

Security profile * Mona Lisa (joconde)

Note

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

Related concepts

Dataspace [p 90]

Dataset [p 108]

Data workflows [p 160]

Introduction [p 596]

CHAPTER 34

Generating data service WSDLs

This chapter contains the following topics:

34.1 Generating a WSDL for operations on data

To generate a WSDL for accessing data, select 'Data' in the navigation panel in the **Data Services** area, then follow through the steps of the wizard:

1. Choose whether the WSDL will be for operations at the dataset level or at the table level.
2. Identify the dataspace and dataset on which the operations will be run
3. Select the tables on which the operations are authorized, as well as the operations permitted.
4. Download the generated WSDL file by clicking the button **Download WSDL**.

Operations on datasets

The following operations can be performed using the WSDL generated for operations at the dataset level:

- Select dataset content for a dataspace or snapshot.
- Get dataset changes between dataspaces or snapshots
- Replication unit refresh

Operations on tables

The following operations, if selected, can be performed using the WSDL generated for operations at the table level:

- Insert record(s)
- Select record(s)

- Update record(s)
- Delete record(s)
- Count record(s)
- Get changes between dataspace or snapshot
- Get credentials
- Run multiple operations on tables in the dataset

See also

WSDL download from HTTP protocol *[p 609]*

Operations generated from a data model *[p 615]*

34.2 Generating a WSDL for dataspace operations

To generate a WSDL for dataspace-level operations, selecting 'Dataspace' in the navigation panel of the **Data Services** area. The generated WSDL is generic to all dataspaces, thus no additional information is required.

Download the generated WSDL file by clicking the button **Download WSDL**.

Operations on dataspaces

The following operations can be performed using the WSDL generated for operations at the dataspace level:

- Create a dataspace
- Close a dataspace
- Create a snapshot
- Close a snapshot
- Merge a dataspace
- Lock a dataspace
- Unlock a dataspace
- Validate a dataspace or a snapshot
- Validate a dataset

See also

WSDL download from HTTP protocol *[p 609]*

Operations on datasets and dataspaces *[p 636]*

34.3 Generating a WSDL for data workflow operations

To generate a WSDL to control data workflows, select 'Data workflow' from the **Data Services** area. The generated WSDL is not specific to any particular workflow publication, thus no additional information is required.

Download the generated WSDL file by clicking the button **Download WSDL**.

Operations on data workflows

- Start a data workflow
- Resume a data workflow
- End a data workflow

See also

WSDL download from HTTP protocol [p 609]

Operations on data workflows [p 642]

34.4 Generating a WSDL for lineage

To generate a WSDL for lineage, select 'Lineage' from the **Data Services** area. It will be based on authorized profiles that have been defined by an administrator in the 'Lineage' section of the **Administration** area.

The operations available for accessing tables are the same as for WSDL for operations on data [p 183].

Steps for generating the WSDL for lineage are as follows:

1. Select the profile whose permissions will be used. The selected user or role must be authorized for use with lineage by an administrator.
2. Identify the dataspace and dataset on which the operations will be run
3. Select the tables on which the operations are authorized, as well as the operations permitted.
4. Download the generated WSDL file by clicking the button **Download WSDL**.

See also*Lineage* [p 180]

34.5 Generating a WSDL for administration

This action is only available to administrators.

To generate a WSDL for:

- managing the user interface
- getting system information

select 'Administration' from the **Data Services** area.

Operations for administration

- Close user interface
- Open user interface
- Get system information

See also

WSDL download from HTTP protocol [p 609]

User interface operations [p 646]

System information operation [p 646]

34.6 **Generating a WSDL to modify the default directory**

This action is only available to administrators, and only if using the default directory.

To generate a WSDL to update the default directory, select 'Directory' from the **Data Services** area.

Operations on the default directory

The operations available for accessing tables are the same as for WSDL for operations on data [p 183].

See also

WSDL download from HTTP protocol [p 609]

Directory services [p 645]

Reference Manual

Integration

CHAPTER **35**

Overview of integration and extension

Several service and component APIs allow you to develop custom extensions for TIBCO EBX and integrate it with other systems.

This chapter contains the following topics:

35.1 Using EBX as a Web Component

It is possible to use EBX as a user interface web component, by calling it using the HTTP protocol. Such EBX Web Components can be integrated into any application that is accessible through a supported web browser [p 310].

A typical use is to integrate EBX views into an organization's intranet framework. Web Components can also be invoked from the EBX user interface using User services [p 189].

See also*Using EBX as a Web Component* *[p 193]*

35.2 User interface customization

User services

A user service is an extension of EBX that provides a graphical user interface (GUI) allowing users to access specific or advanced functionalities.

See also*User services overview* *[p 563]*

Custom form layout

A presentation layer provides the ability to override the default layout of forms in the user interface with highly customized form layouts.

See also*Form layout* *[p 561]*

Custom widgets

A custom widget is a graphical component developed specifically to customize the look and feel of groups and fields in data models or in programmatically defined schemas.

See also*Custom widgets* *[p 561]*

Ajax

EBX supports Ajax asynchronous exchange of data with the server without refreshing the currently displayed page.

See also

User service Ajax callbacks *[p 560]*

Ajax component `UIAjaxComponent`API

Specifying UI filters on a table

In addition to the default filters and search panes in the user interface, it is possible to specify additional filters dependent on the structure of the table. For that, a specific class must be defined in the definition of the table and must extend `UITableFilter`.

See `UITableFilter`API for more information.

35.3 Data services

The data services module provides a means for external systems to interact with EBX using one of following:

- Web Services Description Language (WSDL/SOAP) standard
- Representational state transfer (REST)

See also

WSDL/SOAP data services *[p 596]*

REST data services *[p 650]*

35.4 XML and CSV import/export services

EBX includes built-in services for importing data from and export data to XML and CSV formats. Imports and exports for XML and CSV can be performed using the user interface, data services, or the Java API.

See also

XML import and export *[p 215]*

CSV import and export *[p 221]*

35.5 **Programmatic services**

Programmatic services allow executing procedures in a well-defined context, for example in a scheduled task or in a batch.

Some examples of programmatic services include:

- Importing data from an external source,
- Exporting data to multiple systems,
- Data historization, launched by a supervisory system
- Optimizing and refactoring data if EBX **built-in optimization services** `AdaptationTreeOptimizerSpec`API are not sufficient.

See also*`ProgrammaticService`*API

CHAPTER 36

Using TIBCO EBX as a Web Component

This chapter contains the following topics:

36.1 Overview

EBX can be used as a user interface Web Component, called through the HTTP protocol. An EBX Web Component can be integrated into any application that is accessible through a supported web browser. This method of access offers the major benefits of EBX, such as user authentication, data validation, and automatic user interface generation, while additionally providing the ability to focus user navigation on specific elements of the repository.

Typical uses of EBX Web Components include integrating them into the intranet frameworks of organizations or into applications that manage the assignment of specific tasks to users.

See also*Supported web browsers* *[p 310]*

36.2 Integrating EBX Web Components into applications

A web application that calls an EBX Web Component can be:

1. A non-Java application, the most basic being a static HTML page.

   In this case, the application must send an HTTP request that follows the EBX Web Component request specifications [p 195].

2. A Java application, for example:

   - A Java web application running on the same application server instance as the EBX repository it targets or on a different application server instance.

- An EBX User service [p 189] or a Custom widget [p 190], in which case, the new session will automatically inherit from the parent EBX session.

> **Note**
>
> In Java, the recommended method for building HTTP requests that call EBX web components is to use the class `UIHttpManagerComponent`API in the API.

36.3 Repository element and scope selection

When an EBX Web Component is called, the user must first be authenticated in the newly instantiated HTTP session. The Web Component then selects a repository element and displays it according to the `scope` layout parameter defined in the request.

The parameter `firstCallDisplay` may change this automatic display according to its value.

The repository elements that can be selected are as follows:

- Dataspace or snapshot
- Dataset
- Node
- Table or a published view
- Table record

The scope determines how much of the user interface is displayed to the user, thus defining where the user is able to navigate in the session. The default scope that the Web component uses is the smallest possible depending on the entity or service being selected or invoked by the request.

See also*Scope* *[p 198]*

See also*firstCallDisplay* *[p 198]*

It is also possible to select a specific perspective as well as a perspective action.

By default, the selection of the element is done in the context of the perspective of the user if the scope is "full".

See also*Perspective* *[p 17]*

36.4 Combined selection

A URL of a Web component can specify a perspective and an action or an entity (dataspace, dataset, etc). Thus, for a Web component that has specified in its URL a perspective and an entity (but no action), if an action of the perspective matches this entity, then this action will be automatically selected.

Otherwise, if no action matches this entity, no action will be selected but the entity is opened regardless.

If an action is specified at the same time than an entity, this last is ignored and the action will be selected.

Specific case

If the target entity is a record and if an action is on the table that contains this record, then this action will be selected and the record will be opened inside the action.

In the same way, if a workflow work item is targeted by the web component, and if an action on « inbox » exists in the perspective, then this action will be selected and the work item will be opened inside it.

Known limitations

If the Web component specifies a predicate to filter a table, the perspective action must specify the exact same predicate to be selected.

In the same way, if the perspective action specifies a predicate to filter a table, the Web component must specify the exact same predicate to establish the match.

36.5 Request specifications

Base URL

In a default deployment, the base URL must be of the following form:

```
http://<host>[:<port>]/ebx/
```

> **Note**
>
> The base URL must refer to the servlet `FrontServlet`, defined in the deployment descriptor `/WEB-INF/web.xml` of the web application `ebx.war`.

User authentication and session information parameters

Parameter	Description	Required
`login` and `password`, or a *user directory-specific token*	Specifies user authentication properties. If neither a login and password pair nor a user directory-specific token is provided, user will be required to authenticate through the repository login page. See `Directory`API for more information.	No
`trackingInfo`	Specifies the tracking information of the new session. Tracking information is logged in history tables. Additionally, it can be used to programmatically restrict access permissions. See `AccessRule`API for more information.	No
`redirect`	The URL to which the user will be redirected at the end of the component session, when they click on the button 'Close'. The close button is always displayed for record selections, but whether or not it is displayed for all other cases must be specified using the parameter `closeButton`. For more information, see Exit policy [p 387].	No
`locale`	Specifies the locale to use. Value is either `en-US` or `fr-FR`.	No, default is the locale registered for the user.

Entity and service selection parameters

Parameter	Description	Required
`branch`	Selects the specified dataspace.	No
`version`	Selects the specified snapshot.	No
`instance`	Selects the specified dataset. The value must be the reference of a dataset that exists in the selected dataspace or snapshot.	Only if `xpath` or `viewPublication` is specified.
`viewPublication`	Specifies the publication name of the tabular or hierarchical view to apply to the selected content. This publication name is the one declared during the publication of the view. It can be found in the 'Administration' area under *Views configuration > Views*. All settings of the view, that is, its filters, sort order, and displayed columns, are applied to the result. A dataspace and a dataset must be selected in order for this view to be applied. The target table selection is not necessary, as it can be automatically determined based on the definition of the view. This parameter can be combined with the predicate specified in the `xpath` parameter as a logical 'AND' operation.	No
`xpath`	Specifies a node selection in the dataset. Value may be a valid absolute path located in the selected dataset. The notation must conform to a simplified XPath, with abbreviated syntax. It can also be a predicate surrounded by "[" and "]" if a table can be automatically selected using other Web Component parameters (for example, `viewPublication` or `workflowView`). For XPath syntax, see XPath supported syntax [p 227] See `UIHttpManagerComponent.setPredicate`API for more information.	No
`service`	Specifies the service to access. For more information on built-in User services, see Built-in services [p 201]. In the Java API, see `ServiceKey`API for more information.	No
`workflowView`	Specifies the workflow section to be selected. See `WorkflowView`API for more information.	No.
`perspectiveName`	Specifies the name of the perspective to be selected. If this parameter is specified, the `scope` parameter can have only two values: full and data.	Only if `perspectiveActionId` or `perspectiveActionName` is specified.
`perspectiveActionId`	**Deprecated.** Please consider using `perspectiveActionName` instead. Specifies the identifier of the perspective action to be selected.	No.

Parameter	Description	Required
`perspectiveActionName`	Specifies the unique name of the perspective action to be selected.	No.

Layout parameters

Parameter	Description	Required
`scope`	Specifies the scope to be used by the web component. Value can be `full`, `data`, `dataspace`, `dataset` or `node`. See `UIHttpManagerComponent.Scope`API for more information.	No, default will be computed to be the smallest possible according to the target selection.

`firstCallDisplay`	Specifies which display must be used instead of the one determined by the combination of selection and `scope` parameter.
	Possible values are:
	• `auto`: The display is automatically set according to the selection.
	• `view`: Forces the display of the tabular view or of the hierarchical view.
	• `record`: If the predicate has at least one record, forces the display of the first record in the list.
	For example,
	`firstCallDisplay=view`
	`firstCallDisplay=view:hierarchyExpanded`
	`firstCallDisplay=record`
	`firstCallDisplay=record:{predicate}`
	See `UIHttpManagerComponent.setFirstCallDisplay`API for more information.
	See `UIHttpManagerComponent.setFirstCallDisplayHierarchyExpanded`API for more information.
	See `UIHttpManagerComponent.setFirstCallDisplayRecord`API for more information.
		No, default will be computed according to the target selection.

`closeButton`	Specifies how to display the session close button. Value can be `logout` or `cross`. See `UIHttpManagerComponent.CloseButtonSpec`API for more information.	No. If scope is not `full`, no close button will be displayed by default.

`dataSetFeatures`	Specifies which features to display in a UI service at the dataset level or a form outside of a table.
	These options pertain only to features in the workspace. It is recommended to use this property with the smallest `scope` possible, namely `dataset` or `node`.
	Syntax:
	`<prefix> ":" <feature> [ "," <feature>]*`
	where
	• `<prefix>` is `hide` or `show`,
	• `<feature>` is `services`, `title`, `save`, or `revert`.
	For example,
	`hide:title`
	`show:save,revert`
	See `UIHttpManagerComponent.DataSetFeatures`API for more information.
		No.

`viewFeatures`	Specifies which features to display in a tabular or a hierarchy view (at the table level).	No.

Parameter	Description	Required

	These options pertain only to features in the workspace. It is recommended to use this property with the smallest `scope` possible, namely `dataset` or `node`.
	Syntax:
	`<prefix> ":" <feature> [ "," <feature>]*`
	where
	• `<prefix>` is `hide` or `show`,
	• `<feature>` is `create`, `views`, `selection`, `filters`, `services`, `refresh`, `title`, or `breadcrumb`.
	For example,
	`hide:title,selection`
	`show:service,title,breadcrumb`
	See `UIHttpManagerComponent.ViewFeatures`API for more information.	

`recordFeatures`	Specifies which features must be displayed in a form at the record level.
	These options pertain only to features in the workspace. It is recommended to use this property with the smallest `scope` possible, namely `dataset` or `node`.
	Syntax:
	`<prefix> ":" <feature> [ "," <feature>]*`
	where
	• `<prefix>` is `hide` or `show`,
	• `<feature>` is `services`, `title`, `breadcrumb`, `save`, `saveAndClose`, `close`, or `revert`.
	For example,
	`hide:title`
	`show:save,saveAndClose,revert`
	See `UIHttpManagerComponent.RecordFeatures`API for more information.
		No.

`pageSize`	Specifies the number of records that will be displayed per page in a table view (either tabular or hierarchical).	No.
`startWorkItem`	Specifies a work item must be automatically taken and started. Value can be `true` or `false`. See `ServiceKey.WORKFLOW`API for more information.	No. Default value is `false`, where the target work item state remains unchanged.

36.6 Example calls to an EBX Web Component

Minimal URI:

`http://localhost:8080/ebx/`

Logs in as the user 'admin' and selects the 'Reference' dataspace:

`http://localhost:8080/ebx/?login=admin&password=admin&branch=Reference`

Selects the 'Reference' dataspace and accesses the built-in validation service:

```
http://localhost:8080/ebx/?
login=admin&password=admin&branch=Reference&service=@validation
```

Selects the roles table in the default directory:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/roles
```

Selects the record 'admin' in the default directory:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/user[./login="admin"]
```

Accesses the interface for creating a new user in the default directory:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/user&service=@creation
```

Compares the record 'admin' in the default directory with the record 'jSmith':

Compares the record 'R1' in the dataset 'instanceId' in the dataspace 'Reference' with the record 'R0':

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/user[./
login="admin"]&service=@compare&compare.branch=ebx-directory&compare.instance=ebx-
directory&compare.xpath=/directory/user[./login="jSmith"]
```

CHAPTER 37

Built-in user services

EBX includes a number of built-in user services. Built-in user services can be used:

- when defining workflow model tasks [p 138]
- when defining perspective action menu items [p 18]
- as extended user services when used with service extensions [p 587]
- when using EBX as a Web Component [p 193]

This reference page describes the built-in user services and their parameters.

This chapter contains the following topics:

37.1 Access data (default service)

By default, workflows automatically consider this service as complete. That is, the 'Accept' button is always available.

This is the default service used if no service is specified.

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`disableAutoComplete`	Disable Accept at start	By default, the interaction associated with this service is directly considered as complete. Therefore, the Accept button is automatically displayed at the opening of the work item. This parameter is useful to disable this behavior. If the value is 'true', the developer will be in charge of completing the interaction by using SessionInteraction in a user service or a trigger, for example. The default value is 'false'. Perspectives do not use this parameter.
`firstCallDisplay`	First call display mode	Defines the display mode that must be used when displaying a filtered table or a record upon first call. Default (value = 'auto'): the display is automatically set according to the selection. View (value = 'view'): forces the display of the tabular view or of the hierarchical view. Record (value = 'record'): if the predicate has at least one record, forces the display of the record form.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.
`viewPublication`	View	The publication name of the view to display. The view must be configured for the selected table.
`xpath`	Dataset node (XPath)	The value must be a valid absolute location path in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax.

37.2 Create a new record

For a workflow, the creation service is considered complete when the first successful submit is performed (record has been created). If this service is called whereas it is already complete, the created record is displayed in update or read-only mode (depending on the user rights).

Service name parameter: `service=@creation`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - This field is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Dataset table (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Output parameters

Parameter	Label	Description
`created`	Created record	Contains the XPath of the created record.

37.3 Duplicate a record

For a workflow, the duplicate service is considered complete when the first successful submit is performed (record has been created). If this service is called whereas it is already complete, the created record is displayed in update or read-only mode (depending on the user rights).

Service name parameter: `service=@duplicate`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - This field is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Record to duplicate (XPath)	The value must be a valid absolute location path of an existing record. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Output parameters

Parameter	Label	Description
`created`	Created record	Contains the XPath of the created record.

37.4 **Export data to an XML file**

The exportToXML service is considered complete when export is done and file downloaded.

Service name parameter: `service=@exportToXML`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.
`xpath`	Dataset table to export (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

37.5 **Export data to a CSV file**

Workflows consider the exportToCSV service as complete when export is done and file downloaded.

Service name parameter: `service=@exportToCSV`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.
`xpath`	Dataset table to export (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

37.6 Import data from an XML file

Workflows consider the importFromXML service as complete when import is performed.

Service name parameter: `service=@importFromXML`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Dataset table to import (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

37.7 Import data from a CSV file

Workflows consider the importFromCSV service as complete when import is performed.

Service name parameter: `service=@importFromCSV`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Dataset table to import (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

37.8 Access a dataspace

A workflow automatically considers that the dataspace selection service is complete.

Service name parameter: `service=@selectDataSpace`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.

37.9 Validate a dataspace, a snapshot or a dataset

Workflows automatically consider the validation service as complete.

Service name parameter: `service=@validation`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace or snapshot is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace or snapshot is required for this service.

Output parameters

Parameter	Label	Description
`hasError`	Found errors	Contains 'true' if validation has produced errors.
`hasFatal`	Found fatal errors	Contains 'true' if validation has produced fatal errors.
`hasInfo`	Found informations	Contains 'true' if validation has produced informations.
`hasWarning`	Found warnings	Contains 'true' if validation has produced warnings.

37.10 **Merge a dataspace**

Workflows consider the merge service as complete when merger is performed and dataspace is closed.

Service name parameter: `service=@merge`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.

Output parameters

Parameter	Label	Description
`mergeResult`	Merge success	Contains 'true' if merge succeeded, otherwise 'false'.
`mergeState`	Merge state	Contains the return code of the merge. It is strongly recommended to parse this value by using the InteractionMergeState UIHttpManagerComponentReturnCode.

37.11 Access the dataspace merge view

The merge.view service is automatically considered complete.

Service name parameter: `service=@merge.view`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.

37.12 Compare contents

Workflows automatically consider the compare service as complete.

Service name parameter: `service=@compare`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`compare.branch`	Dataspace to compare	The identifier of the dataspace to compare - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`compare.filter`	Comparison filter	To ignore inheritance and computed values fields in the comparison (disable resolved mode), the filter "persistedValuesOnly" must be specified. By default, when no filter is defined, the comparison uses resolved mode.
`compare.instance`	Dataset to compare	The value must be the reference of a dataset that exists in the selected dataspace to compare.
`compare.version`	Snapshot to compare	The identifier of the snapshot to compare - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`compare.xpath`	Table or record to compare (XPath)	The value must be a valid absolute location path of a table or a record in the selected dataset to compare. The notation must conform to a simplified XPath, in its abbreviated syntax.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`xpath`	Table or record (XPath)	The value must be a valid absolute location path of a table or a record in the selected dataset. The notation must

Parameter	Label	Description
		conform to a simplified XPath, in its abbreviated syntax.

37.13 **Data workflows**

This service provides access to the data workflows user interfaces.

Service name parameter: `service=@workflow`

> **Note**
>
> This service is for perspectives only.

Input parameters

Parameter	Label	Description
`scope`	Scope	Defines the scope of the user navigation for this service.
`viewPublication`	View publication	Defines the publication name of the view to apply for this service.
`workflowView`	Workflow view	Specifies the workflow view type. Value can be one of the following: "inbox", "launcher", "monitoringPublications", "monitoringWorkflows", "monitoringWorkItems" or "completedWorkflows".
`xpath`	Filter (XPath)	An optional filter. The syntax should conform to an XPath predicate surrounded by "[" and "]".

CHAPTER 38

XML import and export

This chapter contains the following topics:

38.1 Introduction

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

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

Imports and exports can also be done programmatically.

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

38.2 Imports

Attention

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

Import mode

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

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

Insert and update operations

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

See the data services operations update [p 623] and insert [p 625], as well as `ImportSpec.setByDelta`API in the Java API for more information.

State in source XML document	Behavior
Element does not exist in the source document	**If 'by delta' mode is disabled (default):** Target field value is set to one of the following: • If the element defines a default value, the target field value is set to that default value. • If the element is of a type other than a string or list, the target field value is set to `null`. • If the element is an aggregated list, the target field value is set to an empty list. • If the element is a string that distinguishes `null` from an empty string, the target field value is set to `null`. If it is a string that does not distinguish between the two, an empty string. • If the element (simple or complex) is hidden in data services, the target value is not changed. **See also***Hiding a field in Data Services* *[p 540]* **Note:** The user performing the import must have the permissions necessary to create or change the target field value. Otherwise, the value will remain unchanged. **If 'by delta' mode has been enabled through data services or the Java API:** • For the `update` operation, the field value remains unchanged. • For the `insert` operation, the behavior is the same as when `byDelta` mode is disabled.
Element exists but is empty (for example, `<fieldA/>`)	• For nodes of type `xs:string` (or one of its sub-types), the target field's value is set to `null` if it distinguishes `null` from an empty string. Otherwise, the value is set to empty string. • For non-`xs:string` type nodes, an exception is thrown in conformance with XML Schema. **See also***TIBCO EBX whitespace management for data types* *[p 525]*
Element is present and `null` (for example, `<fieldA xsi:nil="true"/>`)	The target field is always set to `null` except for lists, for which it is not supported. In order to use the `xsi:nil="true"` attribute, you must import the namespace declaration `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`.

Set missing values as null

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

Ignore extra columns

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

Optimistic locking

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

38.3 **Exports**

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

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

The XML export options are as follows:

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

38.4 Handling of field values

Date, time & dateTime format

The following date and time formats are supported:

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

38.5 **Known limitations**

Association fields

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

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

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

Selection nodes

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

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

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

CHAPTER **39**

CSV import and export

This chapter contains the following topics:

39.1 **Introduction**

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

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

Imports and exports can also be done programmatically.

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

See also*Default option values* *[p 391]*

39.2 **Exports**

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

The CSV export options are as follows:

Download file name	Specifies the name of the CSV file to be exported. This field is pre-populated with the name of the table from which the records are being exported.
File encoding	Specifies the character encoding to use for the exported file. The default is UTF-8.
Enable inheritance	In order to consider the inheritance [p 27] during a CSV export, the option has to be defined in the model. For more information on inheritance, see Inheritance and value resolution [p 270]. Specifies if inheritance will be taken into account during a CSV export. If inheritance is enabled, resolved values of fields are exported with the technical data that define the possible inheritance mode of the record or the field. If inheritance is disabled, resolved values of fields are exported and occulted records are ignored. By default, this option is disabled. **Note:** Inheritance is always ignored, if the table dataset has no parent or if the table has no inherited field.
User-friendly mode	Specifies whether exported values will be represented in a user-friendly way, or in a raw format. For example, in user-friendly mode, dates and numbers are formatted according to the user's locale, and foreign keys and enumerated values display their associated labels. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Include technical data	Specifies whether internal technical data will be included in the export. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Column header	Specifies whether or not to include column headers in the CSV file. • **No header** • **Label:** For each column in the spreadsheet, the CSV displays its label. Each label is localized according to the locale preference of the current session. If no user-

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

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

39.3 Imports

Download file name	Specifies the name of the CSV file to be imported.
Import mode	When importing a CSV file, you must specify one of the following import modes, which will control the integrity of operations between the source and the target table. • **Insert mode:** Only record creation is allowed. If a record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled. • **Update mode:** Only modifications of existing records are allowed. If no record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled. • **Update or insert mode:** If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created. • **Replace (synchronization) mode:** If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created. If a record exists in the target table but is not present in the source XML file, that record is deleted from the table.
File encoding	Specifies the character encoding to use for the exported file. The default is UTF-8.
Enable inheritance	In order to consider the inheritance [p 27] during a CSV import, the option has to be defined in the model. For more information on inheritance, see Inheritance and value resolution [p 270] and `ExportImportCSVSpec.setInheritanceEnabled`API. Specifies whether the inheritance will be taken into account during a CSV import. If technical data in the CSV file define an inherit mode, corresponding fields or records are forced to be inherited. If technical data define an occult mode, corresponding records are forced to be occulted. Otherwise, fields are overwritten with values read from the CSV file. By default, this option is disabled. **Note:** Inheritance is always ignored if the dataset of the table has no parent or if the table has no inherited field.

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

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

39.4 Handling of field values

Aggregated lists

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

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

Hidden fields

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

'Null' value for strings

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

Using programmatic services, the specific value `ebx-csv:nil` can be assigned to strings with values set to `null`. If this is done, the `null` string values will not be replaced by empty strings during round trip export-import procedures. See `ExportImportCSVSpec.setNullStringEncoded`API in the Java API for more information.

Date, time & dateTime format

The following date and time formats are supported:

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

39.5 Known limitations

Aggregated lists of groups

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

Terminal groups

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

As a consequence, some differences may appear during comparison after performing an export followed by an import. To ensure the symmetry of import and export, use XML import and export instead. See XML import and export [p 215].

Column label headers

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

Association fields

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

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

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

Selection nodes

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

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

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

CHAPTER 40

Supported XPath syntax

This chapter contains the following topics:

40.1 Overview

The XPath notation used in TIBCO EBX must conform to the *abbreviated syntax* of the XML Path Language (XPath) Version 1.0 standard, with certain restrictions. This document details the abbreviated syntax that is supported.

40.2 Example expressions

The general XPath expression is:

```
path[predicate]
```

Absolute path

```
/library/books/
```

Relative paths

```
./Author
../Title
```

Root and descendant paths

```
//books
```

Table paths with predicates

```
../../books/[author_id = 0101 and (publisher = 'harmattan')]
/library/books/[not(publisher = 'dumesnil')]
```

Complex predicates

```
starts-with(col3,'xxx') and ends-with(col3,'yyy') and osd:is-not-null(./col3))
```

`contains(col3 ,'xxx') and ( not(col1=100) and date-greater-than(col2,'2007-12-30') )`

Predicates with parameters

`author_id = $param1 and publisher = $param2` where the parameters `$param1` and `$param2` refer respectively to `0101` and `'harmattan'`

`col1 < $param1 and col4 = $param2` where the parameters `$param1` and `$param2` refer respectively to `100` and `'true'`

`contains(col3,$param1) and date-greater-than(col2,$param2)` where the parameters `$param1` and `$param2` refer respectively to `'xxx'` and `'2007-12-30'`

> **Note**
>
> The use of this notation is restricted to the Java API since the parameter values can only be set by the method `Request.setXPathParameter`API of the Java API.

Predicates on label

```
osd:label(./delivery_date)='12/30/2014' and ends-with(osd:label(../adress),'Beijing -
 China')
```

Predicates on record label

```
osd:contains-record-label('dumesnil') or osd:contains-record-label('harmattan')
```

Predicates for validation search

```
osd:has-validation-item()
osd:has-validation-item('error,info')
osd:contains-validation-message('xxx')
osd:contains-validation-message('xxx','info,warning')
```

> **Note**

- XPath functions for validation search cannot be used on XPath predicates defined on associations and foreign key filters.
- The predicates `osd:label`, `osd:contains-record-label` and `osd:contains-validation-message` are localized. The locale can be set by the methods of the Java API `Request.setLocale`API or `Request.setSession`API.

Attention

To ensure that the search is performed on an up-to-date validation report, it is necessary to perform an explicit validation of the table just before using these predicates.

40.3 **Syntax specifications for XPath expressions**

Overview

Expression	**Format**	**Example**
XPath expression	*<container path>*[*predicate*]	`/books[title='xxx']`
<container path>	*<absolute path>* or *<relative path>*	
<absolute path>	`/a/b` or `//b`	`//books`
<relative path>	`../../b`, `./b` or `b`	`../../books`

Predicate specification

Expression	**Format**	**Notes/Example**
<predicate>	Example: A and (B or not(C)) A,B,C: *<atomic expression>*	Composition of: logical operators parentheses, not() and atomic expressions.
<atomic expression>	*<path><comparator><criterion>* or method(*<path>*,*<criterion>*)	`royalty = 24.5` `starts-with(title, 'Johnat')` `booleanValue = true`
<path>	*<relative path> or osd:label(<relative path>)*	Relative to the table that contains it: `../authorstitle`
<comparator>	*<boolean comparator>*, *<numeric comparator>* or *<string comparator>*	
<boolean comparator>	= or `!=`	
<numeric comparator>	= ,`!=` ,<, >, <=, or `>=`	
<string comparator>	`=`	
<method>	*<date method>*, *<string method>*, `osd:is-null` method or `osd:is-not-null` method	
<date, time & dateTime method>	`date-less-than`, `date-equal` or `date-greater-than`	
<string method>	`matches`, `starts-with`, `ends-with`, `contains`, `osd:is-empty`, `osd:is-not-empty`, `osd:is-empty-or-nil`, `osd:is-neither-empty-nor-nil`, `osd:is-equal-case-insensitive`, `osd:starts-with-case-insensitive`, `osd:ends-with-case-insensitive`, `osd:contains-case-insensitive`, or `osd:contains-record-label`	
<criterion>	*<boolean criterion>*, *<numeric criterion>*, *<string criterion>*, *<date criterion>*, *<time criterion>*, or *<dateTime criterion>*	
<boolean criterion>	`true` ,`false`	
<numeric criterion>	An integer or a decimal	`-4.6`
<string criterion>	Quoted character string	`'azerty'`

Expression	Format	Notes/Example
<date criterion>	Quoted and formatted as 'yyyy-MM-dd'	`'2007-12-31'`
<time criterion>	Quoted and formatted as 'HH:mm:ss' or 'HH:mm:ss.SSS'	`'11:55:00'`
<dateTime criterion>	Quoted and formatted as 'yyyy-MM-ddTHH:mm:ss' or 'yyyy-MM-ddTHH:mm:ss.SSS'	`'2007-12-31T11:55:00'`

XPath 1.0 formula

It is possible to use an XPath 1.0 formula in the criterion value part of an atomic predicate expression (right-hand side).

For example, instead of `[./a=3]`, you may use the expression `[./a=(floor(./d)+ 2.0)]`.

Due to the strong dependence of predicates on the data model node and the node type of the criterion, the path portion of the atomic predicate expression (left-hand side) must be a node path and cannot be an XPath formula. For example, the expression `/table[floor(./a) > ceiling(./d)]` is not valid.

Predicate on label

The `osd:label()` function can be applied to the path portion of the atomic predicate, in order to resolve the predicate on the label instead of the value. In this case, only string operators and string criteria can be used, i.e. `ends-with(osd:label(./price),'99')`.

A predicate on label is localized, so the criterion must be expressed in the same locale as the predicate-filtered request. For example: `request.setLocale(Locale.FRENCH); request.setXPathFilter("osd:label(./delivery_date)='30/12/2014'");`

Note

It is forbidden to use the `osd:label` function if the right part of the predicate is a contextual value.

Note

If the `osd:label` function is used in a data model, for example on a selection or in the filter predicate of a table reference node, the default locale of the data model (as defined in its module declaration) must be used for the criterion format (even though this is generally not recommended).

See also *`SchemaNode.displayOccurrence`* API

Contextual values

For predicates that are relative to a selected node, the criterion value (that is, the right-hand side of the predicate) can be replaced with a contextual path using the syntax `${<relative-path>}` where *<relative-path>* is the location of the element relative to the selected node.

> **Note**
>
> When calling a method, the criterion is the second parameter, and the first parameter cannot be a relative value.

Aggregated lists

For predicates on aggregated lists, the predicate returns `true` regardless of the comparator if one of the list elements verifies the predicate.

> **Note**
>
> Special attention must be paid to the comparator `!=`. For example, for an aggregated list, `./list != 'a'` is not the same as `not(./list = 'a')`. Where the list contains the elements `(e1,e2,..)`, the first predicate is equivalent to `e1 != 'a' or e2 != 'a' ...`, while the second is equivalent to `e1 != 'a' and e2 != 'a' ...`.

'Null' values

Null values must be explicitly treated in a predicate using the operators `osd:is-null` and `osd:is-not-null`.

For example, `/root/products[./price<100]` or `/root/products[./price!=100]` will not return any products whose prices are not set (`null`). For the latter case to return unset values as well, the predicate must instead be: `/root/products[./price!=100 or osd:is-null(./price)]`.

How to manage single and double quotes in literal expressions

By default, a literal expression is delimited by single quotes (`'`). If the literal expression contains single quotes and no double quotes, the expression must be delimited by double quotes (`"`). If the literal expression contains both single and double quotes, the single quotes must be doubled.

The method `XPathExpressionHelper.encodeLiteralStringWithDelimiters`API in the Java API handles this.

Examples of using `encodeLiteralStringWithDelimiters`

Value of Literal Expression	**Result of this method**
`Coeur`	`'Coeur'`
`Coeur d'Alene`	`"Coeur d'Alene"`
`He said: "They live in Coeur d'Alene".`	`'He said: "They live in Coeur d''Alene".'`

Extraction of foreign keys

In EBX, the foreign keys are grouped into a single field with the `osd:tableRef` [p 498] declaration.

The standard XPath syntax has been extended so as to extract the value of any targeted primary key field.

Example

If the table `/root/tableA` has an `osd:tableRef` field named 'fkB' whose target is `/root/tableB` and the primary key of `tableB` has two fields, `id` of type `xs:int` and `date` of type `xs:date`, then the following expressions would be valid:

- `/root/tableA[ fkB = '123|2008-01-21' ]`, where the string "123|2008-01-21" is a representation of the entire primary key value.

  See **Syntax of the internal String representation of primary keys** `PrimaryKey.syntax`API for more information.
- `/root/tableA[ fkB/id = 123 and date-equal(fkB/date, '2008-01-21') ]`, where this predicate is a more efficient equivalent to the one in the previous example.
- `/root/tableA[  fkB/id >= 123 ]`, where any number operator could be used, as the targeted primary key field is of type `xs:int`.
- `/root/tableA[ date-greater-than( ./fkB/date,'2007-01-01' ) ]`, where any date operator could be used, as the targeted primary key field is of type `xs:date`;
- `/root/tableA[ fkB = "" ]` is not valid as the targeted primary key has two columns.
- `/root/tableA[ osd:is-null(fkB) ]` checks if a foreign key is `null` (not defined).

40.4 Java API

Using the XPath in the Java API:

In the Java API, the `XPathFilter` class allows to define XPath predicates and to execute requests on them.

The `XPathExpressionHelper` class provides utilitarian methods to handle XPath predicates and paths.

Localization

CHAPTER **41**

Labeling and localization

This chapter contains the following topics:

41.1 Overview

TIBCO EBX offers the ability to handle the labeling and the internationalization of data models.

Localizing user interactions

In EBX, language preferences can be set for two scopes:

1. Session: Each user can select a default locale from the user pane.
2. The EBX main configuration file, named `ebx.properties` by default. See Extending TIBCO EBX internationalization [p 239] for more information.

Textual information

In EBX, most master data entities can have a label and a description, or can correspond to a user message. For example:

- Dataspaces, snapshots and datasets can have their own label and description. The label is independent of the unique name, so that it remains localizable and modifiable;
- Any node in the data model can have a static label and description;
- Values can have a static label when they are enumerated;
- Validation messages can be customized, and permission restrictions can provide text explaining the reason;
- Each record is dynamically displayed according to its content, as well as the context in which it is being displayed (in a hierarchy, as a foreign key, etc.);

All this textual information can be localized into the locales that are declared in `ebx.properties`.

See also

Labels and messages [p 531]

Tables declaration [p 493]

Foreign keys declaration [p 498]

41.2 Value formatting policies

When a value is displayed to the user, it is formatted according to its type and the formatting policy of the current locale. For example, a date will be displayed in some locales as "dd/MM/yyyy" and "MM/dd/yyyy" in others.

A formatting policy is used to define how to display the values of simple types [p 480].

For each locale declared in `ebx.properties`, its formatting policy is configured in a file located at `/WEB-INF/ebx/{locale}/frontEndFormattingPolicy.xml`. For instance, to define the formatting policy for Greek (`el`), the engine looks for the following path in the module:

```
/WEB-INF/ebx/el/frontEndFormattingPolicy.xml
```

If the corresponding file does not exist, the formatting policy is looked up in the class-path of EBX. If the locale-specific formatting policy is not found, the formatting policy of `en_US` is applied.

The content of the file `frontEndFormattingPolicy.xml` is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<formattingPolicy  xmlns="urn:ebx-schemas:formattingPolicy_1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ebx-schemas:formattingPolicy_1.0 ../schema/ebx-reserved/formattingPolicy_1.0.xsd">
 <date pattern="dd/MM" />
 <time pattern="HH:mm:ss" />
 <dateTime pattern="dd/MM/yyyy HH:mm" />
 <decimal pattern="00,00,00.000" groupingSeparator="|" decimalSeparator="^"/>
 <int pattern="000,000" groupingSeparator=" "/>
</formattingPolicy>
```

The elements `date`, `dateTime` and `time` are mandatory.

The group and decimal separators that appear in the formatted numbers can be modified by defining the attributes `groupingSeparator` and `decimalSeparator` for the elements `decimal` and `int`.

41.3 Syntax for locales

There are two ways to express a locale:

1. The XML recommendation follows the IETF BCP 47 recommendation, which uses a hyphen '-' as the separator.
2. The Java specification uses an underscore '_' instead of a hyphen.

In any XML file (XSD, formatting policy file, etc.) read by EBX, either syntax is allowed.

For a web path, that is, a path within the web application, only the Java syntax is allowed. Thus, formatting policy files must be located in directories whose locale names respect the Java syntax.

See also*Extending TIBCO EBX internationalization* [p 239]

CHAPTER 42

Extending TIBCO EBX internationalization

This chapter contains the following topics:

42.1 Overview of the native EBX localization

By default, the EBX built-in user interface is provided in English (en-US) and French (fr-FR).

Localization consists of a formatting policy and a set of message files (resource bundle):

- For English, localization is provided by a formatting policy and a set of message files with no locale defined,
- For French, localization is provided by a formatting policy and a set of message files with locale set to "fr".

EBX provides an option to add locales in order to extend the localization of the user interface and to internationalize the documentation of data models and associated services.

42.2 Extending EBX user interface localization

EBX supports the localization of its user interface into any compatible language and region.

> **Note**
>
> Currently, Latin & Cyrillic characters are supported. Locales that use other character sets may be usable, but are not supported.

Adding a new locale

In order to add a new locale, the following steps must be followed:

- Declare the new locale in the EBX main configuration file. For example:

  ebx.locales.available=en-US, fr-FR, xx

- The first locale is always considered the default.
- The built-in locales, en-US and fr-FR, can be removed if required.

See Configuring EBX localization [p 349].

- Deploy the following files in the EBX class-path:
  - A formatting policy file, named `com.orchestranetworks.i18n.frontEndFormattingPolicy_xx.xml`,
  - A set of localized message files (*_xx.mxml) in a resource bundle.

> **Note**
>
> The files must be ending with ".mxml".

42.3 Localized resources resolution

Since version 5.7.0, localized resources are resolved on a locale-proximity base, with the following lookup mechanism:

- resourceName + "_" + language + "_" + country + "_" + variant + ".mxml"
- resourceName + "_" + language + "_" + country + ".mxml"
- resourceName + "_" + language + ".mxml"
- resourceName + ".mxml"

> **Note**
>
> The resolution is done at the localized message level. It is therefore possible to define one or more files for a locale that only includes messages for which specific localization is required.

42.4 Known limitations

Non extendable materials

Localization of the following cannot be extended:

- EBX product documentation,
- EBX HTML editor and viewer.

Persistence

CHAPTER 43

Overview of persistence

This chapter describes how master data, history, and replicated tables are persisted. A given table can employ any combination of master data persistence mode, historization, and replication.

While all persisted information in TIBCO EBX is ultimately stored as relational tables in the underlying database, whether it is in a form that is accessible outside of EBX depends on if it is in mapped mode.

> **Note**
>
> The term *mapped mode* [p 243] refers to any tables that are stored as-is, and thus whose contents can be accessed directly in the database.

This chapter contains the following topics:

43.1 Persistence of managed master data

Data that is modeled in and governed by the EBX repository can be persisted in one of two modes, semantic (default) or relational, as specified in its underlying data model. Distinct tables defined in either mode can co-exist and collaborate within the same EBX repository.

For a comparison between relational mode and semantic mode, see the chapter Overview of modes [p 245]

43.2 Historization

Master data tables can activate historization in order to track modifications to their data, regardless of whether they are persisted in semantic or relational mode, and whether they are replicated.

The history itself is in mapped mode, meaning that it can potentially be consulted directly in the underlying database.

See also*History* *[p 251]*

43.3 **Replication**

Replication enables direct SQL access to tables of master data, by making a copy of data in the repository to relational table replicas in the database. Replication can be enabled on any table regardless of whether it is persisted in semantic or relational mode, and whether it has history activated.

The replica tables are persisted in mapped mode, as their primary purpose is to make master data accessible to direct queries outside of EBX.

See also_Replication_ *[p 259]*

43.4 **Mapped mode**

Overview of mapped mode

Mapped mode refers to cases where tables are persisted in the underlying relational database in a format that allows their data to be accessed directly, outside of EBX. Master data modeled in relational mode, history, and replica tables are all examples of tables in mapped mode.

All cases of mapped mode involve automatic alterations of the database schema (the database tables, indexes, etc.) when necessary, by automatically executing required DDL statements in the background. Such procedures are always triggered at data model compilation time and the data model compilation report notifies of any resulting errors.

Another general consideration regarding mapped modes is that, in most cases, when a data model entity is removed, its corresponding database object is not deleted immediately. Instead, it is marked as disabled, which leaves the possibility of later re-enabling the object. In order to definitively drop the object and its associated data and resources from the database, it must be marked for purge. The removal then takes place during the next global purge.

See also

Database mapping administration *[p 405]*

Data model evolutions *[p 265]*

Data model restrictions due to mapped mode

Due to the nature of persisting directly in the underlying database, some restrictions apply to all tables stored in mapped mode:

- Limitations of supported databases [p 313]
- Unlimited-length strings: All string fields, except foreign keys, of type `xs:string`, its derived types, and `xs:anyURI` must define a 'maxLength' or 'length' facet. Since a foreign key field is composed of the final primary key field(s) of its target table(s), this facet requirement applies to each of those final primary key fields instead of the foreign key field itself. Additionally, limitations of the underlying database concerning the maximum length of its character types apply, such as *VARCHAR* and *NVARCHAR2*.
- Large lists of columns might not be indexable. Example for Oracle: the database enforces a limit on the maximum cumulated size of the columns included in an index. For strings, this size also depends on the character set. If the database server fails to create the index, you should consider redesigning your indexes, typically by using a shorter length for the concerned columns, or by

including fewer columns in the index. The reasoning is that an index leading to this situation would have headers so large that it could not be efficient anyway.

- Fields of type `type="osd:password"` are ignored.
- Terminal complex types are supported; however, they cannot be globally set to `null` at record-level.

More generally, tables in mapped mode are subject to any limitations of the underlying RDBMS. For example, the maximum number of columns in a table applies (1000 for Oracle 11g R2, 1600 for PostgreSQL). Note that a history table contains twice as many fields as declared in the schema (one functional field, plus one generated field for the operation code).

Data model evolutions may also be constrained by the underlying RDBMS, depending on the existing data model.

See also*Data model evolutions* *[p 265]*

CHAPTER 44 Relational mode

This chapter contains the following topics:

44.1 Overview of modes

Semantic mode explained

Semantic mode offers all TIBCO EBX advanced features of master data management, in particular, dataspaces, dataset inheritance, and inherited fields.

Semantic mode is the default mode for persisting the data governed by the EBX repository. Data models are in semantic mode unless relational mode [p 246] is explicitly specified [p 246].

Internally, the master data managed in semantic mode is represented as standard XML, which complies with the XML Schema Document of its data model. The XML representation is additionally compressed and segmented for storage into generic relational database tables. This mode provides efficient data storage and access, including for:

- Dataspaces: no data is duplicated when creating a child dataspace, and
- Inheritance: no data is duplicated when creating an inherited instance.

Semantic mode also makes it possible to maintain an unlimited number of datasets for each data model, organized into an unlimited number of dataspaces or snapshots. This can be done with no impact on the database schema.

As this mode only uses common, generic internal tables, modifications to the structure of the data model also never impact the database schema. Data model evolutions only impact the content of the generic database tables.

See also

dataspaces [p 90]

dataset inheritance [p 271]

inherited fields [p 272]

Relational mode explained

Relational mode, which is a mapped mode, persists master data directly into the database. The primary function of relational mode is to be able to benefit from the performance and scalability capabilities of the underlying relational database. However, relational mode does not support the advanced governance features offered by semantic mode.

For some cases where the management advantages of semantic mode are not necessary, such as "current time" tables, or tables that are regularly updated by external systems, the performance gains offered by relational mode may be more valuable.

Generally, when a dataset is in relational mode, every table in this dataset has a corresponding table in the database and every field of its data model is mapped to a relational table column.

Direct comparison of semantic and relational modes

This table summarizes the differences between the two persistence modes:

	Semantic mode	**Relational mode**
Dataspaces	Yes	No
Dataset inheritance	Yes	No
Inherited fields	Yes	No
Data model	All features are supported.	Some restrictions, see Data model restrictions for tables in relational mode [p 249].
Direct SQL reads	No	Yes, see SQL reads [p 249].
Direct SQL writes	No	Yes, but only under precise conditions, see SQL writes [p 249].
Data validation	Yes, enables tolerant mode.	Yes, some constraints become blocking, see Validation [p 247].
Transactions	See Concurrency and isolation levels [p 466].	See Concurrency and isolation levels [p 466].
Data model evolutions	See Data model evolutions [p 265] in the Reference Manual.	See Data model evolutions [p 265] in the Reference Manual.

44.2 Enabling relational mode for a data model

The data model declares that it is in relational mode. Due to the necessary restrictions of relational mode, such as not having child dataspaces or snapshots, a specific relational dataspace must be provided, to which the data model will be published. Relational dataspaces do not allow creating sub-dataspaces or snapshots.

Example of a relational mode declaration:

```
<xs:schema>
 <xs:annotation>
  <xs:appinfo>
   <osd:relationalMode>
    <dataSpace>aDataSpaceKey</dataSpace>
    <dataSet>aDataSetReference</dataSet>
    <tablesPrefix>aPrefixForTablesInRDBMS</tablesPrefix>
   </osd:relationalMode>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:schema>
```

with the elements:

Element	**Description**	**Required**
`dataSpace`	Specifies the dataspace where the data model must be published. This dataspace must itself be in relational mode. No dataspace or snapshot can be created from a dataspace declared in such a mode.	Yes
`dataSet`	Specifies the dataset where the data model must be published.	Yes
`tablesPrefix`	Specifies the common prefix used for naming the generated tables in the database.	Yes

44.3 Validation

This section details the impact of relational mode on data validation.

Structural constraints

Some EBX data model constraints will generate a "structural constraint" on the underlying RDBMS schema for relational mode and also if table history is activated [p 251]. This concerns the following facets:

- facets `xs:maxLength` and `xs:length` on `string` elements;
- facets `xs:totalDigits` and `xs:fractionDigits` on `xs:decimal` elements.

Databases do not support as tolerant a validation mode as EBX. Hence, the above constraints become *blocking constraints*. A blocking constraint means that updates are rejected if they do not comply. Additionally, such constraints are no longer checked during validation process, except for foreign key constraints under some circumstances (see Foreign key blocking mode [p 247]). When a transaction does not comply with a blocking constraint, it is cancelled and a `ConstraintViolationException`API is thrown.

See also*Blocking and non-blocking constraints [p 522]*

Foreign key blocking mode

In order to reduce validation time, foreign key constraints are automatically set in blocking mode if:

- The foreign key constraints are defined on a table in relational mode,
- The foreign key constraints are defined on a table in semantic or relational mode referencing a table in relational mode.

For these constraints, blocking mode implies that attempting the following actions will result in a `ConstraintViolationException`API:

- Deleting a record referenced by a foreign key constraint,
- Deleting an instance referenced by a foreign key constraint,
- Closing a dataspace referenced by a foreign key constraint.

However, it is possible to overwrite this behavior by setting a specific control policy. See Blocking and non-blocking constraints [p 522] for more information.

In order to ensure the integrity of foreign key constraints after direct SQL writes that bypass the EBX governance framework, the foreign key constraints will be validated on the following cases:

- On the first explicit validation through the user interface or API,
- On the first explicit validation through the user interface or API after refreshing the schema,
- On the first explicit validation through the user interface or API after resetting the validation report of a dataset in the user interface.

Important:

- Blocking aspect of the foreign key constraint does not concern filters that may be defined. That is, a foreign key constraint is non-blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected and thus an error will be added to the validation report.
- Foreign key constraints are not in blocking mode upon archive import. Indeed, all blocking constraints, excepted structural constraints, are always disabled when importing archives. This allows flexibility upon archive import where under certain circumstances the import of foreign keys referencing records that are not yet imported must be tolerant.

Constraints on the whole table

Programmatic **constraints** `Constraint`API are checked on each record of the table at validation time. If the table defines millions of records, this becomes a performance issue. It is then recommended to define a **table-level constraint** `ConstraintOnTable`API.

In the case where it is not possible to define such a table-level constraint, it is recommended to at least define a **local or explicit dependency** `DependenciesDefinitionContext.dependencies`API, so as to reduce the cost of incremental validation.

See also *`ConstraintOnTable`*API

44.4 SQL access to data in relational mode

This section describes how to directly access the data in relational mode, through SQL.

See also *SQL access to history [p 254]*

Finding the table in the database

For every EBX table in relational mode, a corresponding table is generated in the RDBMS. Using the EBX user interface, you can find the name of this database table by clicking on the documentation pane of the table.

SQL reads

Direct SQL reads are possible in well-managed, preferably short-lived transactions. However, for such accesses, EBX permissions are not taken into account. As a result, applications given allowed to perform reads must be trusted through other authentication processes and permissions.

SQL writes

Direct SQL writes bypass the governance framework of EBX. Therefore, they must be used with ***extreme caution***. They could cause the following situations:

- failure to historize EBX tables;
- failure to execute EBX triggers;
- failure to verify EBX permissions and constraints;
- modifications missed by the incremental validation process;
- losing visibility on EBX semantic tables, which might be referenced by foreign keys.

Consequently, direct SQL writes are to be performed *if, and only if, all the following conditions are verified*:

- The written tables are not historized and have no EBX triggers.
- The application performing the writes can be fully trusted with the associated permissions, to ensure the integrity of data. Specifically, the integrity of foreign keys (`osd:tableRef`) must be preserved at all times. See Foreign key blocking mode [p 247] for more information.
- The application server running EBX is shut down *whenever writes are performed*. This is to ensure that incremental validation does not become out-of-date, which would typically occur in a batch context.

44.5 Limitations of relational mode

The relational mode feature is fully functional, but has some known limitations, which are listed below. If using relational mode, it is strongly recommended to read these limitations carefully and to contact the TIBCO EBX Support team at https://support.tibco.com in case of questions.

See Supported databases [p 313] for the databases on which relational mode is supported.

Data model restrictions for tables in relational mode

Some restrictions apply to data models in relational mode:

- Data model restrictions due to mapped mode [p 243]
- Aggregated lists [p 490] are not supported in relational tables. Such a schema will cause a compilation error.
- User-defined attributes on relational tables result in data model compilation errors.

- Dataset inheritance [p 271].
- Inherited fields [p 272].
- Programmatic constraints, since the computation cost of validation would be too high. However, **constraints on tables** `ConstraintOnTable`API remain available.

Schema evolutions may also be constrained by the underlying RDBMS, depending on the data already contained in the concerned tables.

See also*Data model evolutions [p 265]*

Other limitations of relational mode

- Limitations of mapped mode [p 243]
- From a dataspace containing datasets in relational mode, it is not possible to create child dataspaces and snapshots.
- For D3, it is not possible to broadcast a dataspace defined in relational mode.
- For very large volumes of data, the validation will show poor performance if the relational table declares any of these features: `osd:function`, `osd:select`, `osd:uiFilter`, `osd:tableRef/filter`. Additionally, a sort cannot be applied on a `osd:function` column.
- It is not possible to set the `AdaptationValue.INHERIT_VALUE` to a node belonging to a data model in relational mode.

CHAPTER 45
History

This chapter contains the following topics:

45.1 Overview

History is a feature allowing to track all data modifications on a table (records creation, update and deletion).

It is an improvement on the XML audit trail [p 419]. XML audit trail is still activated by default; it can be safely deactivated if history is enabled for the relevant tables.

See also

History [p 28]

Relational mode [p 245]

Replication [p 259]

Data model evolutions [p 265]

45.2 Configuring history

In order to activate historization for a table, a history profile has to be set for the table in the data model. This section describes history profiles and the way they are associated with tables.

Configuring history in the repository

A history profile specifies when the historization is to be created. In order to edit history profiles, select *Administration > History and logs*.

A history profile is identified by a name and defines the following information:

- An internationalized label.

- A list of dataspaces (branches) for which history is activated. It is possible to specify whether direct children and/or all descendants should also be concerned.

Some profiles are already created when installing the repository. These profiles can neither be deleted nor modified.

Profile Id	**Description**
`ebx-referenceBranch`	This profile is activated only on the reference dataspace.
`ebx-allBranches`	This profile is activated on all dataspaces.
`ebx-instanceHeaders`	This profile historizes dataset headers. However, this profile will only be setup in a future version, given that the internal data model only defines dataset nodes.

Configuring history in the data model

Activating table history

History can be activated on a table either through the data model assistant, or by editing the underlying data model.

To activate history by editing the data model, a history profile should be declared on the table using the `historyProfile` element.

```
<osd:table>
 <primaryKeys>/key</primaryKeys>
 <historyProfile>historyProfileForProducts</historyProfile>
</osd:table>
```

The data model assistant allows you to view the historization profiles defined in the repository.

Historization must be activated for each table separately. See model design [p 476] documentation for more details.

Disabling history on a specific field or group

For a historized table, the default behavior is to historize all its supported elements (see Impacts and limitations of historized mode [p 256]).

It is possible to disable history for a given field or group, either through the data model assistant, or by editing the underlying data model.

To disable the history of a field or group by editing the data model, use the element `osd:history` with the attribute `disable="true"`.

```
<xs:element name="longDescription" type="xs:string">
    <xs:annotation>
        <xs:appinfo>
            <osd:history disable="true" />
        </xs:appinfo>
    </xs:annotation>
</xs:element>
```

To disable the history of a field or group through the data model assistant, use the `History` property in the `Advanced properties` of the element.

When this property is defined on a group, history is disabled recursively for all its descendants. Once a group disables history, it is not possible to specifically re-enable history on a descendant.

> **Note**
>
> If the table containing the field or group is not historized, this property will not have any effect.
>
> It is not possible to disable history for primary key fields.

Integrity

If problems are detected at data model compilation, warning messages or error messages will be added to the validation report associated with this data model. Furthermore, if any error is detected, each associated instance (dataset) will be inaccessible. The most common error cases are the following:

- A table references a profile that is not defined in the repository.
- A history profile that is referenced in the data model mentions a non-defined or closed dataspace in the current repository.

> **Note**
>
> Deploying a data model on a repository that does not have the expected profiles requires the administrator to add them.

45.3 History views and permissions

Table history view

When the history has been activated on a table in the data model, it is possible to access the history view from various locations in the user interface: record, selection of records, table and dataset.

The next section explains how permissions are resolved.

For more information, see table history view [p 29] section. To access the table history view from Java, the method `AdaptationTable.getHistory`API must be invoked.

Permissions for table history

Data permissions are also applied to data history. History permissions are resolved automatically as the most restricted permission between data permissions and *read-only* access right.

This is true for user-defined permission rules and also for programmatic permission rules.

When defining a programmatic rule, it may be required to distinguish between the functional dataset context and the history view context, either because the expected permissions are not the same, or because some fields are not present in the history structure. This is the case for dataset fields, computed values and fields for which history has been disabled [p 252]. The methods `Adaptation.isHistory`API and `AdaptationTable.getHistory`API can then be used in the programmatic rule in order to implement specific behavior for history.

Transaction history views

The transaction history view gives access to the executed transactions, independently of a table, a dataset or a data model, directly from the user interface.

To see the 'Transaction history' table, navigate to the Administration area and select 'History and logs' using the down arrow menu in the navigation pane. Transaction history can also be accessed from the Dataspaces area by selecting a historized dataspace and using the **Actions** menu in the workspace.

For more information, see transaction history view [p 29].

45.4 **SQL access to history**

This section describes how to directly access the history data by means of SQL.

See also*SQL access to data in relational mode [p 248]*

Access restrictions

The database tables must be accessed only in read-only mode. It is up to the database administrator to forbid write access except for the database user used by TIBCO EBX, as specified in the section Rules for the database access and user privileges [p 373].

Relational schema overview

Here is a description of the history tables in the database.

The database schema contains (see also the diagram in the next section):

Common and generic tables	The main table is `HV_TX`; each record of this table represents a transaction. Only transactions that involve at least one historized table are recorded. These common tables are all prefixed by "HV".
Specific generated tables	For each historized table, a specific history table is generated. This table contains the history of the data modifications on the table. In the EBX user interface, the name of this table in database can be obtained by clicking on the table documentation pane (advanced mode). All the specific history tables are prefixed with "HG".

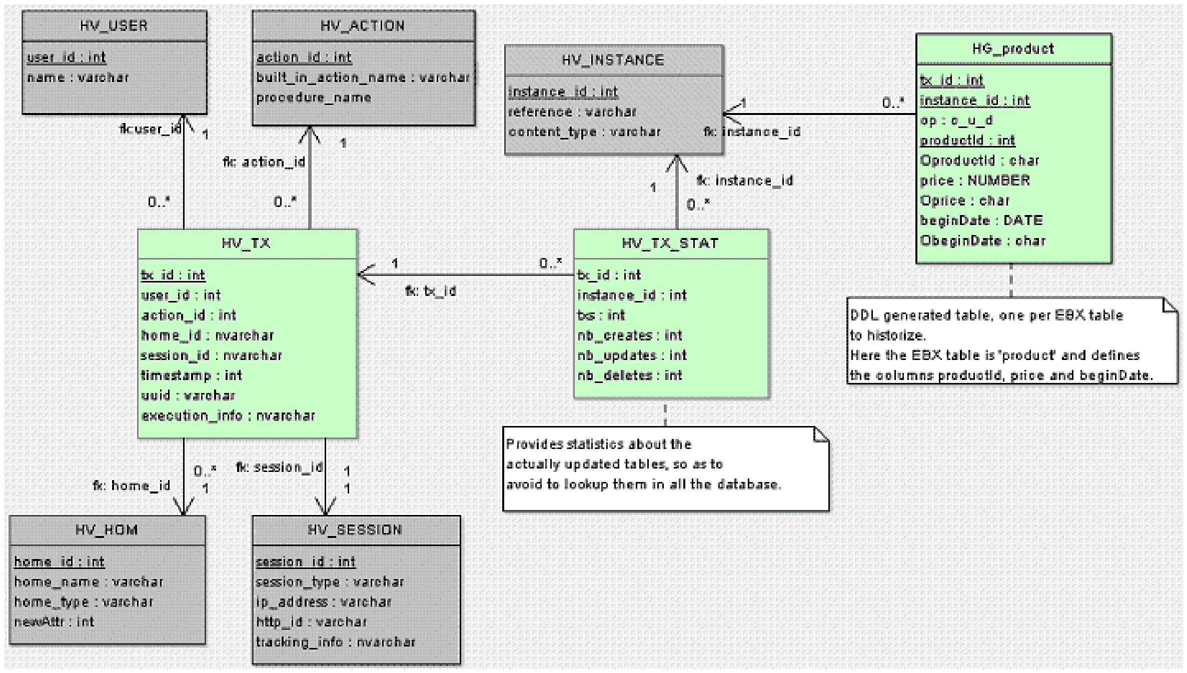
Example of a generated history table

In the following example, we are historizing a table called `product`. Let us assume this table declares three fields in EBX data model:

Product

- productId: int
- price: int
- beginDate: Date

The diagram below shows the resulting relational schema:



Activating history on this table generates the HG_product table shown in the history schema structure above. Here is the description of its different fields:

- `tx_id`: transaction ID.
- `instance`: instance ID.
- `op`: operation type - C (create), U (update) or D (delete).
- `productId`: `productId` field value.
- `OproductId`: operation field for `productId`, see next section.
- `price`: `price` field value.
- `Oprice`: operation field for `price`, see next section.
- `beginDate`: `date` field value.
- `ObeginDate`: operation field for `beginDate`, see next section.

Combination of operations

If several operations are combined in the same transaction, the operation field is resolved as follows:

-
  ```
  C + U -> C
  D + U -> D
  D + C -> U
  C + D -> {} (no entry in history)
  ```

Values for operation fields

For each functional field, an additional operation field is defined, composed of the field name prefixed by the character `O`. This field specifies whether the functional field has been modified. It is set to one of the following values:

- `null`: if the functional field value has not been modified (and its value is not INHERIT).
- `M`: if the functional field value has been modified (not to INHERIT).
- `D`: if record has been deleted.

If inheritance [p 270] is enabled, the operation field can have three additional values:

- `T`: if the functional field value has not been modified and its value is INHERIT.
- `I`: if the functional field value has been set to INHERIT.
- `O`: if the record has been set to OCCULTING mode.

45.5 Impacts and limitations of historized mode

The history feature has some impacts and known limitations, which are listed in this section. If using historized mode, it is strongly recommended to read these limitations carefully and to contact TIBCO Software Inc. support in case of questions.

Validation

Some EBX data model constraints become blocking constraints when table history is activated. For more information, see the section Structural constraints [p 247].

Data model restrictions for historized tables

Some restrictions apply to data models containing historized tables:

- Data model restrictions due to mapped mode [p 243]
- Limitations exist for two types of aggregated lists: aggregated lists under another aggregated list, and aggregated lists under a terminal group. Data models that contain such aggregated lists can be used, however these lists will be ignored (not historized).
- Computed values are ignored.
- User-defined attributes on historized tables result in data model compilation errors.

Data model evolutions may also be constrained by the underlying RDBMS, depending on the data already contained in the concerned tables.

See also*Data model evolutions [p 265]*

Other limitations of historized mode

- No data copy is performed when a table with existing data is activated for history.
- Global operations on datasets are not historized (create an instance and remove an instance), even if they declare a historized table.
- Default labels referencing a non-historized field are not supported for historized tables.

  As a consequence, default labels referencing a computed field are not supported for historized tables.

  The workaround is to implement the `UILabelRenderer` interface and adapt the label computation for history.
- D3: the history can be enabled in the delivery dataspace of a primary node, but in the delivery dataspace of the replica nodes, the historization features are always disabled.

- Recorded user in history: for some specific operations, the user who performs the last operation and the one recorded in the corresponding history record may be different.

  This is due to the fact that these operations are actually a report of the data status at a previous state:

  - Archive import: when importing an archive on a dataspace, the time and user of the last operation performed in the child dataspace are preserved, while the user recorded in history is the user who performs the import.
  - Programmatic merge: when performing a programmatic merge on a dataspace, the time and user of the last operation performed in the child dataspace are preserved, while the user recorded in history is the user who performs the merge.
  - D3: for distributed data delivery feature, when a broadcast is performed, the data from the primary node is reported on the replica node and the time and user of the last operation performed in the child dataspace are preserved, while the user recorded in history is 'ebx-systemUser' who performs the report on the replica node upon the broadcast.

CHAPTER 46
Replication

This chapter contains the following topics:

46.1 Overview

Data stored in the TIBCO EBX repository can be mirrored to dedicated relational tables to enable direct access to the data by SQL requests and views.

Like history and relational mode, this data replication is transparent to end-users and client applications. Certain actions trigger automatic changes to the replica in the database:

- Activating replication at the model-level updates the database schema by automatically executing the necessary DDL statements.
- Data model evolutions that impact replicated tables, such as creating a new column, also automatically update the database schema using DDL statements.
- When using the 'onCommit' refresh mode: updating data in the EBX repository triggers the associated inserts, updates, and deletions on the replica database tables.

See also

Relational mode *[p 245]*

History *[p 251]*

Data model evolutions *[p 265]*

Repository administration *[p 372]*

Note

replicated table: refers to a primary data table that has been replicated

replica table (or *replica*): refers to a database table that is the target of the replication

46.2 **Configuring replication**

Enabling replication

To define a *replication unit* on a data model, use the element `osd:replication` under the elements `annotation/appinfo`. Each replication unit specifies tables in a single dataset in a specific dataspace.

The nested elements are as follows:

Element	**Description**	**Required**
`name`	Name of the replication unit. This name identifies a replication unit in the current data model. It must be unique.	Yes
`dataSpace`	Specifies the dataspace relevant to this replication unit. It cannot be a snapshot or a relational dataspace.	Yes
`dataSet`	Specifies the dataset relevant to this replication unit.	Yes
`refresh`	Specifies the data synchronization policy. The possible policies are: • `onCommit`: The replica table content in the database is always up to date with respect to its source table. Every transaction that updates the EBX source table triggers the corresponding insert, update, and delete statements on the replica table. • `onDemand`: The replication of specified tables is only done when an explicit refresh operation is performed. See Requesting an 'onDemand' replication refresh [p 262].	Yes
`table/path`	Specifies the path of the table in the current data model that is to be replicated to the database.	Yes
`table/nameInDatabase`	Specifies the name of the table in the database to which the data will be replicated. This name must be unique amongst all replications units.	Yes
`table/element/path`	Specifies the path of the aggregated list in the table that is to be replicated to the database.	Yes
`table/element/nameInDatabase`	Specifies the name of the table in the database to which the data of the aggregated list will be replicated. This name must be unique amongst all replications units.	Yes

For example:

```
<xs:schema>
 <xs:annotation>
  <xs:appinfo>
   <osd:replication>
    <name>ProductRef</name>
    <dataSpace>ProductReference</dataSpace>
    <dataSet>productCatalog</dataSet>
    <refresh>onCommit</refresh>
    <table>
     <path>/root/domain1/tableA</path>
     <nameInDatabase>PRODUCT_REF_A</nameInDatabase>
    </table>
    <table>
     <path>/root/domain1/tableB</path>
     <nameInDatabase>PRODUCT_REF_B</nameInDatabase>
     <element>
      <path>/retailers</path>
      <nameInDatabase>PRODUCT_REF_B_RETAILERS</nameInDatabase>
     </element>
    </table>
   </osd:replication>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:schema>
```

Notes:

- See Data model restrictions for replicated tables [p 262]
- If, at data model compilation, the specified dataset and/or dataspace does not exist in the current repository, a warning is reported, but the replica table is created in the database. Once the specified dataspace and dataset are created, the replication becomes active.
- At data model compilation, if a table replication is removed, or if some of the above properties has changed, the replica table is dropped from the database, and then recreated with the new definition if needed.

Disabling replication on a specific field or group

For a replicated table, the default behavior is to replicate all its supported elements (see Data model restrictions for replicated tables [p 262]).

It is possible to disable replication for a specific field or group, either through the data model assistant, or by editing the underlying data model.

To disable the replication of a field or group by editing the data model, use the element `osd:replication` with the attribute `disable="true"`.

```
<xs:element name="longDescription" type="xs:string">
    <xs:annotation>
        <xs:appinfo>
            <osd:replication disable="true" />
        </xs:appinfo>
    </xs:annotation>
</xs:element>
```

To disable the replication of a field or group through the data model assistant, use the `Replication` property in the `Advanced properties` of the element.

When this property is defined on a group, replication is disabled recursively for all its descendents. Once a group disables replication, it is not possible to specifically re-enable replication on a descendant.

> **Note**
>
> If the table containing the field or group is not replicated, this property will not have any effect.
>
> It is not possible to disable replication for primary key fields.

46.3 **Accessing a replica table using SQL**

This section describes how to directly access a replica table using SQL.

See also *SQL access to history [p 254]*

Finding the replica table in the database

For every replicated EBX table, a corresponding table is generated in the RDBMS. Using the EBX user interface, you can find the name of this database table by clicking on the documentation pane of the table.

Access restrictions

The replica database tables must only be directly accessed in read-only mode. It is the responsibility of the database administrator to block write-access to all database users except the one that EBX uses.

See also*Rules for the database access and user privileges* [p 373]

SQL reads

Direct SQL reads are possible in well-managed, preferably short-lived transactions. However, for such accesses, EBX permissions are not taken into account. As a result, applications given the privilege to perform reads must be trusted through other authentication processes and permissions.

46.4 Requesting an 'onDemand' replication refresh

The 'onDemand' refresh policy requires an explicit request to refresh the replicated table data.

There are several ways to request a replication refresh:

- **User interface**: In the dataset actions menu, use the action 'Refresh replicas' under the group 'Replication' to launch the replication refresh wizard.
- **Data services**: Use the replication refresh data services operation. See Replication refresh [p 642] for data services for more information.
- **Java API**: Call the `ReplicationUnit.performRefresh`API methods in the `ReplicationUnit` API to launch a refresh of the replication unit.

46.5 Impact and limitations of replication

The replication feature has some known limitations and side-effects, which are listed below. If using replication, it is strongly recommended to read this section carefully and to contact TIBCO Software Inc. support in case of questions.

See Supported databases [p 313] for the databases for which replication is supported.

Validation

Some EBX data model constraints become blocking constraints when replication is enabled. For more information, see Structural constraints [p 247].

Data model restrictions for replicated tables

Some restrictions apply to data models containing tables that are replicated:

- Data model restrictions due to mapped mode [p 243]
- Dataset inheritance is not supported for the 'onCommit' refresh policy if the specified dataset is not a root dataset or has not yet been created. See dataset inheritance [p 271] for more information.

- Field inheritance is also only supported for the 'onDemand' refresh policy. This means that, at data model compilation, an error is reported if the refresh mode is 'onCommit' and the table to be replicated has an inherited field. See inherited fields [p 272] for more information.
- Computed values are ignored.
- Limitations exist for two types of aggregated lists: aggregated lists under another aggregated list, and aggregated lists under a terminal group. Data models that contain such aggregated lists can be used, however these lists will be ignored (not replicated).
- User-defined attributes are not supported. A compilation error is raised if they are included in a replication unit.

Data model evolutions may also be constrained by the underlying RDBMS, depending on the data already contained in the concerned tables.

See also*Data model evolutions [p 265]*

Database configuration

The refresh operation is optimized to transmit only the rows of the source table that have been modified (with respect to creation and deletion) since the last refresh. However, depending on the volume of data exchanged, this can be an intensive operation, requiring large transactions. In particular, the first refresh operation can concern a large number of rows. It is necessary for the database to be configured properly to allow such transactions to run under optimal conditions.

For instance, with Oracle:

- It is mandatory for the bulk of all replica tables in a replication unit to fit into the 'UNDO' tablespace.
- It is recommended to provide enough space in the buffer cache to allow those transactions to run with minimal disk access.
- It is recommended to provision 'REDO' log groups big enough to avoid those transactions to wait for the 'db_writer' process.

Distributed data delivery (D3)

Replication is available on both D3 primary and replica delivery dataspaces. On the primary dataspace, the replication behavior is the same as in a standard semantic dataspace, but on replica dataspaces, the replicated content is that of the last broadcast snapshot.

In a replica delivery dataspace, some restrictions occur:

- The refresh policy defined in the data model has no influence on the behavior described above: replication always happens on snapshot.
- The action item `Refresh replicas` is not available.
- It is not allowed to invoke the `ReplicationUnit.performRefresh`API method.

See also*D3 overview [p 424]*

Other limitations of replication

- Limitations of supported databases [p 313]

- For inheritance, a replica record field cannot hold the "inherit value" flag (`AdaptationValue.INHERIT_VALUE`). It only holds the inherited value in such cases. More generally, it is not possible to distinguish inheriting state from overwriting state.

CHAPTER **47**

Data model evolutions

This chapter describes the modifications that are possible on data models, as well as potential limitations. The restrictions and/or potential impacts of data model evolutions depend on the persistence mode. The principles for each mode are the following:

- Semantic mode: flexible and non-blocking. Can lead to a loss of data; for instance, a primary key definition can freely evolve, but all existing records in any dataspace and snapshot that violate the primary key constraint will no longer be loaded.
- Any mapped mode: restrictive and thus blocking if data exists and if the evolution would violate their integrity according to the new data model.

Attention

Whenever the data modeler performs an evolution on the data model, it is important to anticipate the fact that it could lead to a loss of data. In such cases, if existing data must be preserved in some ways, a data migration plan must be set up and operated before the new data model is published or deployed. It can also be noted that data is not destroyed immediately after the data model evolution; in semantic mode, as long as no update is performed on a table whose definition has evolved, if the data model is rolled back to its previous state, then the previous data is retrieved.

Note

Certain types of data model evolutions cannot be performed directly in the user interface, and thus the data model must be exported, modified in XSD format, then re-imported. For changes to a data model that impact its configuration, not just its structure, the XSD must be imported into TIBCO EBX from a module. Otherwise, the configuration modifications are not taken into account.

See also*Mapped mode* *[p 243]*

This chapter contains the following topics:

47.1 Types of permitted evolutions

This section describes the possible modifications to data models after their creation.

Model-level evolutions

The following modifications can be made to existing data models:

- A data model in semantic mode can be declared to be in relational mode. Data should be manually migrated, by exporting then re-importing an XML or archive file.
- Relational mode can be disabled on the data model. Data should be manually migrated, by exporting then re-importing an XML or archive file.
- Replication units can be added to the data model. If their refresh policy is 'onCommit', the corresponding replica tables will be created and refreshed on next schema compilation.
- Replication units can be removed from the data model. The corresponding replica tables will be dropped immediately.
- The data model can be deleted. If it declares replication units, the corresponding replica tables will be dropped immediately. If it is relational or contains historized tables, this change marks the associated mapped tables as disabled. See Database mapping [p 405] for the actual removal of associated database objects.

Table-level evolutions

The following modifications can be made to a data model at the table-level:

- A new table can be added. Upon creation, the table can also declare one or more mapped modes.
- An existing table can be deleted. If it declares replication units, the corresponding replica tables will be dropped immediately. If it historized or relational, this change marks the mapped table as disabled. See Database mapping [p 405] for the actual removal of associated database objects.
- An existing table in semantic mode can be declared to be in relational mode. Data should be manually migrated, by exporting then re-importing an XML or archive file.
- History can be enabled or disabled on a table. History will not take into account the operations performed while it is disabled.
- A table can be renamed. Data should be manually migrated, by exporting then re-importing an XML or archive file, because this change is considered to be a combination of deletion and creation.

Field-level evolutions

The following modifications can be made to a data model at the field-level:

- A new field can be added.
- An existing field can be deleted. In semantic mode, the data of the deleted field will be removed from each record upon its next update. For a replica table, the corresponding column is automatically removed. In history or relational mode, the field is marked as disabled.
- A field can be specifically disabled from the history or replication which applies to its containing table, by using the attribute `disable="true"`. For a replica table, the corresponding column is automatically removed. For a history table, the column remains but is marked as disabled. See Disabling history on a specific field or group [p 252] and Disabling replication on a specific field or group [p 261].
- The facets of a field can be modified, except for the facets listed under Limitations/restrictions [p 267].

The above-mentioned changes are accepted, but they can lead to a loss of data. Data should be migrated manually, by exporting then re-importing an XML or archive file, since these changes are considered to be a combination of deletion and creation.

- A field can be renamed.
- The type of a field can be changed.

Index-level evolutions

- An index can be added or renamed.
- An index can be modified, by changing or reordering its fields. In mapped mode, the existing index is deleted and a new one is created.
- An index can be deleted. In mapped mode, a deleted index is also deleted from the database.

47.2 Limitations/restrictions

> **Note**
>
> All limitations listed in this section that affect mapped mode can be worked around by purging the mapped table database resources. For the procedure to purge mapped table database resources, see Database mapping [p 405].

Limitations related to primary key evolutions

When a primary key definition is modified:

- In semantic mode, the existing records are only loaded into the cache if they:
  - Respect the uniqueness constraint of the primary key,
  - Comply with the new structure of the primary key.
- In mapped mode, the underlying RDBMS only accepts a primary key modification if all table records respect its uniqueness and non-nullity constraints. In particular, if a table already has existing records:
  - Adding a new field to the primary key requires assigning a default value to this field. Workaround for replicated or relational tables: first add the field, value it for the existing records, then add the field to the primary key.
  - Removing an existing field from the primary key will be rejected if it would cause the existing records to no longer have a unique primary key (assigning a default value makes no change in this case).
  - It is generally not possible to rename a field of the primary key; more formally, it is only possible if the field was not needed for making all primary keys unique. Indeed, renaming a field translates to a combination of deletion and creation; consequently, the operation will be rejected if it would cause the existing records to no longer have a unique primary key (assigning a default value makes no change in this case).

Limitations related to foreign key evolutions

- When the declaration of a facet `osd:tableRef` is added or modified, or the primary key of the target table of a facet `osd:tableRef` is modified:

- In semantic mode, the existing values for this field are only loaded into the cache if they comply with the new structure of the target primary key.
- In mapped mode, the structure of a foreign key field is set to match that of the target primary key. A single field declaring an `osd:tableRef` constraint may then be split into a number of columns, whose number and types correspond to that of the target primary key. Hence, the following cases of evolutions will have an impact on the structure of the mapped table:
  - declaring a new `osd:tableRef` constraint on a table field;
  - removing an existing `osd:tableRef` constraint on a table field;
  - adding (resp. removing) a column to (resp. from) a primary key referenced by an existing `osd:tableRef` constraint;
  - modifying the type or path for any column of a primary key referenced by an existing `osd:tableRef` constraint.

  These cases of evolution will translate to a combination of field deletions and/or creations. Consequently, the existing data should be migrated manually.

Limitations related to field-level evolutions

- In mapped mode, when a `maxLength`, `length`, `totalDigits` or `fractionDigits` facet is modified:

  Whether or not this modification is accepted depends on the underlying DBMS, as well as the field type and the contents of the table.

  For example, Oracle will accept changing a VARCHAR(20) to a VARCHAR(50), but will only change a VARCHAR(50) to a VARCHAR(20) if the table does not contain any values over 20 characters long.

  PostgreSQL has the same limitations, but additionally, any modification of a field type (including modifications of its length) will invalidate all related prepared statements, and require restarting the application server.
- When a cardinality of an element is modified:
  - In semantic mode, this change is supported. However, two cases are distinguished:
    - When changing a single element to an aggregated list, the previous single value is preserved and added to the new aggregated list.
    - When changing an aggregated list to a single element, only the last value of the aggregated list is preserved in the single element. Other values are lost.
  - In relational mode, aggregated lists are not supported. An error message is added to the compilation report of the data model if an element is changed to an aggregated list.
  - In historized mode, when changing a single element to an aggregated list, the modification is taken into account, but the previous single value is lost.

Other

CHAPTER 48

Inheritance and value resolution

This chapter contains the following topics:

48.1 Overview

The principle of inheritance is to mutualize resources that are shared by multiple contexts or entities. TIBCO EBX offers mechanisms for defining, factorizing and resolving data values: *dataset inheritance* and *inherited fields*.

Furthermore, *functions* can be defined to compute values.

> **Note**
>
> Inheritance mechanisms described in this chapter should not be confused with "structural inheritance", which usually applies to models and is proposed in UML class diagrams for example.

See also*Inheritance (glossary)* *[p 27]*

Dataset inheritance

Dataset inheritance is particularly useful when data applies to global enterprise contexts, such as subsidiaries or business partners.

Based on a hierarchy of datasets, it is possible to factorize common data into the root or intermediate datasets and define specialized data in specific contexts.

The dataset inheritance mechanisms are detailed below in Dataset inheritance [p 271].

Inherited fields

Contrary to dataset inheritance, which exploits global built-in relationships between datasets, inherited fields exploit finer-grained dependencies that are specific to the data structure. It allows factorizing and specializing data at the business entities-level.

For example, if the model specifies that a 'Product' is associated with a 'FamilyOfProducts', it is possible that some attributes of 'Product' inherit their values from the attributes defined in its associated 'FamilyOfProducts'.

> **Note**
>
> When using both inheritance in the same dataset, field inheritance has priority over the dataset one.

Computed values (functions)

In the data model, it is also possible to specify that a node holds a *computed value*. In this case, the specified JavaBean function will be executed each time the value is requested.

The function is able to take into account the current context, such as the values of the current record or computations based on another table, and to send requests to third-party systems.

See also Computed values *[p 527]*

48.2 Dataset inheritance

Dataset inheritance declaration

The dataset inheritance mechanism is declared as follows in a data model:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ebxbnd="urn:ebx-schemas:binding_1.0">
  <xs:annotation>
    <xs:appinfo>
      <osd:inheritance>
        <dataSetInheritance>all</dataSetInheritance>
      </osd:inheritance>
    </xs:appinfo>
  </xs:annotation>
   ...
</xs:schema>
```

The element `osd:inheritance` defines the property `dataSetInheritance` to specify the use of inheritance on datasets based on this data model. The following values can be specified:

- `all`, indicates that inheritance is enabled for all datasets based on the data model.
- `none`, indicates that inheritance is disabled for all datasets based on the data model.

If not specified, the inheritance mechanism is disabled.

Value lookup mechanism

The dataset inheritance lookup mechanism for values proceeds as follows:

1. If the value is locally defined, it is returned.

   It can be explicitly `null`.
2. Otherwise, looks up the first locally defined value according to the built-in child-to-parent relationship of the dataset in the hierarchy of datasets.
3. If no locally defined value is found, the default value is returned.

   If no default value is defined, `null` is returned.

   Note: Default values cannot be defined on:

   - A single primary key node

- Auto-incremented nodes
- Nodes defining a computed value

Record lookup mechanism

Like values, table records can also be inherited as a unit by multiple contexts, but they can also be partially redefined (*overwritten*), defined for a specific context (*root mode*), or be *occulted*.

Formally, a table record has one of four distinct definition modes:

root record	Locally defined in the table and has no parent. This means that no record with the same primary key exists in the parent table, or that this parent is an occulting record.
overwriting record	Locally defined in the table and has a parent record. This means that a record with the same primary key exists in the parent table, and that this parent is not an occulting record. The overwriting record inherits its values from its parent, except for the values that it explicitly redefines.
inherited record	Not locally defined in the current table and has a parent record. All values are inherited. Functions are always resolved in the current record context and are not inherited.
occulting record	Specifies that, if a parent with the same primary key is defined, this parent will not be visible in table descendants.

See also*Dataset inheritance* *[p 127]*

Defining inheritance behavior at the table level

It is also possible to specify management rules in the declaration of a table in the data model.

See also*Properties related to dataset inheritance* *[p 497]*

48.3 Inherited fields

The specific inheritance mechanism allows fetching a value of a field according to its relationship to other tables.

Field inheritance declaration

Specific inheritance must be specified on terminal nodes in the underlying data model and is declared as follows:

```
<xs:element name="sampleInheritance" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:inheritance>
    <sourceRecord>
     /root/table1/fkTable2, /root/table2/fkTable3
    </sourceRecord>
    <sourceNode>color</sourceNode>
   </osd:inheritance>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

The element `sourceRecord` is an expression that describes how to look up the record from which the value is inherited. It is a foreign key, or a sequence of foreign keys, from the current element to the source table.

If `sourceRecord` is not defined in the data model, the inherited fields are fetched from the current record.

The element `sourceNode` is the path of the node from which to inherit in the source record.

The following conditions must be satisfied for specific inheritance:

- The element `sourceNode` is mandatory.
- The expression for the path to the source record must be a consistent path of foreign keys, from the current element to the source record. This expression must involve only one-to-one and zero-to-one relationships.
- The `sourceRecord` cannot contain any aggregated list elements.
- Each element of the `sourceRecord` must be a foreign key.
- If the inherited field is also a foreign key, the `sourceRecord` cannot refer to itself to get the path to the source record of the inherited value.
- Every element of the `sourceRecord` must exist.
- The source node must belong to the table containing the source record.
- The source node must be terminal.
- The source node must be writeable.
- The source node type must be compatible with the current node type.
- The source node cardinalities must be compatible with those of the current node.
- The source node cannot be the same as the inherited field if the fields to inherit from are fetched into the same record.

Value lookup mechanism

The lookup mechanism for inherited fields values proceeds as follows:

1. If the value is locally defined, it is returned.

   It can be explicitly `null`
2. Otherwise, looks up the source record and value to inherit from, according to the properties that are defined in the data model.
3. The process is recursive; if the source node does not locally define a value, it is then looked up according to the inheritance behavior of the source node.

48.4 Optimize & Refactor service

EBX provides a built-in user service for optimizing the dataset inheritance in the hierarchy of datasets. This service performs the following functions:

- **Handles duplicated values:** Detects and removes all parameter values that are duplicates of the inherited value.
- **Mutualizes common values:** Detects and mutualizes the common values among the descendants of a common ancestor.

Procedure details

Datasets are processed from the bottom up, which means that if the service is run on the dataset at level *N*, with *N+1* being the level of its children and *N+2* being the level of its children's children, the service will first process the datasets at level *N+2* to determine if they can be optimized with respect to the datasets at level *N+1*. Next, it would proceed with an optimization of level *N+1* against level *N*.

> **Note**
>
> - These optimization and refactoring functions do not handle default values that are declared in the data model.
> - The highest level considered during the optimization procedure is always the dataset on which the service is run. This means that optimization and refactoring are not performed between the target dataset and its own ancestors.
> - Table optimization is performed on records with the same primary key.
> - Inherited fields are not optimized.
> - *The optimization and refactoring functions do not modify the resolved view of a dataset, if it is activated.*

Service availability

The 'Optimize & Refactor' service is available on datasets that have child datasets and have the 'Activated' property set to 'No' in their dataset information.

The service is available to any profile with write access on current dataset values. It can be disabled by setting restrictive access rights on a profile.

> **Note**
>
> For performance reasons, access rights are not verified on every node and table record.

CHAPTER **49**

Permissions

Permissions dictate the access each user has to data and actions.

This chapter contains the following topics:

49.1 Overview

Permissions are related to whether actions are authorized or not. They are also related to access rights, that is, whether an entity is hidden, read, or read-write. The main entities controlled by permissions are:

- Dataspace
- Dataset
- Table
- Group
- Field

Users, roles and profiles

The definition and resolution of permissions make extensive use of the notion of *profiles*, which is the generic term applied to users or roles.

Each user can participate in several roles, and a role can be shared by several users.

These relationships are defined in the user and roles directory. See Users and roles directory [p 399].

Special definitions:

- An *administrator* is a member of the built-in role 'ADMINISTRATOR'.
- An *owner of a dataset* is a member of the *owner* attribute specified in the information of a root dataset. In this case, the built-in role 'OWNER' is activated when permissions are resolved in the context of the dataset.

- An *owner of a dataspace* is a member of the *owner* attribute specified for a dataspace. In this case, the built-in role 'OWNER' is activated when permissions are resolved in the context of the dataspace.

Permission rules

A permission rule defines the authorization granted to a profile for a particular entity.

User-defined permission rules are created through the user interface. See the section Defining user-defined rules [p 278].

Programmatic permission rules can be created by developers. See the section Defining programmatic rules [p 282].

Resolution of permissions

Permissions are always resolved in the context of an authenticated user session, thus permissions are mainly based on the user profiles.

In general, resolution of permissions is performed restrictively between a given level and its parent level. Thus, at any given level, a user cannot have a higher permission than the one resolved at a parent level.

Programmatic permissions are always considered to be restrictive.

> **Note**
>
> In the Java API, the class `SessionPermissions`API provides access to the resolved permissions.

See also

Resolving permissions on data [p 283]

Resolving permissions on services [p 287]

Resolving permissions on actions [p 289]

Owner and administrator special permissions

On a dataset

An administrator or owner of a dataset can perform the following actions:

- Manage its permissions
- Change its owner, if the dataset is a root dataset
- Change its general information (localized labels and descriptions)

Attention

While the definition of permissions can restrict an administrator or dataset owner's right to view data or perform certain actions, it remains possible for them to modify their own access, as they will always have access to permissions management.

On a dataspace

To be a *super owner* of a dataspace, a user must either:

- Own the dataspace and be allowed to manage its permissions, or
- Own a dataspace that is an ancestor of the current dataspace and be allowed to manage the permissions of that ancestor dataspace.

An administrator or super owner of a dataspace can perform the following actions:

- Manage its permissions of dataspace.
- Change its owner
- Lock it or unlock it
- Change its general information (localized labels and descriptions)

Furthermore, in a workflow, when using a "Create a dataspace" or "Create a snapshot" built-in script task, resolved permissions are computed using the owner defined in the script task's configuration, rather than the current session. This is because, in these cases, the current session is associated with a system user.

Attention

While the definition of permissions can restrict an administrator or dataspace owner's right to view data or perform certain actions, it remains possible for them to modify their own access, as they will always have access to permissions management.

Impact of merge on permissions

When a dataspace is merged, the permissions of the child dataset are merged with those of the parent dataspace if and only if the user specifies to do so during the merge process. The permissions of its parent dataspace are never impacted.

If some elements are hidden for the profile attempting to perform a merge, it will not be possible to proceed as the impacts of the merge on data will not be fully visible.

Important considerations about permissions

The following statements should be kept in mind while working with permissions:

- Whenever the `hidden` permission is returned for a session on a table child node, a user could "guess" sensitive information by filtering or sorting on this node in the following cases:
  - When using a `Request`API on this table with **permissions enabled** `Request.setSession`API. To avoid this, permissions on this node must be checked explicitly before applying the filter or the sort criteria.
  - When defining a custom view on this table in the UI. To avoid this, view definition permissions should be restricted for such users.
- Resolution of custom display labels for tables ('defaultLabel' property) and relationships ('display' property) ignores permission, and fields usually hidden due to access rights restrictions will be displayed in such labels. As a result, these labels should not contain any confidential field. Otherwise, a permission strategy should also be defined to restrict the display of the whole label.
- When a procedure disables all permission checks by using `ProcedureContext.setAllPrivileges`API, the client code must check that the current user session is allowed to run the procedure.

- When performing actions on a table (create, delete, overwrite or occult) in a procedure, the current user session access right on the table node is ignored during the permission resolution. Should this check be performed, the client code must explicitly call `SessionPermissions.getNodeAccessPermission`API beforehand in the procedure.
- To optimize the resolution of permissions for both data and user services, a dedicated cache is implemented at the session level; it only takes user-defined permissions into account, not programmatic rules (which are not cached since they are contextual and dynamic). The session cache life cycle depends on the context, as described hereafter:
  - In the UI, the cache is cleared for every non-ajax event (i.e on page display, pop-up opening, etc.).
  - In programmatic procedures, the cache lasts until the end of the procedure, unless explicitly cleared (see below).

Attention

When modifying permissions in a procedure context (by importing an EBX archive or merging a dataspace programmatically), the session cache **must** be cleared via a call to `Session.clearCache`API. Otherwise, these modifications will not be reflected until the end of the procedure.

49.2 Defining user-defined rules

Each level has a similar schema, which allows defining permission rules for profiles.

Defining dataspace user-defined rules

For a given dataspace, the allowable permissions for each profile are as follows:

Dataspace access	Authorization
Write	• Can view the dataspace. • Can access datasets according to dataset permissions.
Read-only	• Can view the dataspace and its snapshots. • Can view child dataspaces, if allowed by permissions. • Can view contents of the dataspace, though cannot modify them.
Hidden	• Can neither see the dataspace nor its snapshots. • If allowed to view child dataspace, can see the current dataspace but cannot select it. • Cannot access the dataspace contents, including datasets. • Cannot perform any actions on the dataspace.

Restriction policy	Indicates whether this dataspace profile-permission association should have priority over other permissions rules.
Create a child dataspace	Indicates whether the profile can create child dataspaces from the current dataspace.
Create a child snapshot	Indicates whether the profile can create snapshots of the current dataspace.
Initiate merge	Indicates whether the profile can merge the current dataspace with its parent dataspace.
Export archive	Indicates whether the profile can export the current dataspace as an archive.
Import archive	Indicates whether the profile can import an archive into the current dataspace.
Close a dataspace	Indicates whether the profile can close the current dataspace.
Close a snapshot	Indicates whether the profile can close a snapshot of the current dataspace.
Rights on services	Indicates if a profile has the right to execute services on the dataspace. By default, all dataspace services are allowed.

	An administrator or super owner of the current dataspace or a given user who is allowed to modify permissions on the current dataspace can modify these permissions to restrict dataspace services for certain profiles.
Permissions of child dataspace when created	When a user creates a child dataspace, the permissions of this new dataspace are automatically assigned to the profile's owner, based on the permissions defined under 'Permissions of child dataspace when created' in the parent dataspace. If multiple permissions are defined for the owner through different roles, the owner's profile behaves like any other profile and permissions are resolved [p 276] as usual.

Defining dataset user-defined rules

For a given dataset, the allowable permissions for each profile are as follows:

Actions on datasets

Restriction policy	Indicates whether this dataset profile-permission association should have priority over other permissions rules.
Create a child dataset	Indicates whether the profile has the right to create a child dataset of the current dataset.
Duplicate dataset	Indicates whether the profile has the right to duplicate the current dataset.
Change the dataset parent	Indicates whether the profile has the right to change the parent dataset of a given child dataset.

Actions on tables

The action rights on default tables are defined at the dataset level. It is then possible to override these default rights for one or more tables. The allowable permissions for each profile are as follows:

Create a new record	Indicates whether the profile has the right to create records in the table.
Overwrite inherited record	Indicates whether the profile has the right to overwrite inherited records in the table.
Occult inherited record	Indicates whether the profile has the right to occult inherited records in the table.
Delete a record	Indicates whether the profile has the right to delete records in the table.

Access rights on node values

Permissions defined on specific terminal nodes override their default access rights.

Read-write	Can view and modify node values.
Read	Can view nodes, but cannot modify their values.
Hidden	Cannot view nodes.

Permissions on services

An administrator or an owner of the current dataspace can modify the service default permission to either restrict or grant access to certain profiles.

Enabled	Grants service access to the current profile.
Disabled	Forbids service access to the current profile. It will not be displayed in menus, nor will it be launchable via web components.
Default	Sets the service permission to enabled or disabled, according to the default permission defined upon service declaration. See `ActivationContext.setDefaultPermission`API for more information.

49.3 **Defining programmatic rules**

Programmatic rules give the possibility to define more precisely the conditions for accessing data or user services depending on the context.

There are different types of programmatic rules:

- the `AccessRule`API, described in the section below Defining access rules on data [p 282].
- the ServiceActivationRule [p 282], described in the section below Defining activation rules on service [p 282].
- the `ServicePermissionRule`API, described in the section below Defining permission rules on service [p 283].

Defining access rules on data

`AccessRules` are rules that programmatically define, depending on the context, the read/write rights on a data model node or on the records of a table.

The definition of an `AccessRule` is performed as follows:

1. Creation of a rule in the form of a Java class implementing the `AccessRule`API or `AccessRuleForCreate`API interface.
2. Assignment of this rule to concerned nodes in the schema extension: `SchemaExtensions`API.

   According to the rule target (model node(s) or records) and type (`AccessRule` or `AccessRuleForCreate`), several methods such as `SchemaExtensionsContext.setAccessRuleForCreateOnNode`API or `SchemaExtensionsContext.setAccessRuleOnOccurrence`API can be used.

   The rule thus assigned is said to be "local" and is only executed when the target entity is requested. See Resolving permissions on data [p 283] for more information.

Attention

Only one `AccessRule` can be defined for each node, dataspace or record. Only one `AccessRuleForCreate` can be defined for each table child node. The definition of a new programmatic rule of one type will lead to the replacement of the existing one.

Defining activation rules on service

The `ServiceActivationRules` allow to specify if a service is activated or not for a given dataspace or dataset. A service that has been deactivated through this rule is never available in the entity for which it is deactivated, regardless of the current profile, for execution or display, even in permission screens.

The definition of a `ServiceActivationRule` is carried out as follows:

1. Creation of a rule in the form of a Java class implementing the `ServiceActivationRuleForDataspace`API interface or `ServiceActivationRuleForDataset`API, depending on the service type.
2. Assignment of this rule to the impacted services at their declaration level, depending on the service type, via the `ActivationContextOnDataspace.setActivationRule`API or `ActivationContextWithDatasetSet.setActivationRule`API methods.

The resulting assigned rule will be evaluated during the service activation evaluation. See Resolving permissions on services [p 287] for more information.

Defining permission rules on service

The `ServicePermissionRules` are advanced rules allowing to dynamically define the display and execution conditions of a service depending on the context (current session, selected entity, etc.). The service should be activated for the current context beforehand for this type of rule to be triggered.

The definition of a `ServicePermissionRule` is carried out as follows:

1. Creation of a rule in the form of a Java class implementing the `ServicePermissionRule`API interface.
2. Assignment of this rule to the impacted services:
    - Either, for new services, at their declaration level via the `ActivationContext.setPermissionRule`API method.

      The rule thus assigned is said to be "global" and is only executed when the service is activated for the current context. See Resolving permissions on services [p 287] for more information.
    - Or, for existing services, in the **schema extension** `SchemaExtensions`API via the `SchemaExtensionsContext.setServicePermissionRuleOnNode`API and `SchemaExtensionsContext.setServicePermissionRuleOnNodeAndAllDescendants`API methods. It is thus possible to assign a rule to any service, including standard services provided by EBX, on one or more data model nodes: a table node, an association node, etc.

      The rule thus assigned is said to be "local" and is only executed in the extended schema context and when the node corresponds to the one specified. See Resolving permissions on services [p 287] for more information.

      Attention

      Only one `ServicePermissionRule` can be defined for each model node. Thus, the definition of a new programmatic rule will replace the existing one.

49.4 Resolving permissions on data

Resolving user-defined rules

Access rights defined using the user interface are resolved on four levels: dataspace, dataset, record (if applicable) and node.

If a profile is associated with restrictive access rights at a given level, the minimum of all restrictive rights defined at that level is resolved. If no restrictions are defined at that level, the maximum of all access rights defined at that level is resolved.

When a restrictive permission is defined for a profile, it takes precedence over the other permissions potentially granted by the user's other roles. Generally, for all user-defined permission rules that match the current user session:

- If some rules with restrictions are defined, the minimum permissions of these restricted rules are applied.

- If no rules having restrictions are defined, the maximum permissions of all matching rules are applied.

Examples:

Given two profiles *P1* and *P2* concerning the same user, the following table lists the possibilities when resolving that user's permission to a service.

P1 authorization	P2 authorization	Permission resolution
Enabled	Enabled	Enabled. Restrictions do not make any difference.
Disabled	Disabled	Disabled. Restrictions do not make any difference.
Enabled	Disabled	Enabled, unless P2's authorization is a restriction.
Disabled	Enabled	Enabled, unless P1's authorization is a restriction.

The same restriction policy is applied for data access rights resolution.

In another example, a dataspace can be hidden from all users by defining a restrictive association between the built-in profile "Profile.EVERYONE" and the access right "hidden".

At any given level, the most restrictive access rights between those resolved at this level and higher levels are applied. For instance, if a user's dataset access permissions resolve to read-write access, but the container dataspace only allows read access, the user will only have read-only access to this dataset.

Note

The dataset inheritance mechanism applies to both values and access rights. That is, access rights defined on a dataset will be applied to its child datasets. It is possible to override these rights in the child dataset.

Access rights resolution example

In this example, there are three users who belong to the following defined roles and profiles:

User	Profile
User 1	• user1 • role A • role B
User 2	• user2 • role A • role B • role C
User 3	• user3 • role A • role C

The access rights of the profiles on a given element are as follows:

Profile	Access rights	Restriction policy
user1	Hidden	Yes
user3	Read	No
Role A	Read/Write	No
Role B	Read	Yes
Role C	Hidden	No

After resolution based on the role and profile access rights above, the rights that are applied to each user are as follows:

User	Resolved access rights
User 1	Hidden
User 2	Read
User 3	Read/Write

Resolving dataspace and snapshot access rights

At dataspace level, access rights are resolved as follows:

- If a user has several rights defined through multiple profiles:
  - If the rights include restrictions, the minimum of the restrictive profile-rights associations is applied.
  - Otherwise, the maximum of the profile-rights associations is applied.
- If the user has no rights defined:
  - If the user is an administrator or owner of the dataspace, read-write access is given for this dataspace.
  - Otherwise, the dataspace will be hidden.

Resolving dataset access rights

At the dataset level, the same principle applies as at the dataspace level. After resolving the access rights at the dataset level alone, the final access rights are determined by taking the minimum rights between the resolved dataspace rights and the resolved dataset rights. For example, if a dataspace is resolved to be read-only for a user and one of its datasets is resolved to be read-write, the user will only have read-only access to that dataset.

Resolving node access rights

At the node level, the same principle applies as at the dataspace and dataset levels. After resolving the access rights at the node level alone, the final access rights are determined by taking the minimum rights between the resolved dataset rights and the resolved node rights.

Specific access rights can be defined at the node level. If no specific access right is defined, the default access right is used for the resolution process.

> **Note**
>
> The resolution procedure is slightly different for table and table child nodes.

Special case for table and table child nodes

This describes the resolution process used for a given table node or table record *N*.

For each user-defined permission rule that matches one of the user's profiles, the access rights for *N* are either:

1. The locally defined access rights for *N*;
2. Inherited from the access rights defined on the table node;
3. Inherited from the default access rights for dataset values.

All matching user-defined permission rules are used to resolve the access rights for *N*. Resolution is done according to the restriction policy [p 283].

The final resolved access rights will be the minimum between the dataspace, dataset and the resolved access right for *N*.

Resolving programmatic rules

There are three levels of resolution for programmatic access right rules: dataset, record and node. Since only one programmatic access rule can be set for a given level, the last rule set is the one used by the resolution procedure.

Rule resolution on dataset

For a dataset, the last rule set is considered as the resolved rule

Rule resolution on record

For a record, the resolved rule is the minimum between the resolved rule set on the dataset and the rule set on this record.

See `SchemaExtensionsContext.setAccessRuleOnOccurrence`API for more details.

Rule resolution on node

For a node that is a child node of a record, the resolved rule is the minimum between the resolved rule on the record and the rule set on this node.

For a child node of a dataset, the resolved rule is the minimum between the resolved rule set on the dataset and the rule set on this node.

See `SchemaExtensionsContext.setAccessRuleOnNode`API for more details.

Display policy for foreign key drop-down menus

If a record is hidden due to access rules, it will not appear in foreign key drop-down menus.

Attention

The resolved access rights on a dataset or dataset node is the minimum between the resolved access rights defined in the user interface and the resolved programmatic rules, if any.

49.5 Resolving permissions on services

User services give the possibility to execute specific and advanced features from the user interface. Depending on their definition, these services can be called from a menu, as an action in a workflow, as a perspective item, or can be executed directly from a URL as a Web component [p 196].

See also*Overview* *[p 563]*

The permissions of a service are resolved as the service is called from the user interface, namely:

- During the execution, just before the service is displayed.

  If the permission resolved in the user context is not `enabled`, a restriction message is displayed in place of the service.

- During the display of menus if the service is defined as displayable in menus.

  If the permission resolved in the context for the user is not `enabled`, the service will not be displayed in the menu.

Thus, upon every request the resolution of permissions for a service is carried out as follows, in the following order and as long as conditions are respected:

1. The service activation has to correspond to the current context. This activation considers:
   - the selected entity type (dataset, table, record, etc.);
   - static activation rules defined within the `UserServiceDeclaration.defineActivation`API method;
   - the potential dynamic activation rule (ServiceActivationRule [p 282]) also defined within the `UserServiceDeclaration.defineActivation`API method.
2. When the service is activated for the current context, permissions for the user session will be evaluated:
   - If permissions have been defined via the user interface for the current user (or for their roles), their resolution must return `enabled`.

     For more information, please refer to the Resolving user-defined rules [p 288] section.
   - If a global permission rule [p 283] is defined for the service, it must return `enabled` for the context provided (see `ServicePermissionRuleContext`API).
   - If a local permission rule [p 283] is defined for the selected node, it must return `enabled` for the context provided (see `ServicePermissionRuleContext`API).

Resolving user-defined rules

Example

In this example, there are two users belonging to different roles and profiles:

User	Profiles
User 1	• user1 • role A • role B
User 2	• role C • role D

The permissions associated with the roles and profiles defined on the dataset level are as follows:

Profile	Built-in service create (`@creation`)	Built-in service duplicate (`@duplicate`)	Built-in service compare (`@compare`)	Custom service 1 (`custom1`)	Custom service 2 (`custom2`)	Restriction policy
user1	Enabled	Disabled	Enabled	Disabled	Enabled	No
Role A	Enabled	Enabled	Disabled	Enabled	Disabled	Yes
Role B	Enabled	Disabled	Enabled	Enabled	Disabled	Yes
Role C	Enabled	Enabled	Disabled	Disabled	Disabled	No
Role D	Enabled	Disabled	Disabled	Enabled	Disabled	No

The services available to each user after permission resolution are as follows:

Users	Available services
User 1	Built-in service create (`@creation`)
	Custom service 1 (`custom1`)
User 2	Built-in service create (`@creation`)
	Built-in service duplicate (`@duplicate`)
	Custom service 1 (`custom1`)

See also*Resolving user-defined rules* *[p 283]*

49.6 **Resolving permissions on actions**

Actions are low-level operations for EBX object manipulation on which it is possible to define execution rights for a profile. Unlike permissions on user services, which only impact the user interface, these rights are also applicable when an operation is carried out programmatically (i.e. via a `Procedure`API) or indirectly (for example during data import, actions on the table (create, override, occult and delete) are evaluated).

Here is the list of actions on which rights can be defined:

Action object	**Available actions**
Dataspace	Create a child dataspace
	Create a snapshot
	Launch a merge
	Export an archive
	Import an archive
	Close the dataspace
	Close the snapshot
	Create a dataset
Dataset	Duplicate the dataset
	Delete the dataset
	Activate/deactivate the dataset
	Create a view
Table	Create a new record
	Override records
	Occult records
	Delete records

For the resolution of permissions on actions, only the permissions defined via the user interface for the current user (or their roles) will be taken into account, the restriction policy being applied as for any other permission defined via the user interface.

For more information, please refer to the Resolving user-defined rules [p 291] section.

Resolving user-defined rules

Example

In this example, we have two users belonging to different roles and profiles:

User	Profiles
User 1	• user1 • role A • role B
User 2	• role C • role D

Rights associated with roles and profiles on the actions of a given table are as follows:

Profile	Create a record	Override a record	Occult a record	Delete a record	Restriction policy
user1	No	Yes	No	Yes	No
Role A	Yes	No	Yes	No	Yes
Role B	No	Yes	Yes	No	Yes
Role C	Yes	No	No	No	No
Role D	No	No	Yes	No	No

The actions available to each user after resolving the rights are as follows:

Users	Available actions
User 1	Occult a record
User 2	Create a record
	Occult a record

See also*Resolving user-defined rules* *[p 283]*

CHAPTER 50

Criteria editor

This chapter contains the following topics:

50.1 Overview

The criteria editor is included in several different areas of the user interface. It allows defining table filters, as well as validation and computation rules on data. This editor is based on the XPath 1.0 W3C Recommendation.

Two types of criteria exist: atomic criteria and conditional blocks.

See also*Supported XPath syntax* *[p 227]*

50.2 Conditional blocks

Conditional blocks are made up of atomic criteria and other conditional blocks. They express a condition based on the criteria. The following types of blocks exist:

- **No criteria match**: None of the criteria in the block match.
- **Not all criteria match**: At least one criterion in the block does not match.
- **All criteria match**: All criteria in the block match.
- **At least one criterion matches**: One or more of the criteria match.

50.3 **Atomic criteria**

An atomic predicate is defined by a field, an operator, and an expression (either a value or an XPath formula).

Field	Specifies the field of the table to which the criterion applies.
Operator	Specifies the operator used. Available operators depend on the data type of the field.
Value	Specifies the value or expression. See Expression [p 294] below.
Code only	If checked, specifies searching the underlying values for the field instead of labels, which are searched by default.

Expression

The expression can either be a fixed value or a formula. When creating a filter, only fixed values are authorized. During creation of a validation or computation rule, a formula can be created using the wizard.

Known limitation: The formula field does not validate input values, only the syntax and path are checked.

CHAPTER 51

Performance guidelines

This chapter contains the following topics:

51.1 Basic performance checklist

While TIBCO EBX is designed to support large volumes of data, several common factors can lead to poor performance. Addressing the key points discussed in this section will solve the usual performance bottlenecks.

Expensive programmatic extensions

For reference, the table below details the programmatic extensions that can be implemented.

Use case	Programmatic extensions that can be involved
Validation	• **programmatic constraints** `Constraint`API • **computed values** `ValueFunction`API
Table access	• **record-level permission rules** `SchemaExtensionsContext.setAccessRuleOnOccurrence`API • **programmatic filters** `AdaptationFilter`API
EBX content display	• **computed values** `ValueFunction`API • **UI Components** `UIBeanEditor`API • **node-level permission rules** `SchemaExtensionsContext.setAccessRuleOnNode`API
Data update	• **triggers** `Package com.orchestranetworks.schema.trigger`API

For large volumes of data, cumbersome algorithms have a serious impact on performance. For example, a constraint algorithm's complexity is *$O(n^2)$*. If the data size is 100, the resulting cost is proportional to 10 000 (this generally produces an immediate result). However, if the data size is 10 000, the resulting cost will be proportional to 10 000 000.

Another reason for slow performance is calling external resources. Local caching usually solves this type of problem.

If one of the use cases above displays poor performance, it is recommended to track the problem either through code analysis or using a Java profiling tool.

Directory integration

Authentication and permissions management involve the user and roles directory [p 399].

If a specific directory implementation is deployed and accesses an external directory, it can be useful to ensure that local caching is performed. In particular, one of the most frequently called methods is `Directory.isUserInRole`API.

Aggregated lists

In a data model, when an element's cardinality constraint `maxOccurs` is greater than 1 and no `osd:table` is declared on this element, it is implemented as a Java `List`. This type of element is called an aggregated list [p 490], as opposed to a table.

It is important to consider that there is no specific optimization when accessing aggregated lists in terms of iterations, user interface display, etc. Besides performance concerns, aggregated lists are limited with regard to many functionalities that are supported by tables. See tables introduction [p 493] for a list of these features.

Attention

For the reasons stated above, aggregated lists should be used only for small volumes of simple data (one or two dozen records), with no advanced requirements for their identification, lookups, permissions, etc. For larger volumes of data (or more advanced functionalities), it is recommended to use `osd:table` declarations.

51.2 Checklist for dataspace usage

Dataspaces [p 90] available in semantic mode, are an invaluable tool for managing complex data life cycles. While this feature brings great flexibility, it also implies a certain overhead cost, which should be taken into consideration for optimizing usage patterns.

This section reviews the most common performance issues that can appear in case of an intensive use of many dataspaces containing large tables, and how to avoid them.

> **Note**
>
> Sometimes, the use of dataspaces is not strictly needed. As an extreme example, consider the case where every transaction triggers the following actions:
>
> 1. A dataspace is created.
> 2. The transaction modifies some data.
> 3. The dataspace is merged, closed, then deleted.
>
> In this case, no future references to the dataspace are needed, so using it to make isolated data modifications is unnecessary. Thus, using `Procedure`API already provides sufficient isolation to avoid conflicts from concurrent operations. It would then be more efficient to directly do the modifications in the target dataspace, and get rid of the steps which concern branching and merging.
>
> For a developer-friendly analogy, referring to a source-code management tool (CVS, SVN, etc.): when you need to perform a simple modification impacting only a few files, it is probably sufficient to do so directly on the main branch. In fact, it would be neither practical nor sustainable, with regard to file tagging/copying, if every file modification involved branching the whole project, modifying the files, then merging the dedicated branch.

Insufficient memory

When a table is in semantic mode (default), the EBX Java memory cache is used. It ensures a much more efficient access to data when this data is already loaded in the cache. However, if there is not enough space for working data, swaps between the Java heap space and the underlying database can heavily degrade overall performance.

This memory swap overhead can only occur for tables in a dataspace with an on-demand loading strategy [p 299].

Such an issue can be detected by looking at the monitoring log file [p 299]. If it occurs, various actions can be considered:

- reducing the number of child dataspaces that contain large tables;
- reducing the number of indexes specifically defined for large tables;
- using relational mode instead of semantic mode;
- or (obviously) allocating more memory, or optimizing the memory used by applications for non-EBX objects.

See also

Memory management [p 298]

Relational mode [p 245]

Transaction cancels

In semantic mode, when a transaction has performed some updates in the current dataspace and then aborts, loaded indexes of the modified tables are reset. If updates on a large table are often cancelled and, at the same time, this table is intensively accessed, then the work related to index rebuild will slow down the access to the table; moreover, the induced memory allocation and garbage collection can reduce the overall performance.

See also

Functional guard and exceptions *`TableTrigger.guardAndException`*API

*`Procedure`*API

Reorganization of database tables

As with any database, inserting and deleting large volumes of data may lead to fragmented data, which can deteriorate performance over time. To resolve the issue, reorganizing the impacted database tables is necessary. See Monitoring and cleanup of the relational database [p 377].

A specificity of EBX is that creating dataspaces and snapshots adds new entries to tables `HTA` and `ATB`. When poor performance is experienced, it may be necessary to schedule a reorganization of these tables, for large repositories in which many dataspaces are created and deleted.

See also*Monitoring and cleanup of the relational database [p 377]*

51.3 Memory management

Loading strategy

The administrator can specify the loading strategy of a dataspace or snapshot in its information. The default strategy is to load and unload the resources on demand. For resources that are heavily used, a *forced load* strategy is usually recommended.

The following table details the loading modes which are available in semantic mode. Note that the application server must be restarted so as to take into account any loading strategy change.

On-demand loading and unloading	In this default mode, each resource in a dataspace is loaded or built only when it is needed. The resources of the dataspace are "soft"-referenced using the standard Java `SoftReference` class. This implies that each resource can be unloaded "at the discretion of the garbage collector in response to memory demand". The main advantage of this mode is the ability to free memory when needed. As a counterpart, this implies a load/build cost when an accessed resource has not yet been loaded since the server started up, or if it has been unloaded since.
Forced loading	If the forced loading strategy is enabled for a dataspace or snapshot, its resources are loaded asynchronously at server startup. Each resource of the dataspace is maintained in memory until the server is shut down or the dataspace is closed. This mode is particularly recommended for long-living dataspaces and/or those that are used heavily, namely any dataspace that serves as a reference.
Forced loading and prevalidation	This strategy is similar to the forced loading strategy, except that the content of the loaded dataspace or snapshot will also be validated upon server startup.

Monitoring

Indications of EBX load activity are provided by monitoring the underlying database, and also by the 'monitoring' logging category [p 351].

If the numbers for *cleared* and *built* objects remain high for a long time, this is an indication that EBX is swapping.

Tuning memory

The maximum size of the memory allocation pool is usually specified using the Java command-line option `-Xmx`. As is the case for any intensive process, it is important that the size specified by this option does not exceed the available physical RAM, so that the Java process does not swap to disk at the operating-system level.

Tuning the garbage collector can also benefit overall performance. This tuning should be adapted to the use case and specific Java Runtime Environment used.

51.4 **Validation**

The internal incremental validation framework will optimize the work required when updates occur. The incremental validation process behaves as follows:

- The first call to a dataset validation report performs a full validation of the dataset. The loading strategy [p 298] can also specify a dataspace to be prevalidated at server startup.
- Data updates will transparently and asynchronously maintain the validation report, insofar as the updated nodes specify explicit dependencies. For example, standard and static facets, foreign key constraints, dynamics facets, selection nodes specify explicit dependencies.
- If a mass update is executed or if there are too many validation messages, the incremental validation process is stopped. The next call to the validation report will then trigger a full validation.
- If a transaction is cancelled, the validation state of the updated dataset is reset. The next call to the validation report will trigger a full validation as well.

Certain nodes are systematically revalidated, however, even if no updates have occurred since the last validation. These are the nodes with *unknown dependencies*. A node has unknown dependencies if:

- It specifies a **programmatic constraint** `Constraint`API in the default *unknown dependencies* mode,
- It declares a **computed value** `ValueFunction`API, or it declares a dynamic facet that depends on a node that is itself a **computed value** `ValueFunction`API.
- It is an Inherited fields [p 272] or it declares a dynamic facet that depends on a node that is itself an Inherited fields [p 272].

Consequently, on large tables (beyond the order of 10 ^{5}), it is recommended to avoid nodes with unknown dependencies (or at least to minimize the number of such nodes). For programmatic constraints, the developer is able to specify two alternative modes that drastically reduce incremental validation cost: *local dependency* mode and *explicit dependencies*. For more information, see **Dependencies and validation** `DependenciesDefinitionContext.dependencies`API.

> **Note**
>
> It is possible for an administrator user to manually reset the validation report of a dataset. This option is available from the validation report section in EBX.

51.5 **Mass updates**

Mass updates can involve several hundred thousands of insertions, modifications and deletions. These updates are usually infrequent (usually initial data imports), or are performed non-interactively (nightly batches). Thus, performance for these updates is less critical than for frequent or interactive operations. However, similar to classic batch processing, it has certain specific issues.

Batch mode

For relational tables, the implementation of insertions, updates and deletions relies on the `JDBC` batch feature. On large procedures, this can dramatically improve performance by reducing the number of round-trips between the application server and the database engine.

In order to fully exploit this feature, the batch mode can be activated on large procedures. See `ProcedureContext.setBatch`API. This disables the explicit check for existence before record insertions, thus reducing the number of queries to the database, and making the batch processing even more efficient.

Transaction boundaries

It is generally not advised to use a single transaction when the number of atomic updates in the transaction is beyond the order of 10^4. Large transactions require a lot of resources, in particular, memory, from EBX and from the underlying database.

To reduce transaction size, it is possible to:

- Specify the property ebx.manager.import.commit.threshold [p 360]. However, this property is only used for interactive archive imports performed from the EBX user interface.
- Explicitly specify a **commit threshold** `ProcedureContext.setCommitThreshold`API inside the batch procedure.
- Structurally limit the transaction scope by implementing `Procedure`API for a part of the task and executing it as many times as necessary.

On the other hand, specifying a very small transaction size can also hinder performance, due to the persistent tasks that need to be done for each commit.

> **Note**
>
> If intermediate commits are a problem because transactional atomicity is no longer guaranteed, it is recommended to execute the mass update inside a dedicated dataspace. This dataspace will be created just before the mass update. If the update does not complete successfully, the dataspace must be closed, and the update reattempted after correcting the reason for the initial failure. If it succeeds, the dataspace can be safely merged into the original dataspace.

Triggers

If required, triggers can be deactivated using the method `ProcedureContext.setTriggerActivation`API.

51.6 **Accessing tables**

Functionalities

Tables are commonly accessed through EBX and also through the `Request`API API and data services. This access involves a unique set of functions, including a *dynamic resolution* process. This process behaves as follows:

- **Inheritance**: Inheritance in the dataset tree takes into account records and values that are defined in the parent dataset, using a recursive process. Also, in a root dataset, a record can inherit some of its values from the data model default values, defined by the `xs:default` attribute.
- **Value computation**: A node declared as an `osd:function` is always computed on the fly when the value is accessed. See `ValueFunction.getValue`API.
- **Filtering**: An XPath predicate [p 227], a **programmatic filter** `AdaptationFilter`API, or a record-level **permission rule** `SchemaExtensionsContext.setAccessRuleOnOccurrence`API requires a selection of records.
- **Sort**: A sort of the resulting records can be performed.

Accessing tables in semantic mode

Architecture and design

In order to improve the speed of operations on tables, indexes are managed by the EBX engine.

EBX advanced features, such as advanced life-cycle (snapshots and dataspaces), dataset inheritance, and flexible XML Schema modeling, have led to a specialized design for indexing mechanisms. This design can be summarized as follows:

- *Indexes* maintain an in-memory data structure on a whole table.
- An index is not persisted, and building it requires loading all table blocks from the database.

Attention

Faster access to tables is ensured if indexes are ready and maintained in memory cache. As mentioned above, it is important for the Java Virtual Machine to have enough space allocated, so that it does not release indexes too quickly.

Performance considerations

The request optimizer favors the use of indexes when computing a request result.

Attention

- Only XPath filters are taken into account for index optimization.
- Non-primary-key indexes are not taken into account for child datasets.

Assuming the indexes are already built, the impacts on performance are as follows:

1. If the request does not involve filtering, programmatic rules, or sorting, accessing its first few rows (these fetched by a paged view) is almost instantaneous.
2. If the request can be resolved without an extra sort step (this is the case if it has no sort criteria, or if its sort criteria relate to those of the index used for computing the request), accessing the first few rows of a table should be fast. More precisely, it depends on the cost of the specific filtering algorithm that is executed when fetching at least 2000 records.
3. Both cases above guarantee an access time that is independent of the size of the table, and provide a view sorted by the index used. If an extra sort is required, the time taken by the first access depends on the table size according to an `Nlog(N)` function, where `N` is the number of records in the resolved view.

> **Note**
>
> The paginated requests automatically add the primary key to the end of the specified criterion, in order to ensure consistent ordering. Thus, the primary key fields should also be added to the end of any index intended to improve the performance of paginated requests. These include tabular and hierarchical views, and drop-down menus for table references.

If indexes are not yet built, or have been unloaded, additional time is required. The build time is `O(Nlog(N))`.

Accessing the table data blocks is required when the request cannot be computed against a single index (whether for resolving a rule, filter or sort), as well as for building the index. If the table blocks are not present in memory, additional time is needed to fetch them from the database.

It is possible to get information through the monitoring [p 299] and request logging categories.

Other operations on tables

The new records creations or record insertions depend on the primary key index. Thus, a creation becomes almost immediate if this index is already loaded.

REST access to history table

The merge information in history table (the `merge_info` field) has a potentially high access cost. To improve performance and if the client code does not need this field, the `includeMergeInfo` [p 673] parameter must be set to `false`.

See History [p 251] for more information.

Accessing tables in relational mode

When computing a request result, the EBX engine delegates the following to the RDBMS:

- Handling of all request sort criteria, by translating them to an `ORDER BY` clause.
- Whenever possible, handling of the request filters, by translating them to a `WHERE` clause.

Attention

Only XPath filters are taken into account for index optimization. If the request includes non-optimizable filters, table rows will be fetched from the database, then filtered in Java memory by EBX, until the requested page size is reached. This is not as efficient as filtering on the database side (especially regarding I/O).

Information on the transmitted SQL request is logged to the category *persistence*. See Configuring the EBX logs [p 351].

Indexing

In order to improve the speed of operations on tables, indexes may be declared on a table at the data model level. This will trigger the creation of an index of the corresponding table in the database.

When designing an index aimed at improving the performance of a given request, the same rules apply as for traditional database index design.

Setting a fetch size

In order to improve performance, a fetch size should be set according to the expected size of the result of the request on a table. If no fetch size is set, the default value will be used.

- In semantic mode, the default value is 2000.

- In mapped mode, the default value is assigned by the JDBC driver: 10 for Oracle and 0 for PostgreSQL.

Attention

On PostgreSQL, the default value of 0 instructs the JDBC driver to fetch the whole result set at once, which could lead to an `OutOfMemoryError` when retrieving large amounts of data. On the other hand, using fetchSize on PostgreSQL will invalidate server-side cursors at the end of the transaction. If, in the same thread, you first fetch a result set with a fetchsize, then execute a procedure that commits the transaction, then, accessing the next result will raise an exception.

See also

*Request.setFetchSize*API

*RequestResult*API

Administration Guide

CHAPTER 52 Administration overview

The Administration section in TIBCO EBX is the main point of entry for all administration tasks. In this overview are listed all the topics that an administrator needs to master. Click on your topic of interest in order to access the corresponding chapter or paragraph in the documentation.

This chapter contains the following topics:

52.1 Repository management

For storage optimization, it is recommended to maintain a repository (persistence RDBMS) to the necessary minimum. To this end, it is recommended to regularly perform a purge of snapshots and obsolete dataspaces and to consider using a backup file system.

See also Cleaning up dataspaces, snapshots, and history [p 377] and Deleting dataspaces, snapshots, and history [p 378].

It is also possible to archive files of the file system type in order to reduce the storage costs, see EBX monitoring [p 376].

Administration tasks can be scheduled by means of the task scheduler, using built-in tasks, see Task scheduler [p 413].

Object cache

EBX maintains an object cache in memory. The object cache size should be managed on a case by case basis according to specific needs and requirements (pre-load option and pre-validate on the reference dataspaces, points of reference, and monitoring), while continuously monitoring the repository health reports (`./ebxLog/monitoring.log`).

See Memory management [p 298].

Obsolete contents

Keeping obsolete contents in the repository can lead to a slow server startup and slow responsiveness of the interface. It is strongly recommended to delete obsolete content.

For example: datasets referring to deleted data models or undeployed add-on modules. See Deploying and registering TIBCO EBX add-ons [p 369].

Workflow

Cleanup

The workflow history and associated execution data have to be cleaned up on a regular basis.

The workflow history stores information on completed workflows, their respective steps and contexts. This leads to an ever-growing database containing obsolete history and can thus lead to poor performance of the database if not purged periodically. See Workflow history [p 412] for more information.

Email configuration

It is required to configure workflow emails beforehand in order to be able to implement workflow email notifications. See Configuration [p 410] for more information.

52.2 Disk space management

Purge of logs

The log file size will vary according to the log level (and to the selected severity level) and disk space needs to be accordingly managed.

An automatic purge is provided with EBX, allowing to define how many days should log files be stored. After the defined period, log files are deleted.

Any customized management of the purge of logs (backup, archiving, etc.) is the user's responsibility.

```
##################################################
## 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 number 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
```

Audit trail

EBX is provided with a default audit trail manager. Any customized management (including purge, backups, etc.) is the user's responsibility.

If the audit trail is unwanted, it is possible to fully deactivate it. See Activating the XML audit trail [p 350] and Audit trail [p 419] for more information.

52.3 Data model

Publication management

The management of publications of embedded data models [p 85]. See Data model administration [p 403] for more information on the management of these publications and the administration tasks that can be performed (delete, import and export).

Refresh data models

It is possible to update the data models that are using XML Schema documents not managed by EBX. See Data model refresh tool [p 471] for more information.

52.4 Perspectives

EBX offers extensive UI customization options. Simplified interfaces (Recommended perspectives) [p 395] dedicated to each profile accessing the system can be parameterized by the administrator. According to the profile of the user logging in, the interface will offer more or less options and menus. This allows for a streamlined work environment.

See Advanced perspective [p 384] for more information.

52.5 Administrative delegation

EBX is provided with the built-in administrator profile by default. An administrator can delegate administrative rights to a non-administrator user, either for specific actions or for all activities.

The administrative delegation is defined under 'Administration' in the global permissions [p 383] profile.

Access to the administration section can be granted to specific profiles via the global permissions in order to delegate access rights on corresponding administration datasets.

If all necessary administrative rights have been delegated to non-administrator users, it becomes possible to disable the built-in 'Administrator' role.

See also*Configuring the user and roles directory [p 349]*

Installation & configuration

CHAPTER 53

Supported environments

This chapter contains the following topics:

53.1 Browsing environment

Supported web browsers

The TIBCO EBX web interface supports the following browsers:

Microsoft Edge	Minimum supported version is 44 Compatibility mode is not supported.
Microsoft Internet Explorer 10, 11	Compatibility mode is not supported. **Performance limitations:** page loading with IE10 and IE11 is two times slower. This issue is observed when forms have many input components, and particularly many multi-occurrence groups. **Graphical layout:** graphical rendering in IE10 and IE11 can slightly differ from other browsers (for example, the alignment of some labels, icons and other components can be off by a few pixels).
Mozilla Firefox ESR 68 (see details)	As Mozilla Firefox is updated frequently, TIBCO Software Inc. only fully supports version ESR 68. See Mozilla Firefox ESR for more details.
Google Chrome	As Google Chrome is updated frequently and it is not possible to deactivate automatic updates, TIBCO Software Inc. only tests and makes the best effort to support the latest version available.

Screen resolution

The minimum screen resolution for EBX is 1024x768.

Refreshing pages

Browser page refresh is not supported by EBX. When a page refresh is performed, the last user action is re-executed, and therefore could cause issues. It is thus imperative to use the action buttons and links offered by EBX instead of refreshing the page.

'Previous' and 'Next' buttons

The 'previous' and 'next buttons of the browser are not supported by EBX. When navigating through page history, an obsolete user action is re-executed, and therefore could cause issues. It is thus imperative to use the action buttons and links offered by EBX rather than the browser buttons.

Zoom troubleshooting

Zooming in or out may cause some minor display issues (for example extra scrollbar or misalignment). Those issues can be fixed by refreshing the screen using the provided navigation links.

Browser configuration

The following features must be activated in the browser configuration, for the user interface to work properly:

- JavaScript
- Ajax
- Pop-ups

Attention

Avoid using any browser extensions or plug-ins, as they could interfere with the proper functioning of EBX.

53.2 **Supported application servers**

EBX supports the following configurations:

- Java Runtime Environment: JRE 8 or 11, which necessarily includes the limitations specified by the Java Virtual Machine implementation vendor. For example, for JRE and JDK 8, Oracle states that they are "not updated with the latest security patches and are not recommended for use in production". See Oracle Java Archive site.
- Any Java application server that complies with Servlet 3.0 (inclusive) up to 5.0 (exclusive), for example Tomcat 7.0 (inclusive) up to 10.0 (exclusive), WebSphere Application Server 8.5.R5 or higher, WebLogic Application Server 12cR2 or higher, JBoss EAP 6.0 or higher. See Java EE deployment overview [p 325].
- The application server must use UTF-8 encoding for HTTP query strings from EBX. This can be set at the application server level.

For example, on Tomcat, you can set the server to always use the UTF-8 encoding, by setting `URIEncoding` to 'UTF-8' on the `<Connector>` in the `server.xml` configuration file. Alternatively, you can instruct the server to use the encoding of the request body by setting the parameter `useBodyEncodingForURI` to 'true' in `server.xml`.

Attention

- Limitations apply regarding clustering and hot deployment/undeployment:

  Clustering: EBX does not include a cache synchronization mechanism, thus it cannot be deployed into a cluster of active instances. See Technical architecture [p 372] for more information.

  Hot deployment/undeployment: EBX does not support hot deployment/undeployment of web applications registered as EBX modules, or of EBX built-in web applications.

53.3 Supported databases

The EBX repository supports the relational database management systems listed below, with the suitable JDBC drivers. It is important to follow the database vendor recommendations and update policies regarding the database itself, as well as the JDBC driver.

Oracle Database 12c or higher (but excluding 18c).	The distinction of `null` values bears certain limitations. On simple `xs:string` elements, Oracle does not support the distinction between empty strings and `null` values. See Empty string management [p 527] for more information. The user with which EBX connects to the database requires the following privileges: • CREATE SESSION, • CREATE TABLE, • ALTER SESSION, • CREATE SEQUENCE, • A non-null quota on its default tablespace.
PostgreSQL 9.6 or higher.	When using PostgreSQL as the underlying database, a request fetch size must be set, otherwise the JDBC driver will fetch the whole result set at once. This could lead to an `OutOfMemoryError` when retrieving large amounts of data. Also, see this limitation [p 268] regarding the evolution of datamodels in mapped modes. See `Request.setFetchSize`API. The user with which EBX connects to the database needs the CONNECT privilege on the database hosting the EBX repository. Other than this, the default privileges on the public schema of this database are suitable.
Amazon Aurora PostgreSQL 2.3 (compatible with PostgreSQL 10.7) or higher.	The comments in the above section for PostgreSQL apply.
Google Cloud SQL for PostgreSQL 9.6 (compatible with PostgreSQL 9.6.20) or higher.	The comments in the above section for PostgreSQL apply.
SAP HANA Database 2.0 or Higher.	When using SAP HANA Database as the underlying database, certain schema evolutions are not supported. It is, for example, impossible to reduce the length of a column; this is a limitation of HANA, as mentioned in the SQL

	reference guide: "For row table, only increasing the size of VARCHAR and NVARCHAR type column is allowed." The SAP Hana JDBC driver uses the local timezone of the JVM to handle timestamp SQL columns. Hence, for the specific use cases described in the section SQL access to data in relational mode [p 248], the JVM powering the Hana JDBC driver - that is the JVM powering EBX - should be started with the property `user.timezone` set to UTC. This configuration is not free of side effects: for example, the timestamps shown in the EBX logs will be in UTC instead of the local timezone.
Microsoft SQL Server 2012 SP4 or higher.	When used with Microsoft SQL Server, EBX uses the default database collation to compare and sort strings stored in the database. This applies to strings used in the data model definition, as well as data stored in relational and history tables. The default database collation can be specified when the database is created. Otherwise, the collation of the database server is used. To avoid naming conflicts or unexpected behaviors, a case- and accent-sensitive collation must be used as the default database collation (the collation name is suffixed by "CS_AS" or the collation is binary). The default setting to enforce transaction isolation on SQL Server follows a pessimistic model. Rows are locked to prevent any read/write concurrent accesses. This may cause liveliness issues for mapped tables (history or relational). To avoid such issues, it is recommended to activate snapshot isolation on your SQL Server database. The user with which EBX connects to the database requires the following privileges: • CONNECT, SELECT and CREATE TABLE on the database hosting the EBX repository, • ALTER, CONTROL, UPDATE, INSERT, DELETE on its default schema.
Microsoft Azure SQL Database	EBX has been qualified on Microsoft Azure SQL Database v12 (12.00.700), and is regularly tested to verify compatibility with the current version of the Azure database service. When used with Microsoft Azure SQL, EBX uses the default database collation to compare and sort strings stored in the database. This applies to strings used in the data model definition, as well as data stored in relational and history tables. The default database collation can be specified when the database is created. Otherwise, the database engine server collation is used. To avoid naming conflicts or unexpected behaviors, a case- and accent-sensitive collation

	must be used as the default database collation (the collation name is suffixed by "CS_AS" or the collation is binary). The user with which EBX connects to the database requires the following privileges: • CONNECT, SELECT and CREATE TABLE on the database hosting the EBX repository, • ALTER, CONTROL, UPDATE, INSERT, DELETE on its default schema.
H2 v1.3.170 or higher.	H2 is not supported for production environments. The default H2 database settings do not allow consistent reads when records are modified. Relational tables are locked following a pessimistic model. To prevent concurrency issues, it is possible to activate the MVCC feature. Note, however, that the H2 documentation states this feature is not yet fully tested.

For other relational databases, please contact the Support team at https://support.tibco.com.

Attention

In order to guarantee the integrity of the EBX repository, it is strictly forbidden to perform direct modifications to the database (for example, using direct SQL writes), except in the specific use cases described in the section SQL access to data in relational mode [p 248].

See also

Repository administration [p 372]

Data source of the EBX repository [p 323]

Configuring the EBX repository [p 347]

CHAPTER **54**

Java EE deployment

This chapter contains the following topics:

54.1 Introduction

This chapter details deployment specifications for TIBCO EBX on a Java application server. For specific information regarding supported application servers and inherent limitations, see Supported environments. [p 310]

54.2 Software components

EBX uses the following components:

- Library `ebx.jar`
- Embedded [p 318] and required [p 318] third-party Java libraries
- EBX built-in web applications [p 321] and optional custom web applications [p 321]
- EBX main configuration file [p 345]
- EBX repository [p 372]
- Default user and roles directory [p 399], integrated within the EBX repository, or a third-party system (LDAP, RDBMS) for the user authentication

See also*Supported environments [p 310]*

54.3 Embedded third-party libraries

To increase EBX independence and interoperability, it embeds its own third-party libraries. Even if some of them have been modified, preventing conflicts, others must remain unchanged since they are official Java APIs.

The ones that can produce conflicts are:

- Apache Geronimo JSON
- Javax Activation
- Javax Annotations
- Javax JSON Bind
- Javax SAAJ API
- Javax WS RS
- Javax XML Bind

For more information regarding the versions or the details of the Third-Party Library, please refer to the: `TIB_ebx_5.9.20_license.pdf`.

Since those libraries are already integrated, custom web applications should not include them anew, otherwise linkage errors can occur. Furthermore, they should not be deployed aside from the `ebx.jar` library for the same reasons.

54.4 Required third-party libraries

EBX requires several third-party Java libraries. These libraries must be deployed and be accessible from the class-loader of `ebx.jar`. Depending on the application server and the Java runtime environment being used, these libraries may already be present or may need to be added manually.

Database drivers

The EBX repository requires a database. Generally, the required driver is configured along with a data source, if one is used. Depending on the database defined in the main configuration file, one of the following drivers is required. Keep in mind that, whichever database you use, the version of the JDBC client driver must be equal to or higher than the version of the database server.

H2	Version 2.1.210 validated. Note that H2 is not supported in production environments. https://www.h2database.com/ If the repository has been created using H2 version 1.x, this process must be applied, to migrate to the H2 v2 format.
Oracle JDBC	Oracle database 12cR2 is validated on their latest patch set update. Determine the driver that should be used according to the database server version and the Java runtime environment version. Download the `ojdbc8.jar` certified library with JDK 8. Oracle database JDBC drivers download.
SQL Server JDBC	SQL Server 2012 SP4 and greater, with all corrective and maintenance patches applied, are validated. Remember to use an up-to-date JDBC driver, as some difficulties have been encountered with older versions. Include the `mssql-jdbc-8.4.1.jre8.jar` or `mssql-jdbc-8.4.1.jre11.jar` library, depending on the Java runtime environment version you use. Download Microsoft JDBC Driver 8.4.1 for SQL Server (zip).
PostgreSQL	PostgreSQL 9.6 and above validated Include the latest JDBC driver version 4.2 released for your database server and Java runtime environment. PostgreSQL JDBC drivers download.

See also

Data source of the EBX repository [p 323]

Configuring the EBX repository [p 347]

SMTP and emails

The library for JavaMail 1.5.6 email management is required.

The following libraries are used by email features in EBX. See Activating and configuring SMTP and emails [p 354] for details on the configuration.

- `javax.mail.jar`, version 1.5.6, from August 10, 2016
- `smtp.jar`, version 1.5.6, from August 10, 2016
- `pop3.jar`, version 1.5.6, from August 10, 2016

See also*JavaMail*

Secure Socket Layer (SSL)

These libraries are required if your web applications use SSL features.

- `jsse.jar`: https://www.oracle.com/java/technologies/jsse-v103-for-cdc-v102.html
- `ibmjsse.jar`: https://www.ibm.com/developerworks/java/jdk/security/

See also*TIBCO EBX main configuration file* [p 345]

Java Message Service (JMS)

When using JMS, version 1.1 or higher is required.

Depending on whether a Java EE application server or a Servlet/Java Server Pages (JSP) implementation is being used, the library required is as follows:

- For an application server based on Java EE (Java Platform Enterprise Edition), the required JMS provider library is available by default. See http://www.oracle.com/technetwork/java/javaee/overview for more information.
- For a Servlet/Java Server Pages (JSP) implementation using Java SE (Java Platform Standard Edition), for example Apache Tomcat, a JMS provider library such as Apache ActiveMQ may need to be added. See http://www.oracle.com/technetwork/java/javase/overview for more information.

> **Note**
>
> In EBX, the supported JMS model is exclusively Point-to-Point (PTP). PTP systems allow working with queues of messages.

See also*TIBCO EBX main configuration file* [p 345]

XML Catalog API

A library holding the XML Catalog API, introduces by the JAVA SE 9, is required if your web applications are running over a Java Runtime Environment 8 or below, except when a WebLogic 12c R2 application server is used. To ease the installation steps, the following library has been bundled aside from `ebx.jar`, in the *EBX CD.*

- `xml-apis-1.4.01.jar`, version 1.4.01, from August 20, 2011

See Installation notes [p 325] for more information.

54.5 Web applications

EBX provides pre-packaged EARs that can be deployed directly if your company has no custom EBX module web applications to add. If deploying custom web applications as EBX modules, it is recommended to rebuild an EAR containing the custom modules packaged at the same level as the built-in web applications.

Attention

Web application deployment on `/` path context is no more supported. The path context must not be empty nor equals to `/`. Moreover, web applications deployment on paths of different depth is deprecated. Every web application path context must be set on the same path depth.

For more information, see the note on repackaging the EBX EAR [p 326] at the end of this chapter.

EBX built-in web applications

EBX includes the following built-in web applications.

Web application name	Description	Required
`ebx`	EBX entry point, which handles the initialization on start up. See Deployment details [p 322] for more information.	Yes
`ebx-root-1.0`	EBX root web application. Any application that uses EBX requires the root web application to be deployed.	Yes
`ebx-ui`	EBX user interface web application.	Yes
`ebx-manager`	EBX user interface web application.	Yes
`ebx-dma`	EBX data model assistant, which helps with the creation of data models through the user interface. **Note:** The data model assistant requires the `ebx-manager` user interface web application to be deployed.	Yes
`ebx-dataservices`	EBX data services web application. Data services allow external interactions with the EBX repository using the SOAP operations [p 615] and Web Services Description Language WSDL generation [p 607] standards or using the Built-in RESTful services [p 657]. **Note:** The EBX web service generator requires the deployment of the `ebx-manager` user interface web application.	Yes

Custom web applications

It is possible to extend and customize the behavior of EBX by deploying custom web applications which conform to the EBX module requirements.

See also

Packaging TIBCO EBX modules [p 459]

Declaring modules as undeployed [p 361]

54.6 **Deployment details**

Introduction

This section describes the various options available to deploy the '`ebx`' web application. These options are available in its deployment descriptor (`WEB-INF/web.xml`) and are complemented by the properties defined in the main configuration file.

Attention

For JBoss application servers, any unused resources must be removed from the `WEB-INF/web.xml` deployment descriptor.

See also

TIBCO EBX main configuration file *[p 345]*

Supported application servers *[p 311]*

User interface and web access

The web application '`ebx`' (packaged as `ebx.war`) contains the servlet `FrontServlet`, which handles the initialization and serves as the sole user interface entry point for the EBX web tools.

Configuring the deployment descriptor for 'FrontServlet'

In the file `WEB-INF/web.xml` of the web application '`ebx`', the following elements must be configured for `FrontServlet`:

`/web-app/servlet/load-on-startup`	To ensure that `FrontServlet` initializes upon EBX start up, the `web.xml` deployment descriptor must specify the element `<load-on-startup>1</load-on-startup>`.
`/web-app/servlet-mapping/url-pattern`	`FrontServlet` must be mapped to the path '`/`'.

Configuring the application server for 'FrontServlet'

- `FrontServlet` must be authorized to access other contexts, such as `ServletContext`.

  For example, on Tomcat, this configuration is performed using the attribute `crossContext` in the configuration file `server.xml`, as follows:

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

- When several EBX Web Components are to be displayed on the same HTML page, for instance using iFrames, it may be required to disable the management of cookies due to limitations present in some Internet browsers.

  For example, on Tomcat, this configuration is provided by the attribute `cookies` in the configuration file `server.xml`, as follows:

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

Data source of the EBX repository

> **Note**
>
> If the EBX main configuration specifies the property `ebx.persistence.url`, then the environment entry below will be ignored by EBX runtime. This option is only provided for convenience; it is always recommended to use a fully-configurable datasource. See Configuring the EBX repository [p 347] for more information on this property.

The JDBC datasource for EBX is specified in the deployment descriptor `WEB-INF/web.xml` of the '`ebx`' web application as follows:

Reserved resource name	Default JNDI name	Description
`jdbc/EBX_REPOSITORY`	Weblogic: `EBX_REPOSITORY` JBoss: `java:/EBX_REPOSITORY`	JDBC data source for EBX Repository. Java type: `javax.sql.DataSource`

See also

Configuring the EBX repository [p 347]

Rules for the database access and user privileges [p 373]

Mail sessions

> **Note**
>
> If the EBX main configuration does not set `ebx.mail.activate` to 'true', or if it specifies the property `ebx.mail.smtp.host`, then the environment entry below will be ignored by EBX runtime. See SMTP [p 354] in the EBX main configuration properties for more information on these properties.

SMTP and email is declared in the deployment descriptor `WEB-INF/web.xml` of the '`ebx`' web application as follows:

Reserved resource name	Default JNDI name	Description
`mail/EBX_MAIL_SESSION`	Weblogic: `EBX_MAIL_SESSION` JBoss: `java:/EBX_MAIL_SESSION`	Java Mail session used to send emails from EBX. Java type: `javax.mail.Session`

JMS connection factory

> **Note**
>
> If the EBX main configuration does not activate JMS through the property `ebx.jms.activate`, the environment entry below will be ignored by the EBX runtime. See JMS [p 355] in the EBX main configuration properties for more information on this property.

The JMS connection factory is declared in the deployment descriptor `WEB-INF/web.xml` of the '`ebx`' web application as follows:

Reserved resource name	Default JNDI name	Description	Required
`jms/EBX_JMSConnectionFactory`	Weblogic: `EBX_JMSConnectionFactory` JBoss: `java:/EBX_JMSConnectionFactory`	JMS connection factory used by EBX to create connections with the JMS provider configured in the operational environment of the application server. Java type: `javax.jms.ConnectionFactory`	Yes

> **Note**
>
> For deployment on WildFly, JBoss and WebLogic application servers with JNDI capabilities, you must update `EBX.ear` or `EBXForWebLogic.ear` for additional mappings of all required resource names to JNDI names.

JMS for data services

To configure data services to use JMS instead of the default HTTP, you must configure the JMS connection factory [p 324] and the following queues, declared in the `WEB-INF/web.xml` deployment descriptor of the '`ebx`' web application. This is the only method for configuring JMS for data services.

When a SOAP request is received, the SOAP response is optionally returned if the header field `JMSReplyTo` is defined. If so, the fields `JMSCorrelationID` and `JMSType` are retained.

See JMS [p 355] for more information on the associated EBX main configuration properties.

> **Note**
>
> If the EBX main configuration does not activate JMS through the property `ebx.jms.activate`, then the environment entries below will be ignored by EBX runtime. See JMS [p 355] in the EBX main configuration properties for more information on this property.

Reserved resource name	Default JNDI name	Description	Required
`jms/EBX_QueueIn`	Weblogic: `EBX_QueueIn` JBoss: `java:/jms/EBX_QueueIn`	JMS queue for incoming SOAP requests sent to EBX by other applications. Java type: `javax.jms.Queue`	No
`jms/EBX_QueueFailure`	Weblogic: `EBX_QueueFailure` JBoss: `java:/jms/EBX_QueueFailure`	JMS queue for failures. It contains incoming SOAP requests for which an error has occurred. This allows replaying these messages if necessary. Java type: `javax.jms.Queue` **Note:** For this property to be read, the main configuration must also activate the queue for failures through the property `ebx.jms.activate.queueFailure`. See JMS [p 355] in the EBX main configuration properties for more information on these properties.	No

JAR files scanner

To speed up the web applications server startup, the JAR files scanner configuration should be modified to exclude, at least, the `ebx.jar` and `ebx-addons.jar` libraries.

For example, on Tomcat, this should be performed in the `tomcat.util.scan.DefaultJarScanner.jarsToSkip` property from the `catalina.properties` file.

54.7 Installation notes

EBX can be deployed on any Java EE application server that supports Servlet 3.0 up to 5.0 except. The following documentation on Java EE deployment and installation notes are available:

- Installation note for JBoss EAP 7.1.x [p 327]
- Installation note for Tomcat 8.x [p 331]
- Installation note for WebSphere AS 9 [p 335]

- Installation note for WebLogic 12c R2 [p 341]

Attention

- The EBX installation notes on Java EE application servers do not replace the native documentation for each application server.
- These are *not* general installation recommendations, as the installation process is determined by architectural decisions, such as the technical environment, application mutualization, delivery process, and organizational decisions.
- In these examples, no additional EBX modules are deployed. To deploy additional modules, the best practice is to rebuild an EAR with the module as a web application at the same level as the other EBX modules. The web application must declare its class path dependency as specified by the Java™ 2 Platform Enterprise Edition Specification, v1.4:

  J2EE.8.2 Optional Package Support

  (...)

  A JAR format file (such as a JAR file, WAR file, or RAR file) can reference a JAR file by naming the referenced JAR file in a Class-Path header in the Manifest file of the referencing JAR file. The referenced JAR file is named using a URL relative to the URL of the referencing JAR file. The Manifest file is named META-INF/MANIFEST.MF in the JAR file. The Class-Path entry in the Manifest file is of the form:

  `Class-Path: list-of-jar-files-separated-by-spaces`

  In an "industrialized" process, it is strongly recommended to develop a script that automatically builds the EAR, with the custom EBX modules, the EBX web applications, as well as all the required shared libraries.
- In order to avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar` or other libraries in the class-loading system.
- In case of deployment on Oracle WebLogic server, please refer to the Module structure [p 459] section.

CHAPTER **55**

Installation note for JBoss EAP 7.1.x

This chapter contains the following topics:

55.1 Overview

Attention

- This chapter describes a quick installation example of TIBCO EBX on the JBoss Application Server.
- It does not replace the documentation of this application server.
- These are *not* general installation recommendations, as the installation process is determined by architectural decisions such as the technical environment, application mutualization, delivery process, and organizational decisions.
- The complete description of the components required by EBX is given in the following chapter: Java EE deployment [p 317].
- In order to avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar` or other libraries in the class-loading system.

- JBoss Application Server installation
- `EBX_HOME` directory configuration: copy `ebx.properties`
- Java Virtual Machine properties configuration
- JNDI entries configuration
- Data source and JDBC provider creation

- EBX.ear application update
- EBX.ear application deployment
- EBX application start

55.2 **Requirements**

- JBoss Application Server EAP 7.1
- Database and JDBC driver
- EBX CD
- No CDI features in EBX's additional modules (since CDI will be automatically disable)

See also*Supported environments* *[p 310]*

55.3 **Installation**

This quick installation example is performed for a Linux operating system.

1. To download JBoss EAP 7.1, please first download `Installer` jar version 7.1.0 from: https://developers.redhat.com/products/eap/download/
2. Run the `Installer` using *java -jar* command line.
   For further installation details, please refer to the documentation .
3. Perform a standard installation:
   1. Select the language and click 'OK',
   2. Accept the License and click 'Next',
   3. Choose the installation path and click 'Next',
   4. Keep the 'Component Selection' as it is and click 'Next',
   5. Enter 'Admin username', 'Admin password' and click 'Next',
   6. On 'Installation Overview' click 'Next',
   7. On 'Component Installation' click 'Next',
   8. On 'Configure Runtime Environment' leave selection as it is and click 'Next',
   9. When 'Processing finished' appear, click 'Next',
   10. Uncheck 'Create shortcuts in the start menu' and click 'Next',
   11. Generate 'installation script and properties file' at JBoss EAP 7.1 installation root path,
   12. Click on 'done'.

55.4 **Configuration for EBX**

EBX home directory creation and configuration

1. Create the *EBX_HOME* directory, for example `/opt/ebx/home`.

2. Copy from the *EBX CD* the `ebx.software/files/ebx.properties` file to *EBX_HOME*. In our example, we will then have the following text file:

   `/opt/ebx/home/ebx.properties`.

3. Edit the `ebx.properties` file to override the default database if needed. By default, the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported database and it is required to comment the `h2.standalone` one.

Java Virtual Machine properties configuration

1. Open the `standalone.conf` configuration file, placed in `JBOSS_HOME/bin` (or `jboss-eap.conf` file placed in `JBOSS_HOME/bin/init.d` for a running server as a service).
2. Add 'ebx.properties' and 'ebx.home' properties to 'JAVA_OPTS' respectively set with `ebx.properties` file's path and *EBX_HOME* directory's path.

JNDI entries configuration

1. Open the `standalone-full.xml` file placed in `JBOSS_HOME/standalone/configuration`.
2. Add, at least, the following lines to the `server` tag in `messaging-activemq` subsystem:

```
<connection-factory
    name="jms/EBX_JMSConnectionFactory"
    entries="java:/EBX_JMSConnectionFactory"
    connectors="To Be Defined"/>
<jms-queue
    name="jms/EBX_D3ReplyQueue"
    entries="java:/jms/EBX_D3ReplyQueue"
    durable="true"/>
<jms-queue
    name="jms/EBX_QueueIn"
    entries="java:/jms/EBX_QueueIn"
    durable="true"/>
<jms-queue
    name="jms/EBX_QueueFailure"
    entries="java:/jms/EBX_QueueFailure"
    durable="true"/>
<jms-queue
    name="jms/EBX_D3MasterQueue"
    entries="java:/jms/EBX_D3MasterQueue"
    durable="true"/>
<jms-queue
    name="jms/EBX_D3ArchiveQueue"
    entries="java:/jms/EBX_D3ArchiveQueue"
    durable="true"/>
<jms-queue
    name="jms/EBX_D3CommunicationQueue"
    entries="java:/jms/EBX_D3CommunicationQueue"
    durable="true"/>
```

   Warning: the `connectors` attribute value, from the `connection-factory` element, has to be defined. Since the kind of connectors is strongly reliant on the environment infrastructure, a default configuration can not be provided.

   See configuring messaging for more information.

3. Add, at least, the following line to `mail` subsystem:

```
<mail-session name="mail" debug="false" jndi-name="java:/EBX_MAIL_SESSION"/>
```

Data source and JDBC provider creation

1. After the launch of the JBoss Server, run the management CLI without the use of '--connect' or '-c' argument.

2. Use the 'module add' management CLI command to add the new core module. Sample for PostgreSQL configuration:

```
module add \
  --name=org.postgresql \
  --resources=<PATH_TO_JDBC_JAR> \
  --dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api
```

3. Use the 'connect' management CLI command to connect to the running instance.
4. Register the JDBC driver. When running in a managed domain, be sure to precede the command with '/profile=<PROFILE_NAME>'. Sample for PostgreSQL configuration:

```
/subsystem=\
  datasources/jdbc-driver=\
    postgresql:add(\
      driver-name=postgresql,\
      driver-module-name=org.postgresql,\
      driver-xa-datasource-class-name=org.postgresql.xa.PGXADataSource\
    )
```

5. Define the datasource using the 'data-source add' command, specifying the appropriate argument values. Sample for PostgreSQL configuration:

```
data-source add \
  --name=jdbc/EBX_REPOSITORY \
  --jndi-name=java:/EBX_REPOSITORY \
  --driver-name=postgresql \
  --connection-url=jdbc:postgresql://<SERVER_NAME>:<PORT>/<DATABASE_NAME> \
  --user-name=<PERSISTENCE_USER> \
  --password=<PERSISTENCE_PASSWORD>
```

55.5 **Updating EBX Enterprise Application aRchive**

1. Copy from *EBX CD* the `ebx.software/webapps/ear-packaging/EBX.ear` file to your working directory.
2. Uncompress the ear archive to add the application's specific required third-party libraries.

   Mail: see SMTP and emails [p 319] for more information.

   SSL: see Secure Socket Layer (SSL) [p 320] for more information.

   JMS: see Java Message Service (JMS) [p 320] for more information.

   XML Catalog API: see XML Catalog API [p 320] for more information.
3. Compress anew the ear archive.

55.6 **Deploying EBX**

1. Copy EBX.ear into `*JBOSS_HOME*/standalone/deployments` folder.

55.7 **Start EBX**

1. After the launch of the JBoss Server, run the EBX web application: http://localhost:8080/ebx/.
2. At first launch, EBX Wizard [p 367] helps you to configure the default properties of your initial repository.

CHAPTER **56**

Installation note for Tomcat 8.x

This chapter contains the following topics:

56.1 **Overview**

Attention

- This chapter describes a *quick installation example* of TIBCO EBX on Tomcat Application Server.
- It does not replace the documentation of this application server.
- They are *not* general installation recommendations, as the installation process is determined by architectural decisions, such as the technical environment, application mutualization, delivery process, and organizational decisions.
- Tomcat 10.x is not supported.
- The complete description of the components needed by EBX is given in chapter Java EE deployment [p 317].
- In order to avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar` or other libraries in the class-loading system.
- The description below uses the variable name `$CATALINA_HOME` to refer to the directory into which you have installed Tomcat, and from which most relative paths are resolved. However, if you have configured Tomcat for multiple instances by setting a `$CATALINA_BASE` directory, you should use `$CATALINA_BASE` instead of `$CATALINA_HOME` for each of these references.

- Create `EBX_HOME` directory: copy `ebx.properties`
- Copy EBX and third-party libraries: add libraries (*jar files*) to Tomcat lib directory

- Configure JVM arguments (Java system properties): create the `JAVA_OPTS` environment variable
- Deploy EBX application: copy all war files to Tomcat webapps directory

56.2 Requirements

- Java SE 8 or 11
- Apache Tomcat 8.x
- Database and JDBC driver
- EBX CD

See also*Supported environments* *[p 310]*

56.3 Installation

1. To download Tomcat 8.x, choose a core binary distributions from https://tomcat.apache.org/download-80.cgi
2. Run the installer or extract the archive and perform standard installation with default options

56.4 Configuration for EBX

1. Create *EBX_HOME* directory, for example `C:\EBX\home`, or `/home/ebx`
2. Copy from *EBX CD* the `ebx.software\files\ebx.properties` file to *EBX_HOME*. In our example, we will then have the following file:

   `C:\EBX\home\ebx.properties`, or `/home/ebx/ebx.properties`, a text file
3. Edit the `ebx.properties` file to override the default database if needed, by default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and it is required to comment the `h2.standalone` one.
4. Copy third-party library files to `$CATALINA_HOME\lib\` (or `$CATALINA_BASE\lib\`) directory. In our example, we will have:

   `$CATALINA_HOME\lib\mail.jar`

   `$CATALINA_HOME\lib\h2.jar` (default persistence factory)

   `$CATALINA_HOME\lib\xml-apis-1.4.01.jar` (coming from the *EBX CD* directory `ebx.software\lib\lib-xml-apis\`)

   The exact description of these components is given in chapter Components [p 317]. Obviously, if those components are already deployed on the class-loading system, they do not have to be duplicated
5. Modify the `$CATALINA_HOME\conf\server.xml` (or `$CATALINA_BASE\conf\server.xml`) file. Add the following line to the <Host> element

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

   After this modification, we will have:

   ```
   <Host name=...>
   ... ...
   ```

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

`... ...`

`</Host>`

6. Modify the `$CATALINA_HOME\conf\catalina.properties` (or `$CATALINA_BASE\conf\catalina.properties`) file. Add the following lines to the `tomcat.util.scan.DefaultJarScanner.jarsToSkip` property:

   `ebx.jar,\`

   `ebx-addons.jar,\`

7. Configure the launch properties

   If our Tomcat is launched by a command in Windows' Command Prompt or Unix shell, we can create another launcher file:

   For Windows, edit the launcher file `%CATALINA_HOME%\bin\startup.bat`, and add the following command lines:

```
set EBX_HOME="<path_to_the_directory_ebx_home>"
set EBX_OPTS="-Debx.home=%EBX_HOME% -Debx.properties=%EBX_HOME%\ebx.properties"
set JAVA_OPTS="%EBX_OPTS% %JAVA_OPTS%"
```

   or for Linux, edit the launcher file `$CATALINA_HOME/bin/startup.sh`, and add the following command lines:

```
EBX_HOME="<path_to_the_directory_ebx_home>"
EBX_OPTS="-Debx.home=${EBX_HOME} -Debx.properties=${EBX_HOME}/ebx.properties"
export JAVA_OPTS="${EBX_OPTS} ${JAVA_OPTS}"
```

   (!) Accounts used to launch EBX must have create/update/delete rights on *EBX_HOME* directory.

   Windows users that have installed Tomcat as a service may set Java options through the Tomcat service manager GUI (Java tab).

   Be sure to set options on separate lines in the *Java Options* field of the GUI:

```
-Debx.home=<path_to_the_directory_ebx_home>
-Debx.properties=<path_to_the_directory_ebx_home>\ebx.properties
```

   where <path_to_the_directory_ebx_home> is the directory where we copied `ebx.properties`. In our example, it is `C:\EBX\home`, or `/home/ebx`

56.5 **Deploying EBX**

1. Copy from *EBX CD* the `ebx.software\lib\ebx.jar` file to `$CATALINA_HOME\lib\` (or `$CATALINA_BASE\lib\`) directory. In our example, we will have:

   `$CATALINA_HOME\lib\ebx.jar`

2. Copy from *EBX CD* the war files in ebx.software\webapps\wars-packaging to the `$CATALINA_HOME\webapps\` (or `$CATALINA_BASE\webapps\`) directory. In our example, we will have:

   `$CATALINA_HOME\webapps\ebx.war`: Initialization servlet for EBX applications

   `$CATALINA_HOME\webapps\ebx-root-1.0.war`: Provides a common default module for data models

   `$CATALINA_HOME\webapps\ebx-manager.war`: Master Data Management web application

   `$CATALINA_HOME\webapps\ebx-dataservices.war`: Data Services web application

   `$CATALINA_HOME\webapps\ebx-dma.war`: Data Model Assistant web application

`$CATALINA_HOME\webapps\ebx-ui.war`: User Interface web application

56.6 **Start EBX**

1. After Tomcat launch, run EBX web application: http://localhost:8080/ebx/
2. At first launch, EBX Wizard [p 367] helps you to configure the default properties of your initial repository.

CHAPTER **57**

Installation note for WebSphere AS 9

This chapter contains the following topics:

57.1 Overview

Attention

- This chapter describes a quick installation example of TIBCO EBX on the WebSphere Application Server.
- It does not replace the documentation of this application server.
- These are *not* general installation recommendations, as the installation process is determined by architectural decisions such as the technical environment, application mutualization, delivery process, and organizational decisions.
- The complete description of the components required by EBX is given in the following chapter: Java EE deployment [p 317].
- In order to avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar` or other libraries in the class-loading system.

- Install the WebSphere Application Server
- Create the `EBX_HOME` directory: copy `ebx.properties`
- Create a new profile by using the 'WebSphere Customization Toolbox'
- Create a data source and JDBC provider
- Configure the Java Virtual Machine
- Install the EBX.ear application

- Start the EBX application

57.2 Requirements

- WebSphere Application Server 9
- Database and JDBC driver
- EBX CD
- No CDI features in EBX's additional modules (since CDI will be automatically disabled)

See also*Supported environments* [p 310]

57.3 Installation

This quick installation example is performed for a Linux operating system.

1. To download WebSphere AS 9, please first download the latest 'Installation Manager' from http://www-01.ibm.com/support/docview.wss?uid=swg27025142
2. Run the 'Installation Manager' and add the following repositories:
   - WebSphere Application Server V9.0:

     `http://www.ibm.com/software/repositorymanager/V9WASBase`
   - WebSphere Application Server Network Deployment V9.0:

     `http://www.ibm.com/software/repositorymanager/V9WASND`
3. Install the 'WebSphere Application Server Network Deployment'

   For further installation details, please refer to the documentation.
4. Run the 'WebSphere Customization Toolbox' and perform a standard installation with default options:
   1. Create profile: click 'Create' then select 'Application Server', and click 'Next'
   2. Profile Creation Options: select 'Advanced profile creation' and click 'Next'
   3. Optional Application Deployment: select those options:
      - Deploy the 'Administrative Console'
      - Deploy the 'Installation Verification Tool' application

      Then click 'Next'
   4. Profile Name and Location: enter a profile name (example: 'EbxAppSrvProfile') and directory

      `/opt/IBM/WebSphere/AppServer/profiles/EbxAppSrvProfile`

      further correspond to `PROFILE_HOME` and click 'Next'
   5. Node and Host Names: enter the node name (example: 'Node1'), the server name (example: 'EbxServer'), the host name (example: 'localhost'), and then click 'Next'
   6. Administrative Security: check 'Enable administrative security' option, enter the user name, the password, and click 'Next'
   7. Security Certificate (part 1): select 'Create a new default personnal certificate' and 'Create a new root signing certificate', and click 'Next'

8. Security Certificate (part 2): keep as default and click 'Next'
9. Port Value Assignment: keep as default and click 'Next'
10. Linux Service Definition: check 'Run the application server process as a Linux service' option, enter the user name (example: 'ebx'), and click 'Next'
11. Web Server Definition: keep as default and click 'Next'
12. Profile Creation Summary: keep as default and click 'Create'
13. Profile Creation Complete: uncheck 'Launch the First steps console' option, and click 'Finish'

57.4 Configuration for EBX

1. Create the *EBX_HOME* directory, for example `/opt/ebx/home`
2. Copy from the *EBX CD* the `ebx.software\files\ebx.properties` file to *EBX_HOME*. In our example, we will then have the following text file:

   `/opt/ebx/home/ebx.properties`.
3. Edit the `ebx.properties` file to override the default database if needed. By default, the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported database and it is required to comment the `h2.standalone` one
4. Create the `EBX_LIB` directory, for example `/opt/ebx/lib`
5. Copy third-party library files from the *EBX CD*, or from other sources, to the `<EBX_LIB>` directory. In our example, for a PostgreSQL database, we will have:

   `postgresql-X.X.X-driver.jar` (coming from another source than the *EBX CD*).

   `xml-apis-1.4.01.jar` (coming from the *EBX CD* directory `ebx.software/lib/lib-xml-apis/`).

   The exact description of these components is given in the chapter Components [p 317]. If those components are already deployed on the class-loading system, they do not have to be duplicated (ex: javax.mail-1.5.6.jar is already present on the WebSphere Application Server, and the database driver is configured at the data source level).
6. Start the server (for example 'EbxServer'),

   sudo <PROFILE_HOME>/bin/startServer.sh <serverName>

   `cd /opt/IBM/WebSphere/AppServer/profiles/EbxAppSrvProfile`

   `sudo bin/startServer.sh EbxServer`.
7. Connect into the 'WebSphere Integrated Solutions Console' using the user name and password typed during the profile creation (Administrative Security step), enter the following url in the browser:

   https://localhost:9043/ibm/console
8. Create a data source and JDBC provider
   1. On the left menu, go to 'Resources > JDBC > Data Sources', to configure your database access. Choose the jdbc 'Scope' (for example use 'Cell'), and click 'New'
   2. Enter basic data source information:
      - Data source name: `EBX_REPOSITORY`
      - JNDI name: `jdbc/EBX_REPOSITORY`

   click on 'Next'

   3. Select the JDBC provider: select 'Create new JDBC provider', and click 'Next'
   4. Create a new JDBC provider: (example with a PostgreSQL database)
      - Database type: `User-defined`
      - Implementation class name: `org.postgresql.ds.PGConnectionPoolDataSource`
      - Name: `PostgreSQL`

      and click 'Next'
   5. Enter database class path information: (example with a PostgreSQL database)
      - Class path: `/opt/ebx/lib/postgresql-X.X.X-driver.jar`

      and click 'Next'
   6. Enter database specific properties for the data source: keep as default and click 'Next'
   7. Setup security aliases: keep as default and click 'Next'
   8. Summary: click 'Finish'
   9. Save the master configuration

9. Configure the data source: `jdbc/EBX_REPOSITORY`
   1. Click on 'Data Sources > `EBX_REPOSITORY`'
   2. On the right in the 'Configure additional properties' section, click on 'Additional Properties' and define the database account access:
      - Define `user` value to the according user
      - Define `password` value to the according password
   3. Save the master configuration
   4. Test the connection
10. Configure the Java Virtual Machine
    1. Click on 'Application Servers'
    2. Click on the server name (for example: 'EbxServer')
    3. Click on 'Process definition' under 'Server infrastructure > Java Process Management'
    4. Click on 'Java Virtual Machine' under 'Additional Properties'
    5. Enter in 'Generic JVM arguments' the value:

       ```
       -Debx.properties=/opt/ebx/home/ebx.properties -Debx.home=/opt/ebx/home
       ```
    6. Enter in 'Classpath' the paths to the third-party library files placed in the `<EBX_LIB>` directory except for the JDBC driver
    7. click 'Ok'
    8. Save the master configuration

57.5 **Deploying EBX**

1. Copy from the *EBX CD* the `ebx.software/webapps/ear-packaging/EBX.ear` to the *EBX_HOME* directory. In our example, we will have:

`/opt/ebx/ear/EBX.ear`

2. Connect into the 'WebSphere Integrated Solutions Console' using the user name and password typed during the profile creation (Administrative Security step), enter the following url in the browser:

   https://localhost:9043/ibm/console

3. Click on 'WebSphere enterprise applications' under 'Applications > Application Types'
4. Install the EBX application
   1. New Application: On the right panel, click on 'New Enterprise Application'
   2. Preparing for the application installation: Browse your EBX.ear file, located under `/opt/ebx/ear/EBX.ear`, then click 'Next'
   3. How do you want to install the application?: Select 'Fast Path...', then click 'Next'
   4. Select installation options: keep as default, then click 'Next'
   5. Map modules to servers: select all modules, then click 'Next'
   6. Map resource references to resources: copy the 'Resource Reference' value and paste it in the 'Target Resource JNDI Name' field, for all modules, then click 'Next'
   7. Warnings will appear related to `JNDI:mail/EBX_MAIL_SESSION` and `JNDI:jms/EBX_JMSConnectorFactory`. This behavior is normal since we had not configured these resources. Click 'Continue'
   8. Map resource environment references to resources: Copy the 'Resource Reference' value and paste it to the 'Target Resource JNDI Name' value, for all modules, then click 'Next'
   9. Warnings will appear related to non-available resources. This behavior is normal since we had not configured these resources, then click 'Continue'
   10. Map virtual hosts for Web modules: select all modules and click 'Next'
   11. Summary: keep as default, click 'Finish'
   12. If installation succeeds, it logs 'Application EBX installed successfully', then click 'Save'
5. On the left menu, go to 'Applications > Enterprise Applications'
6. Change EBX application's class loader policy
   1. Click on EBX resource’s name
   2. On the 'configuration' pane, under 'Detail Properties', select 'Class loading and update detection'
   3. Under 'General Properties', change 'Class loader order' to 'Classes loaded with local class loader first (parent last)'
   4. Return to 'Applications > Enterprise Applications'
7. Enterprise Applications: select EBX, and then click 'Start'

   The EBX 'Application status' will be changed into a green arrow

57.6 **Start EBX**

1. After the launch of the WebSphere Server, run the EBX web application: http://localhost:9080/ebx/

2. At first launch, EBX Wizard [p 367] helps you to configure the default properties of your initial repository

CHAPTER **58**

Installation note for WebLogic 12c R2

This chapter contains the following topics:

58.1 Overview

Attention

- This chapter describes a quick installation example of TIBCO EBX on the WebLogic Server.
- It does not replace the documentation of this application server.
- They are *not* general installation recommendations, as the installation process is determined by architectural decisions, such as the technical environment, application mutualization, delivery process, and organizational decisions.
- The complete description of the components needed by EBX is given in the following chapter: Java EE deployment [p 317].
- In order to avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar` or other libraries in the class-loading system.

- Install the Java Virtual Machine: Create the JAVA_HOME environment variable
- Install the WebLogic Server
- Create the `EBX_HOME` directory: copy `ebx.properties`
- Create a new domain by using the 'Configuration Wizard'
- Configure the domain: Declare EBX `JAVA_OPTIONS` and copy the database JDBC driver
- Install and configure the JDBC driver
- Deploy the EBX application: install dedicated ear file

58.2 Requirements

- Java SE 8 or 11
- WebLogic Server 12c R2
- Database and JDBC driver
- EBX CD

See also*Supported environments* *[p 310]*

58.3 Installation

1. To download WebLogic 12c R2, please refer to https://edelivery.oracle.com/osdc/faces/Home.jspx
2. Run the 'Fusion Middleware Configuration Wizard' (`<weblogic_home>/oracle_common/common/bin/config.sh`) and perform a standard installation with default options:
    1. Create Domain: choose 'Create a new domain' and specify the domain home folder, then click 'Next'
    2. Templates: keep as default and click 'Next'
    3. Administrator Account: enter a domain administrator username and password and click 'Next'
    4. Domain Mode and JDK: choose the production mode and your jdk installation home and click 'Next'
    5. Advanced configuration: check 'Administration server' and 'Topology'. That way, we create two independent domain nodes: an administration one and an application one, and click 'Next'
    6. Administration Server: enter your administration node name (for example 'AdminServer') and listen port (by default `7001`), then click 'Next'
    7. Managed Servers: add the application node name (for example 'EbxServer') and listen port (for example `7003`), then click 'Next'
    8. Clusters: keep as default and click 'Next'
    9. Machines: keep as default and click 'Next'
    10. Virtual Targets: keep as default and click 'Next'
    11. Partitions: keep as default and click 'Next'
    12. Configuration Summary: click 'Create'
    13. Configuration Process: click 'Next'
    14. End Of Configuration: click 'Finish'

58.4 Configuration for EBX

1. Create the *EBX_HOME* directory, for example `C:\EBX\home`, or `/home/ebx`
2. Copy from the *EBX CD* the `ebx.software\files\ebx.properties` file to *EBX_HOME*. In our example, we will then have the following file:

`C:\EBX\home\ebx.properties`, or `/home/ebx/ebx.properties`, a text file

3. Edit the `ebx.properties` file to override the default database if needed, by default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and it is required to comment the `h2.standalone` one.
4. Configure the launch properties for the *Managed Server* (for example 'EbxServer')

   Edit the `<DOMAIN_HOME>/bin/startManagedWebLogic.sh` script file by adding the following lines:

   ```
   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}"
   ```
5. Copy third-party library files to the `<DOMAIN_HOME>/lib` directory. In our example, for an H2 standalone data base, we will have:

   `h2.jar` (default persistence factory)

   The exact description of these components is given in chapter Components [p 317]. Obviously, if those components are already deployed on the class-loading system, they do not have to be duplicated (ex: mail.jar and xml-apis-1.4.01.jar are already present in the WebLogic Server).
6. Start the 'Administration server' (for example 'AdminServer'), `<DOMAIN_HOME>/bin/startWebLogic.sh`
7. Launch the 'WebLogic Server Administration Console', enter the following url in the browser:

   http://localhost:7001/console.

   Log in with the domain administrator username and password
8. Click on 'Services > Data sources' in the 'Domain Structure' panel, then click on the 'New > Generic Data Source' button
   1. Type Name: `EBX_REPOSITORY`, JNDI Name: `EBX_REPOSITORY` Database Type: *Your database type* and click 'Next'
   2. Choose your database driver type, and click 'Next'
   3. Uncheck 'Supports Global Transactions', and click 'Next'
   4. Setup your database 'Connection Properties' and click 'Next'
   5. Click 'Test Configuration' and then 'Finish'
   6. Switch on the 'Targets' tab and select all Servers, then click 'Save'
   7. Restart the Administration server (for example 'AdminServer')

      `<DOMAIN_HOME>/bin/stopWebLogic.sh`

      `<DOMAIN_HOME>/bin/startWebLogic.sh`

58.5 **Deploying EBX**

1. Copy from the *EBX CD* the `ebx.software/webapps/ear-packaging/EBXForWebLogic.ear` to the *EBX_HOME* directory. In our example, we will have:

   `C:\EBX\home\EBXForWebLogic.ear`, or `/home/ebx/EBXForWebLogic.ear`
2. Launch the 'WebLogic Server Administration Console', enter the following url in the browser:

   http://localhost:7001/console
3. Click on 'Lock and Edit' in the 'Change Center' panel

4. Click on 'Deployments' in the 'Domain Structure' panel, and click 'Install'
    1. Install Application Assistant: Enter in 'Path' the application full path to EBXForWebLogic.ear file, located on `C:\EBX\home\`, or `/home/ebx/` folder and click 'Next'
    2. Choose the installation type and scope: Click on 'Install this deployment as an application' and 'Global' default scope and click 'Next'
    3. Select the deployment targets: Select a node name (for example 'EbxServer') from the 'Servers' list and click 'Next'
    4. Optional Settings: keep as default and click 'Finish'
5. Click on 'Activate Changes', on the top left corner. The deployment status will change to 'prepared'
6. Switch to 'Control' tab, select the 'EBXForWebLogic' enterprise application, then click on 'Start' > 'Servicing all requests'
7. Start the application node name (for example 'EbxServer'),

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

58.6 **Start EBX**

1. After WebLogic Server launch, run the EBX web application: http://localhost:7003/ebx/
2. At first launch, EBX Wizard [p 367] helps you to configure the default properties of your initial repository

CHAPTER **59**

TIBCO EBX main configuration file

This chapter contains the following topics:

59.1 Overview

The EBX main configuration file, by default named `ebx.properties`, contains most of the basic parameters for running EBX. It is a Java properties file that uses the standard simple line-oriented format.

The main configuration file complements the Java EE deployment descriptor [p 322]. Administrators can also perform further configuration through the user interface, which is then stored in the EBX repository.

See also

Deployment details [p 322]

UI administration [p 383]

Location of the file

The access path to the main configuration file can be specified in several ways. In order of descending priority:

1. By defining the Java system property 'ebx.properties'. For example, this property can be set by adding the option `-Debx.properties=<filePath>` to the `java` command-line command. See Java documentation.
2. By defining the servlet initialization parameter 'ebx.properties'.

   This standard Java EE setting must be specified in the `web.xml` file of the web application '`ebx`'. EBX accesses this parameter by calling the method `ServletConfig.getInitParameter("ebx.properties")` in the servlet `FrontServlet`.

   See getInitParameter in the Oracle `ServletConfig` documentation.
3. By default, if nothing is specified, the main configuration file is located at `WEB-INF/ebx.properties` of the web application '`ebx`'.

> **Note**
>
> In addition to specifying properties in the main configuration file, it is also possible to set the values of properties directly in the system properties. For example, using the `-D` argument of the `java` command-line command.

Custom properties and variable substitution

The value of any property can include one or more variables that use the syntax `${propertyKey}`, where *`propertyKey`* is either a system property, or a property defined in the main configuration file.

For example, the default configuration file provided with EBX uses the custom property `ebx.home` to set a default common directory, which is then included in other properties.

59.2 Setting an EBX license key

See also*Initialization and first-launch assistant* [p 367]

The license key can be retrieved from the TIBCO eDelivery site.

```
##################################################
## EBX® License number
## (as specified by your license agreement)
##################################################
ebx.license=paste_here_your_license_key
```

59.3 Setting automatic installation on first launch

Repository can be automatically installed on first startup.

```
##################################################################
## Installation on first launch.
## All values are ignored if the repository is  already installed.
##################################################################
```

```
## Enables repository installation on first startup (default is false).
## If true, property ebx.license should also be set to a valid license.
ebx.install.enabled=true

## Following properties configure the repository. Values are optional and defaults are automatically generated.
ebx.install.repository.id=00275930BB88
ebx.install.repository.label=A Test

## Following properties specify the EBX 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 none encrypted password used for the EBX 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 EBX 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 password_to_encrypt
ebx.install.admin.password.encrypted=8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
```

59.4 **Setting the EBX root directory**

The EBX root directory contains archives, the XML audit trail and, when the repository is persisted on H2 standalone mode, the H2 database files.

```
##################################################
## Path for EBX® XML repository
##################################################
ebx.repository.directory=${ebx.home}/ebxRepository
```

59.5 **Configuring the EBX repository**

Before configuring the persistence properties of the EBX repository, carefully read the section Technical architecture [p 372] in the chapter 'Repository administration'.

The required library (driver) for each supported database is described in the chapter Database drivers [p 318].

See also

Repository administration [p 372]

Rules for the database access and user privileges [p 373]

Supported databases [p 313]

Data source of the EBX repository [p 323]

Database drivers [p 318]

```
#################################################################
## 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 only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url=jdbc:h2:tcp://127.0.0.1/ebxdb
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


################################################################
## Case EBX® persistence system is Oracle database.
################################################################
#ebx.persistence.factory=oracle

## Specific properties to be set only only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url=jdbc:oracle:thin:@127.0.0.1:1521:ebxDatabase
#ebx.persistence.driver=oracle.jdbc.OracleDriver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


## Activate to use VARCHAR2 instead of NVARCHAR2 on Oracle; never modify on an existing repository.
#ebx.persistence.oracle.useVARCHAR2=false

################################################################
## Case EBX® persistence system is SAP Hana
################################################################
#ebx.persistence.factory=hana

## Specific properties to be set only 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=yyyyyyyy


################################################################
## Case EBX® persistence system is Microsoft SQL Server.
################################################################
#ebx.persistence.factory=sqlserver

## Specific properties to be set only only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url= \
#jdbc:sqlserver://127.0.0.1:1036;databasename=ebxDatabase
#ebx.persistence.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


################################################################
## Case EBX® persistence system is Microsoft Azure SQL database.
################################################################
#ebx.persistence.factory=azure.sql

## Specific properties to be set only only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url= \
#jdbc:sqlserver://myhost.database.windows.net:1433;database=ebxDatabase;encrypt=true;\
#trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;
#ebx.persistence.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


################################################################
## Case EBX® persistence system is PostgreSQL.
################################################################
#ebx.persistence.factory=postgresql

## Specific properties to be set only only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
```

```
#ebx.persistence.url=jdbc:postgresql://127.0.0.1:5432/ebxDatabase
#ebx.persistence.driver=org.postgresql.Driver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy
```

59.6 Configuring the user and roles directory

This parameter specifies the Java directory factory class name. It must only be defined if not using the default EBX directory.

See also

Users and roles directory [p 399]

`DirectoryFactory` API

```
#################################################
## 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
```

It is also possible to disable the built-in role "ADMINISTRATOR".

```
#################################################
## Specifies whether the built-in role ADMINISTRATOR is disabled.
## Default value is false.
#################################################
#ebx.directory.disableBuiltInAdministrator=true
```

59.7 Configuring EBX localization

This parameter is used to configure the locales used at runtime. This list must contain all the locales that are exposed to the end-user. EBX will not be able to display labels and messages in a language that is not declared in this list.

The default locale must be the first one in the list.

```
##################################################################
## Available locales, separated by a comma.
## The first element in the list is considered as the default locale.
## If not set, available locales are 'en-US, fr-FR'.
##
##################################################################
#ebx.locales.available=en-US, fr-FR
```

See also*Extending TIBCO EBX internationalization [p 239]*

59.8 Setting temporary files directories

Temporary files are stored as follows:

```
#################################################
## Directories for temporary resources.
#################################################
# The property ebx.temp.directory allows to specify a directory for temporary files.
# Default value is java.io.tmpdir
#
ebx.temp.directory = \\${java.io.tmpdir}
#ebx.temp.directory = /tmp/java

# 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
```

59.9 Activating the XML audit trail

By default, the XML audit trail is activated. It can be deactivated using the following variable:

```
##################################################################
# The XML history has been replaced by an SQL history.
# This old XML history can be deactivated using the following variable.
# Default is true.
##################################################################
ebx.history.xmlaudittrail.activated = true
```

See also[*Audit trail*] *[p 419]*

59.10 Configuring the EBX logs

The most important logging categories are:

ebx.log4j.category.log.kernel	Logs for EBX main features, processes, exceptions and compilation results of modules and data models.
ebx.log4j.category.log.workflow	Logs for main features, warnings and exceptions about workflow.
ebx.log4j.category.log.persistence	Logs related to communication with the underlying database.
ebx.log4j.category.log.setup	Logs for the compilation results of all EBX objects, except for modules and data models.
ebx.log4j.category.log.validation	Logs for datasets validation results.
ebx.log4j.category.log.mail	Logs for the activity related to the emails sent by the server (see Activating and configuring SMTP and emails [p 354]). **Note:** This category must not use the Custom SMTP appender [p 352] in order to prevent infinite loops.
ebx.log4j.category.log.d3	Logs for D3 events on EBX.
ebx.log4j.category.log.dataservices	Logs for data service events in EBX.
ebx.log4j.category.log.monitoring	Raw logs for monitoring [p 299].
ebx.log4j.category.log.request	The optimization strategy for every `Request`API issued on a semantic table in the EBX repository.
ebx.log4j.category.log.restServices	Logs for REST services events in EBX, including those from the REST Toolkit [p 719].

Some of these categories can also be written to through custom code using the `LoggingCategory`API interface.

```
 ##################################################
## Log4J properties:
##
## We have some specific syntax extensions:
## - Appender ebxFile:<aFileName>
## Defines a file appender with default settings (threshold=DEBUG)
##
```

```
## - property log.defaultConversionPattern is set by Java
##################################################
#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
```

Custom 'ebxFile' appender

The token `ebxFile:` can be used as a shortcut to define a daily rolling file appender with default settings. It must be followed by a file name. It then activates an appender that writes to a file located in the directory `ebx.logs.directory`.

The property `ebx.log4j.appender.ebxFile.backup.Threshold` allows defining the maximum number of backup files for daily rollover.

```
##################################################
## Directory of log files 'ebxFile:'
## This property is used by special appender prefixed
## by 'ebxFile:' (see log section below)
##################################################
ebx.logs.directory=${ebx.home}/ebxLog

#################################################################
# Daily rollover threshold of log files 'ebxFile:'
# Specifies the maximum number of backup files for daily rollover of 'ebxFile:' appenders.
# When set to a negative value, backup log files are never purged.
# Default value is -1.
#################################################################
ebx.log4j.appender.ebxFile.backup.Threshold=-1
```

Custom SMTP appender

The appender `com.onwbp.org.apache.log4j.net.SMTPAppender` provides an asynchronous email sender.

See also*Activating and configuring SMTP and emails* [p 354]

Module's appenders configuration

The configurations of module logging categories are declared according to a hierarchical pattern. Furthermore, every enabled logging request for a given category will be forwarded to all the appenders defined for that category, as well as to the appenders higher in the hierarchy. This fact means that appenders are inherited additively.

The root module logging category can be customized by setting the property `ebx.log4j.category.log.wbp`. For a given module, if the root module logging category configuration is the only one applicable, then a `DailyRollingFileAppender` is automatically added to the module's appenders list. The log file name will be derived from the module's name and from an optional specific sub-category.

```
#####################################################################
## Root module logging category configuration
#####################################################################
ebx.log4j.category.log.wbp = FATAL, Console
```

Every module logging category can be customized by setting the property `ebx.log4j.category.log.module.xxxxxx`, where `xxxxxx` corresponds to the module's name. The explicit configuration of the module logging category will disable the automatic addition of the `DailyRollingFileAppender` previously described. However, any other defined appenders will be inherited additively.

Since the inheritance mechanism may not be suitable for every case, the additivity can be broken by setting to `false` the property `ebx.log4j.additivity.log.wbp.xxxxxx`, where `xxxxxx` corresponds to the module's name.

```
###############################################################################
## Module logging category configuration
##
## The module's log messages will only be written to mycompany-module log file
## since additivity has been broken
###############################################################################
ebx.log4j.category.log.module.mycompany-module = INFO, ebxFile:mycompany-module

ebx.log4j.additivity.log.wbp.mycompany-module = false
```

Module's log threshold

The inheritance mechanism described in Module's appenders configuration [p 353] is applied to the module log threshold as well. Actually, the inherited level for a given logger, is equal to the first non-null level in the hierarchy, starting from this logger and proceeding up to the root module logging category.

Custom module log threshold

There is an exception to the inheritance, for custom module, since the log level threshold of their logging category, by default, is set to `INFO`. This threshold can be customized by setting the property `ebx.log4j.category.log.module.xxxxxx`, where `xxxxxx` corresponds to the custom module's name.

Example: `ebx.log4j.category.log.module.mycompany-module=DEBUG`.

See also *`ModuleContextOnRepositoryStartup.getLoggingCategory`* API

Add-on module log threshold

Like custom module, by default, the log level threshold of any add-on module is set to `INFO`.

The log level threshold can be customized by setting the property `ebx.log4j.category.log.addon.xxxxxx` where `xxxxxx` corresponds to the add-on module's name.

Example: `ebx.log4j.category.log.addon.daqa=DEBUG`

Modules log threshold overrides summary

Thus and considering EBX logging features, the log threshold defined at the root level in the logger hierarchy may be overridden by (by order from least to most weighted):

- the module logging category explicit configuration,
- the module's `log.threshold.xxxx` or `log.threshold.category.xxxx` log configuration properties, where `xxxx` corresponds to the sub category,
- the following log configuration properties: `ebx.log4j.category.log.module.xxxxxx`, where `xxxxxx` corresponds to the custom module's name, or `ebx.log4j.category.log.addon.xxxxxx`, where `xxxxxx` corresponds to the add-on module's name.

59.11 Activating and configuring SMTP and emails

The internal mail manager sends emails asynchronously. It is used by the workflow engine and the custom SMTP appender `com.onwbp.org.apache.log4j.net.SMTPAppender`.

See also*Mail sessions* *[p 323]*

```
##################################################
## 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 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=
## 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
```

59.12 Configuring data services

```
##################################################################
## Data services
##################################################################

# Specifies the default value of the data services parameter
# 'disableRedirectionToLastBroadcast'.
# Default is false.
#ebx.dataservices.disableRedirectionToLastBroadcast.default=false

# Specifies the default value for deletion at the end of close and
# merge operations.
# If the parameter is set in the request operation, it overrides
# this default setting.
# If unspecified, default is false.
#ebx.dataservices.dataDeletionOnCloseOrMerge.default=false
#ebx.dataservices.historyDeletionOnCloseOrMerge.default=false

# 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, it tries authentication 'Basic Authentication Scheme'
# method and set 'Basic' value in 'WWW-Authenticate' header of HTTP
# response.
# 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
```

59.13 Activating and configuring JMS

See also *JMS for data services* *[p 324]*

```
##################################################################
## 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
```

59.14 Configuring distributed data delivery (D3)

See Configuring D3 nodes [p 443] for the main configuration file properties pertaining to D3.

See also

JMS for distributed data delivery (D3) *[p 433]*

Introduction to D3 *[p 424]*

59.15 **Configuring REST toolkit services**

```
#################################################################
## REST configuration
#################################################################

# Defines the maximum number of bytes that will be extracted
# from the REST request body to build some DEBUG log messages.
# Default value is 8192 bytes.
# This value is ignored if DEBUG level is not activated on the restServices logger.
#ebx.restservices.log.body.content.extract.size=8192
```

59.16 **Configuring Web access from end-user browsers**

HTTP Authorization header policy

EBX natively offers three policies to send and receive credentials using HTTP headers:

standard	It corresponds to the authentication scheme, using the HTTP Authorization header, described in the RFC 2617.
ebx	To prevent HTTP Authorization header override issues, this policy acts the same as the `standard` but the credentials are stored in an EBX specific HTTP header.
both	It is the combination of the two previously described policies.

```
#################################################
## EBX® authorization header policy for HTTP requests
##
## Possible values are: standard, ebx, both.
##  standard:
##    the standard HTTP Authorization header holds the credentials
##  ebx:
##    an EBX® specific HTTP header holds the credentials
##  both:
##    both (standard and specific) HTTP headers hold the credentials
##
## Default value is: both.
#################################################
#ebx.http.authorization.header.policy=both
```

URLs computing

By default, EBX runs in "standalone" mode, where external resources (images, JavaScript, etc.) are provided by the application server.

Also by default, URL-related parameters in the main configuration file do not have to be set.

In this case, the server name and the port are obtained from the initial request sent to EBX.

See also*URL policy (deprecated)* *[p 386]*

```
#################################################################
## 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. Default value is false.
##
## 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=
```

Proxy mode

Proxy mode allows using a front-end HTTP server to provide static resources (images, CSS, JavaScript, etc.). This architecture reduces the load on the application server for static HTTP requests. This configuration also allows using SSL security on the front-end server.

The web server sends requests to the application server according to a path in the URL. The `servletAlias` and `uiServletAlias` paths are specified in the main configuration file.

The web server provides all external resources. These resources are stored in a dedicated directory, accessible using the `resourcesAlias` path.

EBX must also be able to access external resources from the file system. To do so, the property `ebx.webapps.directory.externalResources` must be specified.

To force the use of SSL security even if the incoming requests do not, `ebx.servlet.http.useHttpsSettings` and / or `ebx.externalResources.http.useHttpsSettings` properties must be set to `true`. Their default values are `false`.

The main configuration file may be configured as follows:

```
##################################################
## Path for external resources if they are not
## delivered within web applications
## This field is mandatory if in proxy mode.
##################################################
ebx.webapps.directory.externalResources=D:/http/resourcesFolder

##################################################

ebx.servlet.useLocalUrl=true

#ebx.servlet.http.host=
#ebx.servlet.http.port=
ebx.servlet.http.path=servletAlias
ebx.servlet.http.ui.path=uiServletAlias
#ebx.servlet.http.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
```

Attention

When proxy mode is used, the URL to the `ebx-dataservices` module must be configured through the lineage administration panel. Note that the provided URL must end its path with `/ebx-dataservices`.

See also*Path recommendations*

Reverse-proxy mode

If URLs generated by EBX, for requests and external resources, must contain a different protocol than the one from the incoming request, a specific server name, a specific port number or a specific path prefix, properties may be configured as follows:

```
##################################################
#ebx.servlet.useLocalUrl=false

ebx.servlet.http.host=reverseDomain
#ebx.servlet.http.port=
ebx.servlet.http.path=ebx/
#ebx.servlet.http.ui.path=ebx-ui/
#ebx.servlet.http.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=
```

Attention

When reverse-proxy mode is used, the URL to the `ebx-dataservices` module must be configured through the lineage administration panel. Note that the provided URL must end its path with `/ebx-dataservices`.

See also*Path recommendations*

Path recommendations

It is recommended to deploy webapps of EBX modules on the same path level. For example:

- `/module1`
- `/module2`

or:

- `/path/module1`
- `/path/module2`

It avoids difficulties when configuring the advanced features of Proxy mode and Reverse-proxy mode.

Consequently, it is not recommended to use the root path `"/"` as the servlet's HTTP or HTTPS path in the following properties:

- `ebx.servlet.http.path`
- `ebx.servlet.http.ui.path`
- `ebx.servlet.https.path`
- `ebx.servlet.https.ui.path`

59.17 Configuring failover

These parameters are used to configure the failover mode and activation key, as well as heartbeat logging in `DEBUG` mode.

See also*Failover with hot-standby* *[p 373]*

```
##################################################
## 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
```

59.18 Tuning the EBX repository

Some options can be set so as to optimize memory usage.

The properties are configured as follows:

```
################################################################
## Technical parameters for memory and performance tuning
################################################################
# Import commit threshold allows to specify the commit threshold
# exclusively for the archive import launched directly from Manager.
#
# For more details about the commit threshold,
# see the JavaDoc ProcedureContext.setCommitThreshold().
# Default value is 0.
#
ebx.manager.import.commit.threshold=100

# A validation messages threshold allows specifying the maximum number of
# messages to consider per constraint when performing a validation.
# This threshold is considered for each constraint defined in a data model
# and for each severity in each dataset validation report.
# When the threshold is reached by a constraint and a severity:
# - the validation of the constraint is stopped
# - an error message indicating that the threshold has been reached
# is added to the validation report.
# When set to 0 or a negative value, the threshold is not considered.
# Default value is 0.
#
ebx.validation.constraints.messages.threshold = 100


# Specifies whether the validation report should be kept in memory,
# regardless of the loading strategy of the dataspace.
# Default value is true. However, it is recommended to deactivate it
# when the repository contains a large number of open dataspaces and
# # datasets.
#ebx.validation.report.keepInMemory=false
```

See also *Validation report page* *[p 365]*

59.19 Miscellaneous

Activating data workflows

This parameter specifies whether data workflows are activated. This parameter is not taken into account on the fly. The server must be restarted whenever the value changes.

```
##################################################
## Workflow activation.
## Default is false.
##################################################
ebx.workflow.activation = true
```

Disabling user task legacy mode

This parameter specifies whether the creation service of a user task in legacy mode should be offered in the workflow modeling. The default value is `true`.

See `UserTask.UserTaskMode.LEGACY_MODE`API for more information.

```
## Disables legacy work item mode(default is true)
## Specify if the creation service of user task in legacy mode must be offered
## in workflow modeling.
#ebx.manager.workflow.legacy.userTaskMode=false
```

Log procedure starts

This parameter specifies whether starts of the procedure execution are logged.

```
#################################################
## Specifies whether transaction starts are logged. Default is false.
#################################################
ebx.logs.logTransactionStart = true
```

Log validation starts

This parameter specifies whether starts of datasets validation are logged.

```
#################################################
## Specifies whether validation starts are logged. Default is false.
#################################################
ebx.logs.logValidationStart = true
```

Deployment site identification

This parameter allows specifying the email address to which technical log emails are sent.

```
#################################################
## Unique Site Name
## --> used by monitoring emails and by the repository
#################################################
ebx.site.name= name@domain.com
```

Dynamically reloading the main configuration

Some parameters can be dynamically reloaded, without restarting EBX. The parameter `thisfile.checks.intervalInSeconds` indicates how frequently the main configuration file is checked.

```
#################################################
### Checks if this file has been updated
### If value <= 0, no more checks will be done
#################################################
thisfile.checks.intervalInSeconds=1
```

In development mode, this parameter can be set to as low as one second. On production systems, where changes are expected to be less frequent, the value can be greater, or set to '0' to disable hot reloading entirely.

This property is not always supported when the module is deployed as a WAR, as it would then depend on the application server.

Declaring modules as undeployed

On application server startup, the initialization of deployed web applications / EBX modules and the initialization of the EBX repository are performed asynchronously. In order to properly initialize the EBX repository, it is necessary to compile all the data models used by at least a dataset, hence EBX will wait endlessly for referenced modules to be registered.

If a module is referenced by a data model but is not deployed (or no longer deployed), it is necessary to declare this module as undeployed to unlock the wait and continue the startup process.

> **Note**
>
> The `kernel` logging category indicates which modules are awaited.

> **Note**
>
> A module declared as undeployed cannot be registered into EBX until it is removed from the property `ebx.module.undeployedModules`.

> **Note**
>
> Any data model based on an unregistered module will have an "undeployed module" compilation error.

See also

Module registration *[p 460]*

Dynamically reloading the main configuration *[p 361]*

```
##################################################
## Comma-separated list of EBX® modules declared
## as undeployed.
## If a module is expected by the EBX® repository but is
## not deployed, it must be declared in this property.
## Caution:
## if the "thisfile.checks.intervalInSeconds" property is deactivated,
## a restart is mandatory, otherwise it will be hot-reloaded.
##################################################
ebx.module.undeployedModules=
```

Module public path prefix

EBX modules' public paths are declared in the 'module.xml' file of each module. A context prefix can be declared for all modules, without having to modify the 'module.xml' content, by specifying the property that follows.

This prefix will apply to any EBX module, including core, add-on and specific modules.

When proxy and / or reverse-proxy mode are used, the `ebx.servlet.http[s].path` and `ebx.servlet.http[s].ui.path` properties must take into account this module public path prefix setting. Conversely, the `ebx.externalResources.http[s].path` property must end its path just before a potential prefix.

```
##################################################
ebx.servlet.useLocalUrl=true

#ebx.servlet.http.host=
#ebx.servlet.http.port=
ebx.servlet.http.path=reverse-proxy/prefix/ebx/
ebx.servlet.http.ui.path=reverse-proxy/prefix/ebx-ui/
#ebx.servlet.http.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/
```

See URLs computing [p 356] for more information.

EBX run mode

This property defines how EBX runs. Three run modes are available: *development*,*integration* and *production*.

When running in *development* mode, the development tools [p 471] are activated in EBX, some features thus become fully accessible and more technical information is displayed.

> **Note**
>
> The administrator can always access this information regardless of the mode used.

The additional features accessible when running in *development* mode include the following (non-exhaustive list):

Documentation pane	In the case of a computed value, the Java class name is displayed. A button is displayed giving access to the path to a node.
Compilation information	Module and schema compilation information is displayed in the dataset validation report.
Java bindings	The generation of Java bindings is available if the schema of the dataset mentions at least one binding.
Web component link generator	The Web component link generator is available on datasets and dataspaces.
Data model assistant	Data model configuration and additional options, such as Services, Business Objects and Rules, Java Bindings, Toolbars and some advanced properties.
Workflow modeling	Declare specific script tasks.
Log	The logs include additional technical information intended for the developer. For example, a warning is written to logs if a drop-down list is defined on a node which is not an enumeration in a UI Bean.
Product documentation	The product documentation is always the most complete one (i.e "advanced"), including administration and development chapters.

```
##################################################
## Server Mode
## Value must be one of: development, integration, production
## Default is production.
##################################################
backend.mode=integration
```

Note

There is no difference between the *integration* and *production* modes.

Resource filtering

This property allows the filtering of certain files and directories in the resource directory contents (resource type node, with an associated facet that indicates the directory that contains usable resources).

```
##################################################
## list (separated by comma) of regexps excluding resource
## the regexp must be of type "m:[pattern]:[options]".
## the list can be void
##################################################
```

```
ebx.resource.exclude=m:CVS/*:
```

Validation report page

The validation report page can display a finite number of items for each severity. This number can be tuned with this property.

```
# Defines the maximum item displayed for each severity in the validation report page.
# Default value is 100.
#ebx.validation.report.maxItemDisplayed=100
```

See also *Tuning the EBX repository* [p 360]

Validation report logs

This property allows to specify the number of validation messages to display in the logs when validating a dataset or a table.

```
# Defines the maximum number of messages displayed in the logs.
# Default value is 100.
# When set to 0 or a negative value, the limit is not considered.
#ebx.validation.report.maxItemDisplayedInLogs=500
```

See also *Tuning the EBX repository* [p 360]

CHAPTER **60**

Initialization and first-launch assistant

Deliverables can be found on TIBCO eDelivery(an account is mandatory in order to access eDelivery, please contact the support team to request one).

The TIBCO EBX Configuration Assistant helps with the initial configuration of the EBX repository. If EBX does not have a repository installed upon startup and if the automatic installation [p 346] is not enabled, the configuration assistant is launched automatically.

Before starting the configuration of the repository, make sure that EBX is correctly deployed on the application server. See Java EE deployment [p 317].

> **Note**
>
> The EBX main configuration file must also be properly configured. See TIBCO EBX main configuration file [p 345].

This chapter contains the following topics:

1. License key
2. Configuration steps

60.1 License key

When launching EBX, the license key page displays automatically if no valid license key is available, that is, if there is no license key entered in the EBX main configuration file, or if the current license key has expired.

The license key can be retrieved from the TIBCO eDelivery site (an account is mandatory in order to access eDelivery, please contact https://support.tibco.com to request one).

The license key is available in the `TIB_ebx_{ebx.version.public}_license_key.pdf` file.

60.2 Configuration steps

The EBX configuration assistant guides you through the following steps:

1. Validating the license agreement.
2. Configuring the repository.

3. Defining users in the default user and roles directory (if a custom directory is not defined).
4. Validating the information entered.
5. Installing the EBX repository.

Validating the license agreement

In order to proceed with the configuration, you must read and accept the product license agreement.

Configuring the repository

This page displays some of the properties defined in the EBX main configuration file. You also define several basic properties of the repository in this step.

Id of the repository (`repositoryId`)	Must uniquely identify the repository (in the scope of the enterprise). The identifier is 48 bits (6 bytes) long and is usually represented as 12 hexadecimal digits. This information is used for generating the Universally Unique Identifiers (UUIDs) of entities created in the repository, and also of transactions logged in the history. This identifier acts as the "UUID node", as specified by RFC 4122.
Repository label	Defines a user-friendly label that indicates the purpose and context of the repository.

See also*TIBCO EBX main configuration file* *[p 345]*

Defining users in the default directory

If a custom user and roles directory is not defined in the EBX main configuration file, the configuration assistant allows to define default users for the default user and roles directory.

An administrator user must be defined. You may optionally create a second user.

See also*Users and roles directory* *[p 399]*

Validating the information entered

Before proceeding with the installation of the repository, you can review the configuration of the repository and the information entered on the 'Configuration Summary' page. If you need to modify information, you can return to the previous pages using the configuration assistant **< Back** button.

Once you have verified the configuration, click the button **Install the repository >** to proceed with the installation.

Installing the EBX repository

The repository installation is performed using the provided information. When the installation is complete, you are redirected to the repository login page.

CHAPTER **61**

Deploying and registering TIBCO EBX add-ons

> **Note**
>
> Refer to the documentation of each add-on for additional installation and configuration information in conjunction with this documentation.

This chapter contains the following topics:

61.1 Deploying an add-on module

> **Note**
>
> **Each add-on bundle version is intended to run with a specific EBX version and all its fix releases. Make sure that the EBX and add-on bundle versions are compatible, otherwise the add-on registration will abort.**

The web application deployment descriptor for the add-on module must specify that class definitions and resources from the web application are to be loaded in preference to classes from the parent and server classloaders.

For example, on WebSphere Application Server, this can be done by setting `<context-priority-classloader>true</context-priority-classloader>` in the `web-app` element of the deployment descriptor.

On WebLogic, include `<prefer-web-inf-classes>True</prefer-web-inf-classes>` in `weblogic.xml`.

See the documentation on class loading of your application server for more information.

The EBX add-on common JAR file, named `lib/ebx-addons.jar`, must be copied in the library directory shared by all web applications.

> **Note**
>
> The add-on log level can be managed in the main configuration file [p 353].

61.2 Registering an add-on module

To register a new EBX add-on in the repository:

1. Navigate to the 'Administration' area.
2. Click the down-arrow in the navigation pane and select **Technical configuration** > **Add-ons registration**.
3. On the **Registered add-ons** page, click the **+** button to create a new entry.
4. Select the add-on you are registering, and enter its license key.

   > **Note**
   >
   > If the EBX repository is under a trial license, no license key is required for the add-on. The add-on will be subject to the same trial period as the EBX repository itself.

   The license key can be retrieved from the TIBCO eDelivery site.
5. Click on **Save**.

61.3 Deleting an add-on module

To delete an add-on module from the EBX repository:

1. Navigate to the 'Administration' area.
2. Click the down-arrow in the navigation pane and select **Technical configuration** > **Add-ons registration**.
3. On the **Registered add-ons** page, tick the box corresponding to the add-on to be deleted.
4. In the 'Actions' menu, select 'Delete'.
5. Close and purge the Administration datasets related to the previously used add-on, as well as the including dataspaces.

   When an add-on is no longer deployed, a dataspace corresponding to the Administration dataset will then appear in the list of Reference children under the dataspaces. When an add-on module is no longer deployed, it is thus necessary to close/delete and purge manually all data/dataspaces related to the add-on.

Technical administration

CHAPTER 62

Repository administration

This chapter contains the following topics:

62.1 Technical architecture

Overview

The main principles of the TIBCO EBX technical architecture are the following:

- A Java process (JVM) that runs EBX is limited to a single EBX repository. This repository is physically persisted in a supported relational database instance [p 313], accessed through a configured data source [p 347].
- A repository cannot be shared by multiple JVMs at any given time. However, a failover architecture may be used. These aspects are detailed in the sections Single JVM per repository [p 373] and Failover with hot-standby [p 373]. Furthermore, to achieve horizontal scalability, an alternative is to deploy a distributed data delivery (D3) [p 424] environment.
- A single relational database instance can support multiple EBX repositories (used by distinct JVMs). It is then required that they specify distinct table prefixes using the property `ebx.persistence.table.prefix`.

See also

Configuring the EBX repository [p 347]

Supported databases [p 313]

Data source of the EBX repository [p 323]

Rules for the database access and user privileges

Attention

In order to guarantee the integrity of persisted master data, **it is strictly forbidden to perform direct SQL writes to the database**, except for specific use cases described in the section SQL access to data in relational mode [p 248].

It is required for the database user specified by the configured data source [p 347] to have the 'create/alter' privileges on tables, indexes and sequences. This allows for automatic repository installation and upgrades [p 375].

See also

SQL access to history [p 254]

Accessing a replica table using SQL [p 261]

SQL access to data in relational mode [p 248]

Data source of the EBX repository [p 323]

Single JVM per repository

A repository cannot be shared by multiple JVMs. If such a situation was to occur, it would lead to unpredictable behavior and potentially even corruption of data in the repository.

EBX performs checks to enforce this restriction. Before the repository becomes available, the repository must first acquire exclusive ownership of the relational database. After starting the repository, the JVM periodically checks that it still holds ownership of the repository.

These checks are performed by repeatedly tagging a technical table in the relational database. The shutdown command for the application server ensures that the tag on this technical table is removed. If the server shuts down unexpectedly, the tag may be left in the table. If this occurs, the server must wait several additional seconds upon restart to ensure that the table is not being updated by another live process.

Attention

To avoid an additional wait period at the next start up, it is recommended to always properly shut down the application server.

Failover with hot-standby

The exclusion mechanism described above is compatible with failover architectures, where only one server is active at any given time in an active/passive cluster. To ensure that this is the case, the main server must declare the property `ebx.repository.ownership.mode=failovermain`. The main server claims ownership of the repository database, as in the case of a single server.

A backup server can still start up, but it will not have access to the repository. It must declare the property `ebx.repository.ownership.mode=failoverstandby` to act as the backup server. Once started, the backup server is registered in the connection log. Its status can be retrieved using the Java API or through an HTTP request, as described in the section Repository status information and logs [p 374] below.

In order to activate the backup server and transfer exclusive ownership of the repository to it, a specific request must be issued by an HTTP request, or using the Java API:

- Using HTTP, the request must include the parameter `activationKeyFromStandbyMode`, and the value of this parameter must be equal to the value declared for the entry `ebx.repository.ownership.activationkey` in the EBX main configuration file. See Configuring failover [p 359].

  The format of the request URL must be:

  ```
  http[s]://<host>[:<port>]/ebx?activationKeyFromStandbyMode={value}
  ```
- Using the Java API, call the method `RepositoryStatus.wakeFromStandby`API.

If the main server is still up and accessing the database, the following applies: the backup server marks the ownership table in the database, requesting a clean shutdown for the main server (yet allowing any running transactions to finish). Only after the main server has returned ownership can the backup server start using the repository.

Repository status information and logs

A log of all attempted Java process connections to the repository is available in the Administration area under 'History and logs [p 251]' > 'Repository connection log'.

The status of the repository may be retrieved using the methods in the `RepositoryStatus`API API.

It is also possible to get the repository status information using an HTTP request that includes the parameter **`repositoryInformationRequest`** with one of following values:

`state`	The state of the repository in terms of ownership registration. • `D`: Java process is stopped. • `O`: Java process has exclusive ownership of the database. • `S`: Java process is started in failover standby mode, but is not yet allowed to interact with the repository. • `N`: Java process has tried to take ownership of the database but failed because another process is holding it.
`heart_beat_count`	The number of times that the repository has made contact since associating with the database.
`info`	Detailed information for the end-user regarding the repository's registration status. The format of this information may be subject to modifications in the future without explicit warning.

62.2 Auto-increments

Several technical tables can be accessed in the 'Administration' area of the EBX user interface. These tables are for internal use only and their content should not be edited manually, unless removing obsolete or erroneous data. Among these technical tables are:

Auto-increments	Lists all auto-increment fields in the repository.

62.3 Repository management

Installation and upgrades

Automatic installation and upgrades

By complying with the Rules for the database access and user privileges [p 373], the repository installation or upgrade is done automatically.

Inter-database migration

EBX provides a way to export the full content of a repository to another database. The export includes all dataspaces, configuration datasets, and mapped tables. To operate this migration, the following guidelines must be respected:

- The source repository **must be shut down**: no EBX server process must be accessing it; **not strictly complying with this requirement can lead to a corrupted target repository**;
- A new EBX server process must be launched on the target repository, which must be empty. In addition to the classic Java system property `-Debx.properties`, this process must also specify `ebx.migration.source.properties`: the location of an EBX properties file specifying the source repository. (It is allowed to provide distinct table prefixes between target and source.)
- The migration process will then take place automatically. Please note, however, that this process is not transactional: should it fail halfway, it will be necessary to delete the created objects in the target database, before starting over.
- After the migration is complete, an exception will be thrown, to force restarting the EBX server process accessing the target repository.

Limitations:

- For technical reasons, migration to an Oracle database is only supported from another Oracle database.
- The names of the database objects representing the mapped tables (history, replication, relational) may have to be altered when migrated to the target database, to comply with the limitations of its database engine (maximum length, reserved words, ...). Such alterations will be logged during the migration process.
- As a consequence, the names specified for replicated tables in the data model will not be consistent with the adapted name in the database. The first recompilation of this data model will force to correct this inconsistency.

- Due to different representations of numeric types, values for `xs:decimal` types might get rounded if the target database engine offers a lesser precision than the source. For example, a value of `10000000.1234567890123456789` in Oracle will get rounded to `10000000.123456789012345679` in SQL Server.

Repository backup

A global backup of the EBX repository must be delegated to the underlying RDBMS. The database administrator must use the standard backup procedures of the underlying database.

Archives directory

Archives are stored in a sub-directory called `archives` within the `ebx.repository.directory` (see configuration [p 345]). This directory is automatically created during the first export from EBX.

Attention

If manually creating this directory, make sure that the EBX process has read-write access to it. Furthermore, the administrator is responsible for cleaning this directory, as EBX does not maintain it.

> **Note**
>
> The transfer of files between two EBX environments must be performed using tools such as FTP or simple file copies by network sharing.

Repository attributes

A repository has the following attributes:

repositoryId	Uniquely identifies a repository within the scope of the company. It is 48 bits (6 bytes) and is usually represented as 12 hexadecimal digits. This information is used for generating UUIDs (Universally Unique Identifiers) for entities created in the repository, as well as transactions logged in history tables or in the XML audit trail. This identifier acts as the 'UUID node' part, as specified by *RFC 4122*.
repository label	Provides a user-friendly label that identifies the purpose and context of the repository. For example: "Production environment".
store format	Identifies the underlying persistence system, including the current version of its structure.

62.4 Monitoring management

Monitoring and cleanup of the relational database

Some entities accumulate during the execution of EBX.

Attention

It is the *administrator's responsibility* to monitor and clean up these entities.

Monitoring and reorganization

The persistence data source of the repository must be monitored through RDBMS monitoring.

The EBX tables specified in the default semantic mode [p 245] have their content persisted in a set of generic database tables:

- The table `${ebx.persistence.table.prefix}HOM`, in which each record represents a dataspace or a snapshot (its name is `EBX_HOM` if the property `ebx.persistence.table.prefix` is unset).
- The table `${ebx.persistence.table.prefix}BLK`, where the data of EBX tables in semantic mode are segmented into blocks of at most 100 EBX records (its name is `EBX_BLK` if the property `ebx.persistence.table.prefix` is unset).
- The tables ${ebx.persistence.table.prefix}HTA, ${ebx.persistence.table.prefix}TAR and ${ebx.persistence.table.prefix}ATB, defining which blocks belong to a given EBX table in a given dataspace or snapshot.

Database statistics

The performance of requests executed by EBX requires that the database has computed up-to-date statistics on its tables. Since database engines regularly schedule statistics updates, this is usually not an issue. Yet, it could be necessary to explicitly update the statistics in cases where tables are heavily modified over a short period of time (e.g. by an import creating many records).

Impact on UI

Some UI components use statistics to adapt their behavior in order to prevent users from executing costly requests unwillingly.

For example, the combo box will not automatically search on user input if the table contains a large volume of records. This behavior may also occur if the database's statistics are not up to date, because a table may be considered as containing a large volume of records even if it is not actually the case.

Cleaning up dataspaces, snapshots, and history

A full cleanup of dataspaces, snapshots, and history from the repository involves several stages:

1. Closing unused dataspaces and snapshots to keep the cache to a minimal size.
2. Deleting dataspaces, snapshots, and history.
3. Purging the remaining entities associated with the deleted dataspaces, snapshots, and history from the repository.

Closing unused dataspaces and snapshots

In order to keep the cache and the repository to a reasonable size, it is recommended to close any dataspaces and snapshots that are no longer required. This can be done in the following ways:

- Through the user interface, in the 'Dataspaces' area.
- From the 'Dataspaces / Snapshots' table under 'Dataspaces' in the 'Administration' area, using the **Actions** menu in the workspace. The action can be used on a filtered view of the table.
- Through the Java API, using the method `Repository.closeHome`API.
- Using the data service "close dataspace" and "close snapshot" operations. See Closing a dataspace or snapshot [p 641] for more information.

Once the dataspaces and snapshots have been closed, the data can be safely removed from the repository.

> **Note**
>
> Closed dataspaces and snapshots can be reopened in the 'Administration' area, under 'Dataspaces'.

Deleting dataspaces, snapshots, and history

Dataspaces, associated history and snapshots can be permanently deleted from the repository. However, the deletion of a dataspace does not necessarily imply the deletion of its history. The two operations are independent and can be performed at different times.

> **Note**
>
> The deletion of a dataspace, a snapshot, or of the history associated with them is recursive. The deletion operation will be performed on every descendant of the selected dataspace.

After the deletion of a dataspace or snapshot, some entities will remain until a repository-wide purge of obsolete data is performed. In particular, the complete history of a dataspace remains visible until a repository-wide purge is performed. Both steps, the deletion and the repository-wide purge, must be completed in order to totally remove the data and history. The process has been divided into two steps for performance issues. As the total clean-up of the repository can be time-intensive, this allows the purge execution to be initiated during off-peak periods on the server.

The process of deleting the history of a dataspace takes into account all history transactions recorded up until the deletion is submitted or until a date specified by the user. Any subsequent historized operations will not be included when the purge operation is executed. To delete new transactions, the history of the dataspace must be deleted again.

> **Note**
>
> It is not possible to set a deletion date in the future. The specified date will thus be ignored and the current date will be used instead.

The deletion of dataspaces, snapshots, and history can be performed in a number of different ways:

- From the 'Dataspaces/Snapshots' table under 'Dataspaces' in the 'Administration' area, using the **Actions** menu button in the workspace. The action can be used on a filtered view of the table.

- Using the Java API, and more specifically the methods `Repository.deleteHome`API and `RepositoryPurge.markHomeForHistoryPurge`API.
- At the end of the data service "close dataspace" operation, using the parameters `deleteDataOnClose` and `deleteHistoryOnClose`, or at the end of a "merge dataspace" operation, using the parameters `deleteDataOnMerge` and `deleteHistoryOnMerge`.

Purging remaining entities after a dataspace, snapshot, or history deletion

Once items have been deleted, a purge can be executed to clean up remaining data from *all* deletions performed until that point. A purge can be initiated in the following ways:

- Through the user interface, by selecting in the 'Administration' area **Actions > Execute purge** in the navigation pane.
- Using the Java API, specifically the method `RepositoryPurge.purgeAll`API.
- Using the task scheduler. See Task scheduler [p 413] for more information.

The purge process is logged in the directory `${ebx.repository.directory}/db.purge/`.

Cleaning up tables having unreadable records

Some data model evolutions can lead to unreadable data:

- A column containing null values is added to the primary key of a table.
- The type of a column has changed to a different type with no possible conversion.

In these situations, records that do not match the new table definition are no longer visible, but remain persisted in the table. (This allows retrieving records if switching back to the previous table definition). When such records are encountered, an informative error is recorded in EBX logs.

EBX provides the option to clean the records that no longer conform to the model, once the new version of the data model is stabilized. This allows recovering space in the database and getting rid of error messages. Please proceed carefully, as this operation permanently removes all unreadable records from the selected table, and cannot be undone.

Cleaning up unreadable records is done by selecting in the 'Administration' area **Actions > Clean up unreadable records** in the navigation pane.

Cleaning up other repository entities

It is the *administrator's responsibility* to monitor and regularly cleanup the following entities.

Purge

A purge can be executed to clean up the remaining data from *all* deletions, that is, deleted dataspaces, snapshots and history performed up until that point. A purge can be initiated by selecting in the 'Administration' area **Actions > Execute purge** in the navigation pane.

Task scheduler execution reports

Task scheduler execution reports are persisted in the 'executions report' table, in the 'Task scheduler' section of the 'Administration' area. Scheduled tasks constantly add to this table as they are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

User interactions

User interactions are used by the EBX component as a reliable means for an application to initiate and get the result of a service execution. They are persisted in the *ebx-interactions* administration section. It is recommended to regularly monitor the user interactions table, as well as to clean it, if needed.

Workflow history

The workflow events are persisted in the workflow history table, in the 'Workflow' section of the 'Administration' area. Data workflows constantly add to this table as they are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

The steps to clean history are the following

- Make sure the process executions are removed (it can be done by selecting in the 'Administration' area of Workflows **Actions > Terminate and clean this workflow** or **Actions > Clean from a date** in the navigation pane).
- Clean main processes in history (it can be done by selecting in the 'Administration' area of Workflows history **Actions > Clear from a date** or **Actions > Clean from selected workflows** in the navigation pane).
- Purge remaining entities in workflow history using 'standard EBX purge'

See also*the standard EBX purge* *[p 378]*

Monitoring and clean up of file system

Attention

In order to guarantee the correct operation of EBX, the disk usage and disk availability of the following directories must be supervised by the administrator, as EBX does not perform any clean up:

- **XML audit trail**: `${ebx.repository.directory}/History/`
- **Archives**: `${ebx.repository.directory}/archives/`
- **Logs**: ebx.logs.directory [p 351]
- **Temporary directory**: ebx.temp.directory [p 349]

Attention

For **XML audit trail**, if large transactions are executed with full update details activated (default setting), the required disk space can increase.

Attention

For pagination in the data services `getChanges` operation, a persistent store is used in the **Temporary directory**. Large changes may require a large amount of disk space.

See also

XML audit Trail *[p 419]*

Tuning the EBX repository [p 360]

62.5 Dataspaces

Some dataspace administrative tasks can be performed from the 'Administration' area of EBX by selecting 'Dataspaces'.

Dataspaces/snapshots

This table lists all the existing dataspaces and snapshots in the repository, whether open or closed. You can view and modify the information of dataspaces included in this table.

See also*Dataspace information* [p 96]

From this section, it is also possible to close open dataspaces, reopen previously closed dataspaces, as well as delete and purge open or closed dataspaces, associated history, and snapshots.

See also*Cleaning up dataspaces, snapshots, and history* [p 377]

Dataspace permissions

This table lists all the existing permission rules defined on all the dataspaces in the repository. You can view the permission rules and modify their information.

See also*Dataspace permissions* [p 97]

Repository history

The table 'Deleted dataspaces/snapshots' lists all the dataspaces that have already been purged from the repository.

From this section, it is also possible to delete the history of purged dataspaces.

CHAPTER 63
UI administration

TIBCO EBX comes with a full user interface called Advanced perspective [p 384] that includes all available features. The interface is fully customizable [p 387] (custom logo, colors, field size, default values, etc.) and available to built-in administrators.

Access to the advanced perspective can be restricted in order to simplify the end-user experience, through global permissions [p 383], giving the possibility to grant or restrict access to functional categories. Administrators can create simplified perspectives called recommended perspectives [p 395] for end-users, containing only the features and menus they need for their daily tasks.

This chapter contains the following topics:

63.1 Global permissions

Global permission rules can be created in EBX.

The 'Display area' property allows restricting access to areas of the user interface. To define the access rules, select 'Global permissions' in the 'Administration' area.

Profile	Indicates on which profile the rule will be applied.
Restriction policy	Indicates if the permissions defined here restrict the ones defined for other profiles. See the Restriction policy concept [p 283] for more information.
Dataspaces	Defines permissions for the Dataspaces area.
Data Models	Defines permissions for the Data Models area.
Workflow Models	Defines permissions for the Workflow Models area.
Data Workflows	Defines permissions for the Data Workflows area.
Data Services	Defines permissions for the Data Services area. Independently, it is also possible to: • Defines permissions for the REST built-in connector HTTP(S). This setting does not impact the REST Toolkit applications. • Defines permissions for the SOAP connector HTTP(S) and JMS. • Defines permissions for the WSDL connector HTTP(S).
Administration	Defines permissions for the Administration area.

Note

Permissions can be defined by administrators and by the dataspace or dataset owner.

63.2 Advanced perspective

The advanced perspective and its parameterization are unique. It is the parent perspective from which any new perspective [p 395] will inherit.

Children perspectives can be created from that main perspective in order to offer a customized, simplified menu to the end-users. Thanks to dataset inheritance, these simplified perspectives will receive their parameters from the advanced perspective (the root dataset). These parameters can then be overridden on the newly created simplified perspectives. Simplified perspectives can be created underneath an existing simplified perspective, thus inheriting from the parent's parameters.

See also*Inheritance* *[p 27]*

The advanced perspective is available by default to all end-users but access can be restricted.

Note: Administrators can always access the advanced perspective even when it is deactivated.

It is possible to configure which perspective is applied by default when users log in. This 'default perspective' is based on two criteria: 'recommended perspectives', defined by administrators and 'favorite perspectives', defined by users.

See also

Recommended perspectives *[p 395]*

Favorite perspectives *[p 19]*

Perspective creation

To create a perspective, open the 'Select an administration feature' drop-down menu and click on the + sign to create a child dataset.

See also*Creating an inheriting child dataset* *[p 112]*

User interface

Options are available in the Administration area for configuring the web interface, in the 'User interface' section.

Attention

Be careful when configuring the URL policy (deprecated) [p 386]. If the web interface configuration is invalid, it can lead to the unusability of EBX. If this occurs, use the "rescue mode" by setting `frontEnd.rescueMode.enable=true` in EBX main configuration file [p 345], and accessing the following URL in your browser as a built-in administrator user: `http://.../ebx/?onwbpID=iebx-manager-rescue`.

Session configuration

These parameters configure the user session options:

User session default locale	Default session locale
Session time-out (in seconds)	Maximum duration of user inactivity before the session is considered inactive and is terminated. A negative value indicates that the session should never timeout.

Interface configuration

Entry policy

Describes the URL to access the application.

Login URL	If the user is not authenticated, the session is forwarded to this URL.

The entry policy defines an EBX login page, replacing the default one.

If defined,

- it replaces an authentication URL that may have been defined using a specific user `Directory`API,
- it is used to build the permalinks in the user interface,
- if the URL is full, that is, starting with `http://` or `https://`, it replaces the URL of the workflow email configuration.

URL policy (deprecated)

Describes the URL and proxy policy. Both dynamic (servlet) and static (resources) URLs can be configured.

This configuration manner is deprecated and must be replaced by URLs computing [p 356]. After configuring the EBX main configuration file, these configurations must be unset.

HTTP servlet policy	Header content of the servlet HTTP request: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.
HTTPS servlet policy	Header content of the servlet HTTPS request: • if a field is not set, the default value is chosen (in an environment configuration), • if a default value is not set, the value in the initial request is used.
HTTP external resources policy	Header content of the external resources URL in HTTP: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.
HTTPS external resources policy	Header content of the external resources URL in HTTPS: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.

Exit policy

Describes how the application is exited.

Normal redirection	Specifies the redirection URL used when exiting the session normally.
Error redirection	Specifies the redirection URL used when exiting the session due to an error. This feature is now deprecated and may be ignored by EBX.
Redirection restrictions	Specifies the list of authorized domains and whether HTTPS is mandatory for each domain.

Graphical interface configuration

Activation & Allowed profiles

The 'Activated' radio button allows to activate or deactivate the perspective. When deactivated, the perspective will only be made available to the administrator.

The 'Allowed profiles' feature is used to give access to the perspective to a given profile. Several profiles can be added to the list of authorized profiles by clicking on the + icon below the numbered list.

The available perspective properties are:

Activated	Indicates if the perspective is visible to authorized users.
Allowed profiles	The list of authorized user profiles for the perspective.
Allowed devices	The list of authorized devices for the perspective. If not specified, only "EBX Web Application" can display this perspective.
Default selection	The menu item that is selected by default. This property is not available for the advanced perspective.

Application locking

EBX availability status:

Availability status	This application can be closed to users during maintenance (but still remain open to administrators). Takes effect immediately.
Unavailability message	Message displayed to users when access is restricted to administrator profiles.

Security policy

EBX access security policy. These parameters only apply to new HTTP sessions.

IP access restriction	Restricts access to designated IP addresses (see IP pattern below).
IP restriction pattern	Regular expression representation of IP addresses authorized to access EBX. For example, `((127\.0\.0\.1) | (192\.168\.*\.*))` grants access to the local machine and the network IP range `192.168.*.*`.
Unique session control	Specifies whether EBX should control the uniqueness of user sessions. When set to 'Yes', if a user does not log out before closing the browser, it will not be possible for that user to log in again until the previous session has timed out.

Ergonomics and layout

EBX ergonomics parameters:

Max table columns to display	According to network and browser performance, adjusts the maximum number of columns to display in a table. This property is not used when a view is applied on a table.
Maximum auto-width for table columns	Defines the maximum width to which a table column can auto-size during table initialization. This is to prevent columns from being too wide, which could occur for very long values, such as URLs. Users will still be able to manually resize columns beyond this value.
Max expanded elements for a hierarchy	Defines the maximum number of elements that can be expanded in a hierarchy when using the action "Expand all". A value less than or equal to '0' disables this parameter.
Default table filter	Defines the default table filter to display in the filters list in tabular views. If modified, users must log out and log in for the new value to take effect.
Display the message box automatically	Defines the message severity threshold for displaying the messages pop-up.
IE compatibility mode	Defines whether or not to compensate for Internet Explorer 8+ displaying EBX in compatibility mode. In order to prevent Internet Explorer browsers from using compatibility mode when displaying the repository user interface, the meta-tag *http-equiv="X-UA-Compatible" content="IE=EmulateIE8"* is added to the header of pages. However, in some local environments, this setting may conflict with existing policies, in which case this header must be omitted by setting the parameter to 'No'. The default value is 'Yes'. See Specifying Document Compatibility Modes for more information.
Forms: width of labels	The width of labels in forms.
Forms: width of inputs	The width of form input fields in forms.
Forms: height of text areas	The height of text entry fields in forms.

Forms: aggregated lists	The number of hidden candidate lines to be generated, available to create new instances in the list.
Forms: width of HTML editor	The width of HTML editors in forms.
Forms: height of HTML editor	The height of HTML editors in forms.
Searchable list selection page size	Maximum number of rows downloaded at each request of the searchable list selection (used for selecting foreign keys, enumerations, etc.).
Record form: rendering mode for nodes	Specifies how to display non-terminal nodes in record forms. This should be chosen according to network and browser performance. For impact on page loading, link mode is light, expanded and collapsed modes are heavier. If this property is modified, users are required to log out and log in for the new value to take effect.
Record form: display of selection and association nodes in creation mode	If enabled, the selection and association nodes will be displayed in record creation forms.
Display density	Defines the default display density mode for all users. If no density has been selected by the user yet, this value will be applied. Conversely, if the user already chose a density, their choice will prevail.
Avatar displayed in the header	This property defines the display mode of avatars in the header. For example, it is possible to enable or disable the use of avatars in the header by updating this property. If no value is defined, the default value is 'Avatar only'. If it is a relative path, prefix it with "../" to get back to the application root URL.
Avatar displayed in the history	This property defines the display mode of avatars in the history. For example, it is possible to enable or disable the use of avatars in the history by updating this property. If no value is defined, the default value is 'Avatar only'. If it is a relative path, prefix it with "../" to get back to the application root URL.
Avatar displayed in the workflow	This property defines the display mode of avatars in the workflow. For example, it is possible to enable or disable the use of avatars in the workflow by updating this property. If no value is defined, the default value is 'Avatar only'. If

	it is a relative path, prefix it with "../" to get back to the application root URL.

Default option values

Defines default values for options in the user interface.

Import/Export

CSV file encoding	Specifies the default character encoding to use for CSV file import and export.
CSV : field separator	Specifies the default separator character to use for CSV file import and export.
CSV : list separator	Specifies the default list separator character to use for CSV file import and export.
Import mode	Specifies the default import mode.
Missing XML values as 'null'	If 'Yes', when updating existing records, if a node is missing or empty in the imported file, the value is considered as 'null'. If 'No', the value is not modified.

Colors and themes

Customizes EBX colors and themes.

Web site icon URL (favicon)	Sets a custom favicon. The recommended format is ICO, which is compatible with Internet Explorer.
Logo URL (SVG)	Specifies the SVG image used for compatible browsers. Leave this field blank to use the PNG image, if specified. The user interface will attempt to use the specified PNG image if the browser is not SVG-compatible. If no PNG image is specified, the GIF/JPG image will be used. The logo must have a maximum height of 40px. If the height exceeds 40px, it will be cropped, not scaled, to a height of 40px. The width of the logo will determine the position of the buttons in the header. If it is a relative path, prefix it with "../" to get back to the application root URL.
Logo URL (PNG)	Specifies the PNG image used for compatible browsers. Leave both this field and the SVG image URL field blank to use the GIF/JPG image. The user interface will use the GIF/JPG image if the browser is not PNG-compatible. The logo must have a maximum height of 40px. If the height exceeds 40px, it will be cropped, not scaled, to a height of 40px. The width of the logo will determine the position of the buttons in the header. If it is a relative path, prefix it with "../" to get back to the application root URL.
Logo URL (GIF/JPG)	Specifies the GIF/JPG image to use when neither the PNG nor SVG are defined. The recommended formats are GIF and JPG. The logo must have a maximum height of 40px. If the height exceeds 40px, it will be cropped, not scaled, to a height of 40px. The width of the logo will determine the position of the buttons in the header. If it is a relative path, prefix it with "../" to get back to the application root URL.
Main	Main user interface theme color, used for selections and highlights.
Header	Background color of the user interface header. By default, set to the same value as the Main color.
Workflow badge	Background and text/outline colors of new workflow task counters.
Primary buttons	Color of buttons selected by default. By default, set to the same value as the Main color.

Text of link style buttons	Text color of some buttons having a link style (the text is not dark or light, but colored). By default, set to the same value as the main color.
Selected tab border	Border color of the selected tab. By default, set to the same value as the Main color.
Table history view: technical data	Background color of technical data cells in the table history view.
Table history view: creation	Background color of cells with the state 'creation' in the table history view.
Table history view: deletion	Background color of cells with the state 'deletion' in the table history view.
Table history view: update	Background color of cells with the state 'update' in the table history view.

Child perspective menu

An unlimited number of child perspectives can be created. Child perspectives inherit from the parameters of the 'Advanced perspective'. Some of these parameters can be overridden as detailed hereafter.

Activation & Allowed profiles

See Activation and Allowed profiles for the Advanced perspective [p 387] for more information.

> **Note**
>
> Any specific parameter set for this perspective will override the default parameters that have been set in the 'Advanced perspective' configuration.

Perspective Menu

This view displays the perspective menu. It is a hierarchical table view.

From this view, a user can create, delete or reorder menu item records.

See also*Hierarchical table view* *[p 27]*

Section Menu Item	This is a top level menu item. It contains other menu items.
Menu group	This is a container for other menu items.
Action Menu Item	This menu item displays a user service in the workspace area.

Menu item properties

When creating a record in the 'Perspective' Menu, the available perspective properties are:

Type	The menu item type. **See also***Menu item types* *[p 393]*
Parent	The parent of the menu item. This property is not available for section menu items.
Label	The menu item label. The label is optional for action menu items. If not specified, the label will be dynamically generated by EBX when the menu item is displayed.
Allowed devices	The list of authorized devices for this item. If not specified, all devices can display this menu item. Currently only two devices are supported:"EBX Web Application" and "EBX GO".
Icon	The icon for the menu item. Icon can be either "standard" (provided by EBX) or an image, specified by a URL, that can be hosted on any web server. Icons size should be 16x16 pixels. This property is not available for section menu items.
Top separator	Indicates that the menu item section has a top separator. This property is only available for section menu items.
Action	The user service to execute when the user clicks on the menu item. **See also***User interface services* *[p 563]* If an end-user is allowed to view the perspective but not to execute the user service, an "access denied" message will be displayed when the user clicks on the menu item. This property is only available for action menu items.
Selection on close	The menu item that will be selected when the service terminates. Built-in services use this property when the user clicks on the 'Close' button.

This property is only available for action menu items.

Ergonomics and layout

See Ergonomics and layout for the Advanced perspective [p 389] for more information.

> **Note**
>
> Any specific parameter set for this perspective will override the default parameters that have been set in the 'Advanced perspective' configuration.

Colors and themes

See Colors and themes for the Advanced perspective [p 392] for more information.

> **Note**
>
> Any specific parameter set for this perspective will override the default parameters that have been set in the 'Advanced perspective' configuration.

63.3 Recommended perspectives

It is possible for a perspective administrator to configure recommended perspectives dedicated to a specific audience. These recommended perspectives are a way to choose which perspective is applied by default when a user logs in, based on their role.

However, users always have the possibility to switch between the various perspectives that are available to them and to set one as their favorite. See Favorite perspectives [p 19] for more information.

To configure recommended perspectives, go to *User interface > Recommended perspectives > Manage recommended perspectives*.

Managing recommended perspectives

The main screen shows an ordered list of records associating a profile with a perspective. Note that the order here is important since a user can match more than one record (see Resolution [p 395] for more information).

- To add an entry, use the 'Create' action.
- To edit an entry, first select it in the list by clicking on it, then click on the 'Edit' action, or simply double-click on it.
- To remove an entry, first select it in the list, then click on the 'Delete' action.
- To move an entry, first select it in the list, then use the actions in the toolbar to the right of the list.

Resolution

When a user logs in, the following algorithm determines which perspective is selected by default:

```
// 1) favorite perspective
IF the user has a favorite perspective
AND this perspective is active
AND the user is authorized for this perspective
    SELECT this perspective
    DONE

// 2) recommended perspective
FOR EACH association in the recommended perspectives list, in the declared order
    IF the user is in the declared profile
    AND the associated perspective is active
    AND the user is authorized for the associated perspective
        SELECT this perspective
        DONE

// 3) advanced perspective
IF the advanced perspective is active
AND the user is authorized for this perspective
    SELECT this perspective
    DONE

// 4) any perspective
SELECT any active perspective for which the user is authorized
DONE
```

63.4 Custom views

Users can create and manage custom views directly from the 'View' menu on tables. This administration section is the central point to manage these custom views.

Views

This table contains all custom views defined on any table. Only a subset of fields is editable:

Documentation	Localized labels and descriptions.
Owner	Defines the user(s) owning and authoring this view definition.
View group	Indicates the menu group in which this view is displayed in the 'View' menu.
Share with	Defines the users allowed to select this view from their 'View' menu.

Views permissions

This table allows to manage permissions relative to custom views, by data model and profile. The following permissions can be configured (the default value is applied when no permission is set for a given user):

Permission	Description	Default value
Recommend views	Allows the user to manage recommended views.	If the user is the dataset owner, the default value is 'Yes', otherwise it is 'No'.
Manage views	Defines the views the user can modify and delete.	If the user is a built-in administrator, the default value is 'Owned + shared', otherwise it is 'Owned'.
Share views	Defines the views for which the user can edit the 'Share with' field.	If the user is a built-in administrator, the default value is 'Owned + shared', else if the user is the dataset owner, it is 'Owned', otherwise it is 'None'.
Publish views	Allows the user to publish views to make them available to all users using Web components, workflow user tasks, or data services.	If the user is a built-in administrator, the default value is 'Yes', otherwise it is 'No'.

63.5 User session management

This tool lists all user sessions and allows terminating active sessions when necessary.

For example: it is possible to invalidate and terminate all currently open and active sessions for maintenance purposes. The access to the user interface can be temporarily closed, with an unavailability message being displayed, through Application locking [p 387]. After active sessions are terminated, users will not be able to reconnect and will see the unavailability message. The maintenance operation can then be performed.

CHAPTER 64
Users and roles directory

This chapter contains the following topics:

64.1 Overview

TIBCO EBX uses a directory for user authentication and user role definition.

A default directory is provided and integrated into the EBX repository; the 'Directory' administration section allows defining which users can connect and what their roles are.

It is also possible to integrate another type of enterprise directory.

See also

Configuring the user and roles directory [p 349]

Custom directory [p 402]

64.2 Concepts

In EBX, a user can be a member of several roles, and a role can be shared by several users. Moreover, a role can be included into another role. The generic term *profile* is used to describe either a user or a role.

In addition to the directory-defined roles, EBX provides the following *built-in roles*:

Role	Definition
Profile.ADMINISTRATOR	Built-in Administrator role. Allows performing general administrative tasks.
Profile.READ_ONLY	Built-in read-only role. A user associated with the read-only role can only view the EBX repository, and has no right to perform modifications in the repository.
Profile.OWNER	Dynamic built-in owner role. This role is checked dynamically depending on the current element. It is only activated if the user belongs to the profile defined as owner of the current element.
Profile.EVERYONE	All users belong to this role.

Information related to profiles is primarily defined in the directory.

Attention

Associations between users and the built-in roles *OWNER* and *EVERYONE* are managed automatically by EBX, and thus must not be modified through the directory.

User permissions are managed separately from the directory. See Permissions [p 275].

See also

profile *[p 23]*

role *[p 24]*

user *[p 23]*

administrator *[p 24]*

user and roles directory *[p 24]*

Policy

These properties configure the policies of the user and roles directory, for example, whether or not users can edit their own profiles.

Users

This table lists all the users defined in the internal directory. New users can be added from there.

Roles

This table lists all the users defined in the internal directory. New roles can be created in this table.

64.3 **Default directory**

Directory content

The default directory is represented by the dataset 'Directory', in the 'Administration' area.

This dataset contains tables for users and roles, as well as users' roles table, roles' inclusions table and salutations table.

> **Note**
>
> If a role inclusion cycle is detected, the role inclusion is ignored at the permission resolution. Refresh and check the directory validation report for cycle detection.

> **Note**
>
> Users' roles, roles' inclusions and salutations tables are hidden by default [p 544].

Depending on the policies defined, users can modify information related to their own accounts, regardless of the permissions defined on the directory dataset.

> **Note**
>
> It is not possible to delete or duplicate the default directory.

Password recovery procedure

In the default directory, passwords are encrypted (by default with a SHA256-like algorithm), and stored in this state. Consequently, it is impossible to retrieve lost passwords. A new password must be generated and sent to the user.

There are two options for this procedure:

1. A notification email is sent to the administrator, the administrator manually changes the password and sends the new password to the user.
2. A procedure automatically generates a new password and sends it to the user.

By default, the first option is used. To activate the second option, specify the property `ebx.password.remind.auto=true` in the TIBCO EBX main configuration file [p 345].

> **Note**
>
> For security reasons, the password recovery procedure is not available for administrator profiles. If required, use the administrator recovery procedure instead.

Administrator recovery procedure

If all the 'login/password' credentials of the administrators are lost, a special procedure must be followed. A specific directory class redefines an administrator user with login 'admin' and password 'admin'.

To activate this procedure:

- Specify the following property in the TIBCO EBX main configuration file [p 345]:

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

- Start EBX and wait until the procedure completes.
- Reset the '`ebx.directory.factory`' property.
- Restart EBX and connect using the 'admin' account.

> **Note**
>
> While the '`ebx.directory.factory`' property is set for the recovery procedure, authentication of users will be denied.

64.4 **Custom directory**

As an alternative to the default directory, it is possible to integrate a specific company directory. For example, an LDAP instance, a relational database or a specific directory model instantiated into EBX.

See also *`DirectoryFactory`* API

CHAPTER **65**

Data model administration

This chapter contains the following topics:

65.1 Administrating publications and versions

Technical data related to data model publications and versions can be accessed in the *Administration* section by an administrator.

Data Modeling contains the following two tables:

- *Publications*. Stores the publications available in the repository.
- *Versions*. Stores the versions of the data models available in the repository.

These tables are read-only but it is however possible to delete manually a publication or a version.

Important: If a publication or a version is deleted, then the content of associated datasets will become unavailable. So this technical data must be deleted with caution.

It is possible to spread this technical data to other TIBCO EBX repositories exporting an archive from an EBX repository and importing it to another one. It may be useful for propagating the evolutions of data models to other repositories.

65.2 Migration of previous data models in the repository

In versions before *5.2.0*, published data models not depending on a module were generated in the file system directory `${ebx.repository.directory}/schemas/`, with the name of the data model (*product.xsd* for example if the data model is named *Product*). Since the *5.2.0* version, this kind of data model is now fully managed within EBX through *Publications*. That is, republishing an existing data model migrates it as a *Publication* and redirects linked datasets to the new embedded data model. The previous XML Schema Document located in `${ebx.repository.directory}/schemas/` is renamed and suffixed with *toDelete*, meaning that the document is no longer used and can be safely deleted.

65.3 Schema evolutions

It is crucial to evaluate the impact of data model changes on the administration side. The following points are to be considered:

Impacts on data persistence

Administration tasks can be related to the database cleanup after a modification of the models. The following links describe how the evolutions of data models are managed at the persistence level: Cleaning up tables having unreadable records [p 379] and Purging master tables in the database [p 407].

Impacts on side features

Some components rely heavily on the data models and can be impacted by their evolutions. Some examples are: the user interface, the WSDL documents, existing archives, etc.

The 'Administration' section offers the possibility to manage some of these components (such as the views), whereas other components fall out of the administrator's scope, such as archives, WSDL files, etc.

CHAPTER 66

Database mapping administration

This chapter contains the following topics:

66.1 Overview

Information and services relative to database mapping can be found in the *Administration* area.

See also

Mapped modes *[p 243]*

`DatabaseMapping`API

66.2 Renaming columns in the database

This feature is available on the 'Columns' table records, under the 'Actions' menu. It allows renaming a column in the database.

The administrator can specify the name of each column of the data model in the database for mapped modes.

Once the service is selected on a record, a summary screen displays information regarding the selected column and the administrator is prompted to enter a new name for the column in the database.

> **Note**
>
> It is required that the new identifier begins with a letter.
>
> Besides, the new name must be a valid column identifier, which depends on the naming rules of the underlying RDBMS.

See also`DatabaseMapping`API

66.3 **Purging columns in the database**

This feature is available on the 'Columns' table records, under the 'Actions' menu. It allows purging columns in mapped structures.

A column can be purged if it has been disabled for mapped modes.

A column is disabled for mapped modes when:

- the corresponding field has been removed from the data model, or
- the corresponding field has been changed in the data model, in a way that is not compatible (for example: its data type has been modified), or
- the defined mapped modes have been disabled locally on the corresponding fields, using the elements `osd:history` and `osd:replication`.

See also

Disabling history on a specific field or group *[p 252]*

Disabling replication on a specific field or group *[p 261]*

Note that this behavior will change for aggregated lists:

- when deactivating a complex aggregated list, its inner fields will still be in the `LIVING` state, whereas the list node is disabled. As lists are considered as auxiliary tables in the mapping system, this information can be checked in the 'Tables' table,
- on the other hand, when the deactivation is just for inner nodes of the list, then the list will remain `LIVING`, while its children will be `DISABLED IN MODEL`.

A column can be purged only if its own state is `DISABLED IN MODEL`, or if it is an inner field of a `DISABLED IN MODEL` list.

66.4 **Renaming master tables in the database**

This feature allows renaming master tables for both relational and history modes in the database. However, it is not available for replicated tables since their names are specified in the data model.

Both features are available on the 'Tables' table records, under the 'Actions' menu.

Master tables are database tables used for persisting the tables of the data model.

The administrator can specify in the database the name of each master table corresponding to a table of the data model.

Once the service is selected on a record, a summary screen displays information regarding the selected master table and the administrator is prompted to enter a new name for the master table in the database.

> **Note**
>
> It is required that the new identifier begins with a letter and with the repository prefix.
>
> For history tables, it is also required that the repository prefix is followed by the history tables prefix.
>
> For relational tables, it is also required that the repository prefix is followed by the relational tables prefix.
>
> Besides, the new name must be a valid table identifier, which depends on the naming rules of the underlying RDBMS.

66.5 Renaming auxiliary tables in the database

This feature allows renaming history auxiliary tables in the database. This feature is neither available for replicated tables since their names are specified in the data model, nor for the relational mode, as aggregated lists are never supported in this mode.

This feature is available on the 'Tables' table records, under the 'Actions' menu.

Auxiliary tables are database tables used for persisting aggregated lists.

The administrator can specify in the database the name of each auxiliary table corresponding to an aggregated list of the data model.

Once the service is selected on a record, a summary screen displays information regarding the selected auxiliary table and the administrator is prompted to enter a new name for the auxiliary table in the database.

> **Note**
>
> It is required that the new identifier begins with a letter.
>
> It is required that the new identifier begins with the repository prefix.
>
> It is also required that the repository prefix is followed by the history tables prefix.
>
> Besides, the new name must be a valid table identifier, which depends on the naming rules of the underlying RDBMS.

66.6 Purging master tables in the database

This feature allows purging history and relational tables in the database if they are no longer used.

It is available on the 'Tables' table records, under the 'Actions' menu, and is only available for master tables. This feature only applies to master tables. When a master table is purged, all its auxiliary tables are purged as well.

A mapped table can be purged in the database only if it has been disabled for the corresponding mapped mode.

To disable the mapped mode for a table, follow the procedure hereafter.

For history mode:

- Deactivate historization of the table in the data model, or
- Remove the table from the data model

For relational mode:

- Remove the table from the data model, or
- Deactivate the relational mode on the data model. As data models in semantic mode cannot be used for relational dataspaces, it is thus necessary to create a dataset on a semantic dataspace using this modified data model. TIBCO EBX will then detect that the relational mode was previously used, and will therefore propose the relational table database resources for purge.

CHAPTER 67

Workflow management

This chapter contains the following topics:

67.1 Workflows

To define general parameters for the execution of data workflows, the management of workflow publications, or to oversee data workflows in progress, navigate to the 'Administration' area. Click on the down arrow in the navigation pane and select *Workflow management* > *Workflows*.

Note

In cases where unexpected inconsistencies arise in the workflow execution technical tables, data workflows may encounter errors. It may then be necessary to run the operation 'Clean up inconsistencies in workflow execution tables' from the 'Actions' menu in the navigation pane under *Administration* > *Workflow Management* > *Workflows*.

Execution of workflows

Various tables can be used to manage the data workflows that are currently in progress. These tables are accessible in *Workflow management* > *Workflows* in the navigation pane.

See also*Administration of data workflows* *[p 175]*

Workflows table

The 'Workflows' table contains instances of all data workflows in the repository, including those invoked as sub-workflows. A data workflow is a particular execution instance of a workflow model publication. This table provides access to the data context variables for all data workflows. It can be used to access the status of advancement of the data workflow in terms of current variable values, and in case of a data workflow suspension, to modify the variable values.

From the 'Actions' menu of the 'Workflows' table, it is possible to clear the completed data workflows that are older than a given date, by selecting the 'Clean from a date' service. This service automatically ignores the active data workflows.

Tokens table

The 'Tokens' table allows managing the progress of data workflows. Each token marks the current step being executed in a running data workflow, as well as the current state of the data workflow.

See also*token* *[p 31]*

Work items table

The 'Work items' table contains all the work items associated with user tasks that currently exist. If necessary, you can manually allocate a work item to a user from this table in the case of a blockage in a data workflow. It is preferable, however, to use the buttons in the workspace of the 'Data workflows' area whenever possible to allocate, reallocate, and deallocate work items.

See also*work item* *[p 31]*

Waiting workflows table

The 'Waiting workflows' table contains all the workflows waiting for an event. If needed, a service is available to clean this table: this service deletes all lines associated with a deleted workflow.

See also*wait task* *[p 30]*

Comment table

The 'Comments' table contains the user's comments for main workflows and their sub-workflows.

Workflow publications

The 'Workflow publications' table is a technical table that contains all the workflow model publications of the repository. This table associates published workflow models with their snapshots. It is not recommended to directly modify this table, but rather to use the actions available in the workflow modeling area to make changes to publications.

Configuration

Email configuration

In order for email notifications to be sent during the data workflow execution, the following settings must be configured under 'Email configuration':

- The URL definition field is used to build links and value mail variables in the workflow.
- The 'From email' field must be completed with the email address that will be used to send email notifications.

Interface customization

Modeling default values

The default value for some properties can be customized in this section.

The administrator has the possibility to define the default values to be used when a new workflow model or workflow step is created in the 'Workflow Modeling' section.

Work items views

Specific columns are available in the inbox and in the monitoring work items tables, in the 'Data workflows' section.

10 specific columns are available. For each specific column, a customized label can be defined.

Priorities configuration

The property 'Default priority' defines how data workflows and their work items across the repository display if they have no priority level. For example, if this property is set to the value 'Normal', any workflow and work item with no priority will appear to have the 'Normal' priority.

The 'priorities' table defines all priority levels available to data workflows in the repository. As many integer priority levels as needed can be added, along with their labels, which will appear when users hover over the priority icon in the work item tables. The icons that correspond to each priority level can also be selected, either from the set provided by TIBCO EBX, or by specifying a URL to an icon image file.

Temporal tasks

Under 'Temporal tasks', the polling interval for time-dependent tasks in the workflow can be set, such as deadlines and reminders. If no interval value is set, the 'in progress' steps are checked every hour.

Workflow inbox counter configuration

The workflow inbox counter is refreshed asynchronously, even if the end-user does not launch any action. To adjust it, two parameters need to be set:

Cache expiry (seconds)	Expiration time (in seconds) before a new update of the inbox cache. Please note that this parameter can impact the CPU load and performance since the computation time can be costly for a repository with many work items. If no value is defined, the default value is 600.
User interface refresh periodicity (seconds)	Refresh time (in seconds) between two updates of the inbox counter in the user interface. Please note that this refresh concerns all inbox counters in the user interface: inbox counters of the custom perspective, header inbox counter and Data Workflows inbox counter for the advanced perspective. If no value is defined, default value is 5. If the value is zero (or negative), the refresh is disabled. Also, the modification will only be effective after a logout/login from the user.

Also, please note that some actions can force the inbox counter to refresh:

- access on **Data workflows**
- access on any subdivision of the **Data workflows section**
- accept or reject a work item
- launch a workflow

These parameters are accessible in *Workflow management > Workflows > Configuration > Temporal tasks* in the navigation pane.

67.2 Interactions

To manage workflow interactions, navigate to the Administration area. Click the down arrow in the navigation pane and select the entry *Workflow management > Interactions*.

An *interaction* is generated automatically for every work item that is created. It is a local data context of a work item and is accessible from an EBX session. When a work item is executed, the user performs the assigned actions based upon its interaction, independently of the workflow engine. User tasks define mappings for their input and output parameters to link interactions with the overall data contexts of data workflows.

Interactions can be useful for monitoring the current parameters of work items. For example, an interaction can be updated manually by a trigger or a user service.

67.3 Workflow history

To view the history data workflow execution, browse the 'Administration' area. Click on the down arrow in the navigation pane and select *Workflow management > Workflow history*.

The 'Workflows' table contains all actions that have been performed during the execution of workflows.

This data can be viewed graphically or textually. It is especially useful to view the states of various objects related to workflows at a given moment. This includes actions on work items, variables in the data context, as well as tokens. In case of an error, a technical log is available.

Clean history

From the 'Actions' menu of the 'Workflows' table, the history of completed data workflows older than a given date can be cleared by selecting the 'Clear from a date' service.

Only the history of workflows that have been previously cleaned (e.g. their execution data deleted) is cleared. This service automatically ignores the history associated with existing workflows. It is necessary to clear data workflows before clearing the associated history, by using the dedicated service 'Clear from a date' from the 'Workflows' table. Also, a scheduled 'Clear from a date' can be used with the built-in scheduled task *SchedulerPurgeWorkflowMainHistory*.

Please note that only main processes are cleaned. In order to remove sub-processes and all related data, it will be necessary to run a 'standard EBX purge'.

See also*How to clean workflow history* *[p 380]*

Note

An API is available to fetch the history of a workflow. Direct access to the underlying workflow history SQL tables is not supported. See **WorkflowEngine.getProcessInstanceHistory** `WorkflowEngine.getProcessInstanceHistory`API.

CHAPTER 68

Task scheduler

This chapter contains the following topics:

68.1 Overview

TIBCO EBX offers the ability to schedule programmatic tasks.

> **Note**
>
> In order to avoid conflicts and deadlocks, tasks are scheduled in a single queue.

68.2 Configuration from EBX

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the 'Administration' area.

- **Schedules**: defines scheduling using "cron expressions".
- **Tasks**: configures tasks, including parametrizing task instances and user profiles for their execution.
- **Scheduled tasks**: current schedule, including task scheduling activation/deactivation.
- **Execution reports**: reports of each scheduled task run that appear immediately after the task is triggered. The reports include actions to interrupt, pause, or resume running tasks, when made available by the task definition.

68.3 Cron expression

(An extract of the Quartz Scheduler documentation)

The task scheduler uses "cron expressions", which can create firing schedules such as: "At 8:00am every Monday through Friday" or "At 1:30am every last Friday of the month".

Format

A cron expression is a string comprised of 6 or 7 fields separated by a white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

Field Name	Mandatory	Allowed Values	Allowed Special Characters
Seconds	Yes	0-59	, - * /
Minutes	Yes	0-59	, - * /
Hours	Yes	0-23	, - * /
Day of month	Yes	0-31	, - * ? / L W
Month	Yes	1-12 or JAN-DEC	, - * /
Day of week	Yes	1-7 or SUN-SAT	, - * ? / L #
Year	No	empty, 1970-2099	, - * /

A cron expression can be as simple as this: "**0 * * * * ?**",

or more complex, like this: "**0/5 14,18,3-39,52 * ? JAN,MAR,SEP MON-FRI 2002-2010**".

> **Note**
>
> The legal characters and the names of months and days of the week are not case sensitive. MON is the same as mon.

Special characters

A cron expression is a string comprised of 6 or 7 fields separated by a white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

- ***** ("all values") - used to select all values within a field. For example, "*" in the Minutes field means "every minute".
- **?** ("no specific value") - useful when you need to specify something in one of the two fields in which the character is allowed, but not the other. For example, if I want my trigger to fire on a particular day of the month (say, the 10th), but don't care what day of the week that happens to be, I would put "10" in the day-of-month field, and "?" in the day-of-week field. See the examples below for clarification.
- **-** - used to specify ranges. For example, "10-12" in the hour field means "the hours 10, 11 and 12".
- **,** - used to specify additional values. For example, "MON,WED,FRI" in the day-of-week field means "the days Monday, Wednesday, and Friday".
- **/** - used to specify increments. For example, "0/15" in the seconds field means "the seconds 0, 15, 30, and 45". And "5/15" in the seconds field means "the seconds 5, 20, 35, and 50". You can also

specify '/' after the '' **character - in this case** '' is equivalent to having '0' before the '/'. '1/3' in the day-of-month field means "fire every 3 days starting on the first day of the month".

- **L** ("last") - has different meaning in each of the two fields in which it is allowed. For example, the value "L" in the day-of-month field means "the last day of the month" - day 31 for January, day 28 for February on non-leap years. If used in the day-of-week field by itself, it simply means "7" or "SAT". But if used in the day-of-week field after another value, it means "the last xxx day of the month" - for example "6L" means "the last friday of the month". When using the 'L' option, it is important not to specify lists, or ranges of values, as you'll get confusing results.
- **W** ("weekday") - used to specify the weekday (Monday-Friday) nearest the given day. As an example, if you were to specify "15W" as the value for the day-of-month field, the meaning is: "the nearest weekday to the 15th of the month". So if the 15th is a Saturday, the trigger will fire on Friday the 14th. If the 15th is a Sunday, the trigger will fire on Monday the 16th. If the 15th is a Tuesday, then it will fire on Tuesday the 15th. However if you specify "1W" as the value for day-of-month, and the 1st is a Saturday, the trigger will fire on Monday the 3rd, as it will not 'jump' over the boundary of a month's days. The 'W' character can only be specified when the day-of-month is a single day, not a range or list of days.

> **Note**
>
> The 'L' and 'W' characters can also be combined in the day-of-month field to yield 'LW', which translates to "last weekday of the month".

- **# -** used to specify "the nth" day-of-week day of the month. For example, the value of "6#3" in the day-of-week field means "the third Friday of the month" (day 6 = Friday and "#3" = the 3rd one in the month). Other examples: "2#1" = the first Monday of the month and "4#5" = the fifth Wednesday of the month. Note that if you specify "#5" and there is not 5 of the given day-of-week in the month, then no firing will occur that month.

Examples

Expression	Meaning
0 0 12 * * ?	Fire at 12pm (noon) every day.
0 15 10 ? * *	Fire at 10:15am every day.
0 15 10 * * ?	Fire at 10:15am every day.
0 15 10 * * ? *	Fire at 10:15am every day.
0 15 10 * * ? 2005	Fire at 10:15am every day during the year 2005.
0 * 14 * * ?	Fire every minute starting at 2pm and ending at 2:59pm, every day.
0 0/5 14 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day.
0 0/5 14,18 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at 6pm and ending at 6:55pm, every day.
0 0-5 14 * * ?	Fire every minute starting at 2pm and ending at 2:05pm, every day.
0 10,44 14 ? 3 WED	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
0 15 10 ? * MON-FRI	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday.
0 15 10 15 * ?	Fire at 10:15am on the 15th day of every month.
0 15 10 L * ?	Fire at 10:15am on the last day of every month.
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month.
0 15 10 ? * 6L 2002-2005	Fire at 10:15am on every last friday of every month during the years 2002, 2003, 2004 and 2005.
0 15 10 ? * 6#3	Fire at 10:15am on the third Friday of every month.
0 0 12 1/5 * ?	Fire at 12pm (noon) every 5 days every month, starting on the first day of the month.
0 11 11 11 11 ?	Fire every November 11th at 11:11am.

> **Note**
>
> Pay attention to the effects of '?' and '*' in the day-of-week and day-of-month fields.
>
> Support for specifying both a day-of-week and a day-of-month value is not complete (you must currently use the '?' character in one of these fields).
>
> Be careful when setting fire times between the hours of the morning when "daylight savings" changes occur in your locale (for US locales, this would typically be the hour before and after 2:00 AM - because the time shift can cause a skip or a repeat depending on whether the time moves back or jumps forward.

68.4 Task definition

EBX scheduler comes with some predefined tasks.

Custom scheduled tasks can be added by the means of **scheduler** `Package com.orchestranetworks.scheduler`API Java API.

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the 'Administration' area.

68.5 Task configuration

A user must be associated with a task definition; this user will be used to generate the **session** `Session`API that will run the task.

> **Note**
>
> The user will not be authenticated, and no password is required. As a consequence, a user with no password set in the directory can only be used to run scheduled tasks.

A custom task can be parameterized by means of a JavaBean specification (getter and setter).

Supported parameter types are:

- java.lang.boolean
- java.lang.int
- java.lang.Boolean
- java.lang.Integer
- java.math.BigDecimal
- java.lang.String
- java.lang.Date
- java.net.URI
- java.net.URL

Parameter values are set in XML format.

CHAPTER 69
Audit trail

This chapter contains the following topics:

69.1 Overview

XML audit trail is a feature that allows logging updates to XML files. An alternative history feature is also available to record table updates in the relational database; see History [p 251].

Any persistent updates performed in the TIBCO EBX repository are logged to an audit trail XML file. Procedure executions are also logged, even if they do not perform any updates, as procedures are always considered to be transactions. The following information is logged:

- Transaction type, such as dataset creation, record modification, record deletion, specific procedure, etc.
- Dataspace or snapshot on which the transaction is executed.
- Transaction source. If the action was initiated by EBX, this source is described by the user identity, HTTP session identifier and client IP address. If the action was initiated programmatically, only the user's identity is logged.
- Optional "trackingInfo" value regarding the session
- Transaction date and time (in milliseconds);
- Transaction UUID (conform to the Leach-Salz variant, version 1);
- Error information; if the transaction has failed.
- Details of the updates performed. If there are updates and if history detail is activated, see next section.

69.2 Update details and disk management

The audit trail is able to describe all updates made in the EBX repository, at the finest level. Thus, the XML files can be quite large and the audit trail directory must be carefully supervised. The following should be taken into account:

1. If an archive import is executed in non-interactive mode (without a change set), the audit trail does not detail the updates; it only specifies the archive that has been imported. In this case, if it is important to keep a fine trace of the import-replace, the archive itself must be preserved.
2. If an archive import is executed in interactive mode (with a change set), or if a dataspace is merged to its parent, the resulting log size will nearly triple the unzipped size of the archive. Furthermore, for consistency concerns, each transaction is logged to a temporary file (in the audit trail directory) before being moved to the main file. Therefore, EBX requires *at least six times the unzipped size of the largest archive that may be imported*.
3. In the context of a custom procedure that performs many updates not requiring auditing, it is possible for the developer to disable the detailed history using the method `ProcedureContext.setHistoryActivation`API.

See also*EBX monitoring* *[p 380]*

69.3 File organization

All audit trail files are stored in the directory `${ebx.repository.directory}/History`.

"Closed" audit files

Each file is named as follows:

`<yyyy-mm-dd>-part<nn>.xml`

where `<yyyy-mm-dd>` is the file date and `<nn>` is the file index for the current day.

Writing to current audit files

When an audit file is being written, the XML structure implies working in an "open mode". The XML elements of the modifications are added to a text file named:

`<yyyy-mm-dd>-part<nn>Content.txt`

The standard XML format is still available in an XML file that references the text file. This file is named:

`<yyyy-mm-dd>-part<nn>Ref.xml`

These two files are then re-aggregated in a "closed" XML file when the repository has been cleanly shut down, or if EBX is restarted.

Example of an audit directory

```
2004-04-05-part00.xml
2004-04-05-part01.xml
2004-04-06-part00.xml
2004-04-06-part01.xml
2004-04-06-part02.xml
2004-04-06-part03.xml
2004-04-07-part00.xml
2004-04-10-part00.xml
2004-04-11-part00Content.txt
2004-04-11-part00Ref.xml
```

CHAPTER 70 Other

This chapter contains the following topics:

70.1 Lineage

To administer lineage, three tables are accessible:

- **Authorized profiles**: Profiles must be added to this table to be used for data lineage WSDL generation.
- **History**: Lists the general data lineage WSDLs and their configuration.
- **JMS location**: Lists the JMS URL locations.

70.2 Event broker

Overview

TIBCO EBX offers the ability to receive notifications and information related to specific events using the event broker feature. This feature consists in sending notifications related to EBX core events to the subscriber according to their chosen topics.

Terminology

Event broker	Notification component for loosely-coupled event handling. Consists of dispatching fired events from EBX core to concerned subscribers. The event broker is mainly used for monitoring and statistical purposes.
Topic	Corresponds to the EBX event type that contains messages. The number of subscribers registered to a topic is unlimited.
Subscriber	Client implementation in the modules that receive the events related to the subscribed topic(s).

Topics

Dataspace and snapshot	Corresponds to operations in the dataspace and in the snapshot, such as: create, close, reopen, delete, archive export and archive import (only for dataspace merge).
Repository	Corresponds to operations in the repository, such as: start-up and purge.
User session	Corresponds to the operations related to user authentication, such as: login and logout.

Administration

The management console is located under 'Event broker' in the 'Administration' area. It contains three tables: 'Topics', 'Subscribers' and 'Subscriptions'.

All content is read-only, except for the following operations:

- Topics and subscribers can be manually activated or deactivated using dedicated services.
- Subscribers that are no longer registered to the broker can be deleted.

Distributed Data Delivery (D3)

CHAPTER 71

Introduction to D3

This chapter contains the following topics:

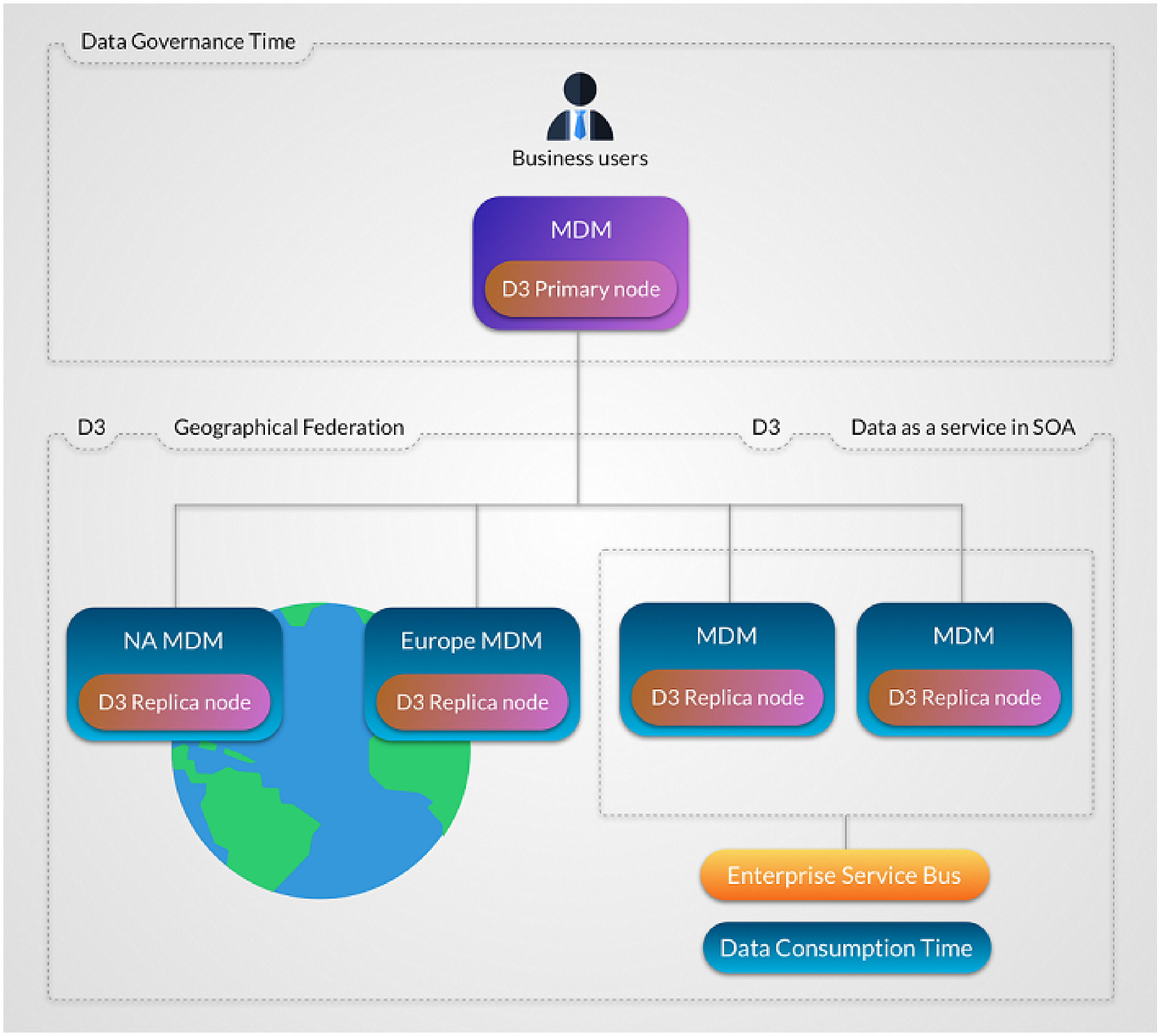
71.1 Overview

TIBCO EBX offers the ability to send data from an EBX instance to other instances. Using a broadcast action, it also provides an additional layer of security and control to the other features of EBX. It is particularly suitable for situations where data governance requires the highest levels of data consistency, approvals and the ability to rollback.

D3 architecture

A typical D3 installation consists of one primary node and multiple replica nodes. In the primary node, a Data Steward declares which dataspaces must be broadcast, as well as which user profile is allowed to broadcast them to the replica nodes. The Data Steward also defines delivery profiles, which are groups of one or more dataspaces.

Each replica node must define from which delivery profile it receives broadcasts.



Involving third-party systems

The features of D3 also allow third-party systems to access the data managed in EBX through data services. Essentially, when a system consumes the data of a delivery dataspace, the data is transparently redirected to the last broadcast snapshot. This ensures a more controlled and reliable view of the managed data.

Third-party systems can either access data directly through the primary node or through a replica node. Thus, a physical architecture consisting of a primary node and no replica nodes is possible.

Protocols

If JMS is activated, the conversation between a primary node and a replica node is based on SOAP over JMS, while archive transfer is based on JMS binary messages.

If JMS is not activated, conversation between a primary node and a replica node is based on SOAP over HTTP(S), while binary archive transfer is based on TCP sockets. If HTTPS is used, make sure that the target node connector is correctly configured by enabling SSL with a trusted certificate.

See also*JMS for distributed data delivery (D3)* *[p 433]*

71.2 **D3 terminology**

broadcast	Send a publication of an official snapshot of data from a primary node to replica nodes. The broadcast transparently and transactionally ensures that the data is transferred to the replica nodes.
delivery dataspace	A delivery dataspace is a dataspace that can be broadcast to authenticated and authorized users using a dedicated action. By default, when a data service accesses a delivery dataspace on any node, it is redirected to the last snapshot that was broadcast. See Data services [p 431].
delivery profile	A delivery profile is a logical name that groups one or more delivery dataspaces. Replica nodes subscribe to one or more delivery profiles.
cluster delivery mode	Synchronization with subscribed replica nodes is performed in a two-phase commit transactional process. This delivery mode is designed to respond to a high volume of queries using load balancing and/or fault tolerance. It ensures the consistency of data in the cluster between replica nodes and their primary node delivery dataspaces. Primary and replica nodes use the same last broadcast snapshots.
federation delivery mode	Synchronization is performed in a single phase, and with each registered replica node independently. This delivery mode is designed to be used with geographically distributed and/or heterogeneous architectures where response time and network availability cannot be guaranteed. At any one time, replica nodes can be at different last broadcast snapshots. The synchronization processes are thus independent of one another and replay of individual replica nodes are performed for certain broadcast failures.
Primary node	An instance of EBX that can define one or more delivery dataspaces, and to which replica nodes can subscribe. A primary node can also act as a regular EBX server.
Replica node	An instance of EBX attached to a primary node, in order to receive delivery dataspace broadcasts. Besides update restrictions on delivery dataspaces, the replica node acts as a regular EBX server.

Hub node	An instance of EBX acting as both a primary node and a replica node. Primary delivery dataspaces and replica node delivery dataspaces **must** be disjoint.

71.3 Known limitations

General limitations

- Each replica node must have only one primary node.
- Embedded data models cannot be used in D3 dataspaces. Therefore, it is not possible to create a dataset based on a publication in a D3 dataspace.
- The compatibility is not assured if at least one replica node product version is different from the primary node.

Broadcast and delivery dataspace limitations

- Access rights on dataspaces are not broadcast, whereas access rights on datasets are.
- Dataspace information is not broadcast.
- Dataspaces defined in relational mode cannot be broadcast.
- If a dataspace and its parent are broadcast, their parent-child relationship will be lost in the replica nodes.
- Once a snapshot has been broadcast to a replica, subsequent broadcasts of *any* snapshot with the same name will result in restoring the originally broadcast version of that same name on the replica node. That is, if the original snapshot on the primary node is purged and a new one is created with the same name and subsequently broadcast, then the content of the replica will be restored to that of the previously broadcast snapshot, and not to the latest one of the same name.
- To guarantee dataspace consistency between D3 nodes, the data model (embedded or packaged in a module) on which the broadcast contents are based, must be the same between the primary node and its replica nodes.
- On a replica delivery dataspace, if several replica nodes are registered, and if replication is enabled in data models, it will be effective for all nodes. No setting is available to activate/deactivate replication according to D3 nodes.
- Replication on replica nodes does not take part in the distributed transaction: it is automatically triggered after commit.

Administration limitations

Technical dataspaces cannot be broadcast, thus the EBX default user directory cannot be synchronized using D3.

CHAPTER 72

D3 broadcasts and delivery dataspaces

This chapter contains the following topics:

72.1 Broadcast

Scope and contents of a broadcast

A D3 broadcast occurs at the dataspace or snapshot level. For dataspace broadcasts, D3 first creates a snapshot to capture the current state, then broadcasts this newly created snapshot.

A broadcast performs one of the following procedures depending on the situation:

- An update of the differences computed between the new broadcast snapshot and the current 'commit' one on the replica node.
- A full synchronization containing all datasets, tables, records, and permissions. This is done on the first broadcast to a given replica node, if the previous replica node commit is not known to the primary node, or on demand using the user service in '[D3] Primary node configuration'.

See also*Services on primary nodes* *[p 446]*

Performing a broadcast

The broadcast can be performed:

- By the end-user, using the **Broadcast** action available in the dataspace or snapshot (this action is available only if the dataspace is registered as a delivery dataspace)
- Using custom Java code that uses `D3NodeAsMaster`API.

Conditions

In order to be able to broadcast, the following conditions must be fulfilled:

- The authenticated user profile has permission to broadcast.

- The dataspace or snapshot to be broadcast has no validation errors.

  Note: Although it is not recommended, it is possible to force a broadcast of a delivery dataspace that contains validation errors. In order to do this, set the maximum severity threshold allowed in a delivery dataspace validation report under '[D3] Primary node configuration' in the 'Administration' area.
- The D3 primary node configuration has no validation errors on the following scope: the technical record of the concerned delivery dataspace and all its dependencies (dependent delivery mappings, delivery profiles and registered replica nodes).
- The dataspace or snapshot does not contain any tables in relational mode.
- There is an associated delivery profile.
- If broadcasting a dataspace, the dataspace is not locked.
- If broadcasting a snapshot, the snapshot belongs to a dataspace declared as delivery dataspace and is not already the current broadcast snapshot (though a rollback to a previously broadcast snapshot is possible).
- The dataspace or snapshot contains differences compared to the last broadcast snapshot.

Persistence

When a primary node shuts down, all waiting or in progress broadcast requests abort, then they will be persisted on a temporary file. On startup, all aborted broadcasts are restarted.

See also*Temporary files* *[p 448]*

> **Note**
>
> Broadcasts are performed asynchronously. Therefore, no information is displayed in the user interface about the success or failure of a broadcast. Nevertheless, it is possible to monitor the broadcast operations inside '[D3] Primary node configuration'. See Supervision [p 447].

Destination

In the target replica or hub node side:

- The `ebx-d3-reference` dataspace identifier is the common parent of all the delivery dataspaces.
- The delivery dataspace has the same identifier in primary, replica or hub nodes.
- If the delivery dataspace is missing, it will be created on the first or on the full synchronization broadcast.
- If the delivery dataspace is already existing on the first broadcast or full synchronization it will be overridden.
- If an existing dataspace with the same identifier of the delivery one is detected outside of the `ebx-d3-reference`, An error will be raisen.

See also*Known limitations* *[p 428]*

72.2 **Replica node registration**

Scope and contents

An initialization occurs at the replica node level according to the delivery profiles registered in the TIBCO EBX main configuration file of the replica node. When the primary node receives that initialization request, it creates or updates the replica node entry, then sends the last broadcast snapshot of all registered delivery dataspaces.

> **Note**
>
> If the registered replica node repository ID or communication layer already exists, the replica node entry in the 'Registered replica nodes' technical table is updated, otherwise a new entry is created.

Performing an initialization

The initialization can be done:

- Automatically at replica node server startup.
- Manually when calling the replica node service 'Register replica node'.

Conditions

To be able to register, the following conditions must be fulfilled:

- The D3 mode must be 'hub' or 'slave'.
- The primary and replica node authentication parameters must correspond to the primary node administrator and replica node administrator defined in their respective directories.
- The delivery profiles defined on the replica node must exist in the primary node configuration.
- All data models contained in the registered dataspaces must exist in the replica node. If embedded, the data model names must be the same. If packaged, they must be located at the same module name and the schema path in the module must be the same in both the primary and replica nodes.
- The D3 primary node configuration has no validation error on the following scope: the technical record of the registered replica node and all its dependencies (dependent delivery profiles, delivery mappings and delivery dataspaces).

> **Note**
>
> To set the parameters, see the replica or hub EBX properties in Configuring primary, hub and replica nodes [p 443].

72.3 **Accessing delivery dataspaces**

Data services

By default, when a data service accesses a delivery dataspace, it is redirected to the current snapshot, which is the last broadcast one. However, this default behavior can be modified either at the request level or in the global configuration.

See also *Common parameter 'disableRedirectionToLastBroadcast'* *[p 618]*

Access restrictions

On the primary node, a delivery dataspace can neither be merged nor closed. Other operations are available depending on permissions. For example, modifying a delivery dataspace directly, creating a snapshot independent from a broadcast, or creating and merging a child dataspace.

On the replica node, aside from the broadcast process, no modifications of any kind can be made to a delivery dataspace, whether by the end-user, data services, or a Java program. Furthermore, any dataspace-related operations, such as merge, close, etc., are forbidden on the replica node.

D3 broadcast Java API

The last broadcast snapshot may change between two calls if a broadcast has taken place in the meantime. If a fully stable view is required for several successive calls, these calls need to specifically refer to the same snapshot.

To get the last broadcast snapshot, see `D3Node.getBroadcastVersion`API.

CHAPTER 73
D3 JMS Configuration

This chapter contains the following topics:

73.1 JMS for distributed data delivery (D3)

To configure D3 to use JMS instead of the default HTTP and TCP protocols, you must configure the JMS connection factory [p 324] and the following queues declared in the `WEB-INF/web.xml` deployment descriptor of the '`ebx`' web application.

Note

If the TIBCO EBX main configuration does not activate JMS and D3 ('slave', 'hub' or 'master' node) through the properties `ebx.d3.mode`, `ebx.jms.activate` and `ebx.jms.d3.activate`, then the environment entries below will be ignored by EBX runtime. See JMS [p 355] and Distributed data delivery (D3) [p 355] in the EBX main configuration properties for more information on these properties.

Common declarations on primary and replica nodes (for shared queues)

Reserved resource name	**Default JNDI name**	**Description**
`jms/EBX_D3MasterQueue`	Weblogic: `EBX_D3MasterQueue` JBoss: `java:/jms/EBX_D3MasterQueue`	D3 primary JMS queue (only for D3 mode 'slave' or 'hub'). It specifies the queue name used to send SOAP requests to the D3 primary node. The message producer sets the primary node repository ID as a value of the header field `JMSType`. Java type: `javax.jms.Queue`
`jms/EBX_D3ReplyQueue`	Weblogic: `EBX_D3ReplyQueue` JBoss: `java:/jms/EBX_D3ReplyQueue`	D3 Reply JMS queue (for all D3 modes except the 'single' mode). It specifies the name of the reply queue for receiving SOAP responses. The consumption is filtered using the header field `JMSCorrelationID`. Java type: `javax.jms.Queue`
`jms/EBX_D3ArchiveQueue`	Weblogic: `EBX_D3ArchiveQueue` JBoss: `java:/jms/EBX_D3ArchiveQueue`	D3 JMS Archive queue (for all D3 modes except the 'single' mode). It specifies the name of the transfer archive queue used by the D3 node. The consumption is filtered using the header field `JMSCorrelationID`. If the archive weight is higher than the threshold specified in the property `ebx.jms.d3.archiveMaxSizeInKB`, the archive will be divided into several sequences. Therefore, the consumption is filtered using the header fields `JMSXGroupID` and `JMSXGroupSeq` instead. Java type: `javax.jms.Queue`
`jms/EBX_D3CommunicationQueue`	WebLogic: `EBX_D3CommunicationQueue` JBoss: `java:/jms/EBX_D3CommunicationQueue`	D3 JMS Communication queue (for all D3 modes except 'single' mode). It specifies the name of the communication queue where the requests are received. The consumption is filtered using the header field `JMSType` which corresponds to the current repository ID. Java type: `javax.jms.Queue`

Note

These JNDI names are set by default, but can be modified inside the web application archive `ebx.war`, included in `EBXForWebLogic.ear` (if using Weblogic) or in `EBX.ear` (if using JBoss, Websphere or other application servers).

Optional declarations on primary nodes (for replica-specific queues)

> **Note**
>
> Used for ascending compatibility prior to 5.5.0 or for mono-directional queues topology.

The deployment descriptor of the primary node must be manually modified by declaring specific communication and archive queues for each replica node. It consists in adding resource names in 'web.xml' inside 'ebx.war'. The replica-specific node queues can be used by one or more replica nodes.

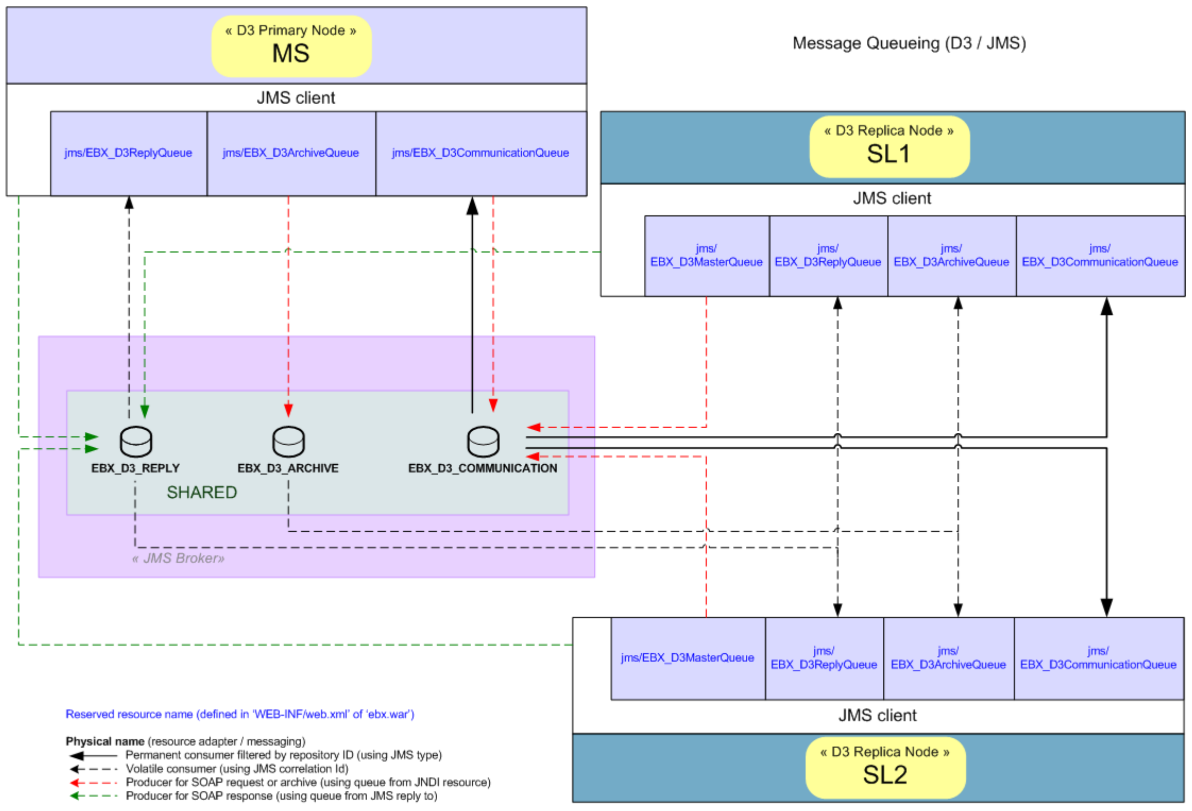
Resources can be freely named, but the physical names of their associated queue must correspond to the definition of replica nodes for resources `jms/EBX_D3ArchiveQueue` and `jms/EBX_D3CommunicationQueue`.

> **Note**
>
> Physical queue names matching: on registration, the replica node sends the communication and archive physical queue names. These queues are matched by physical queue name among all resources declared on the primary node. If unmatched, the registration fails.

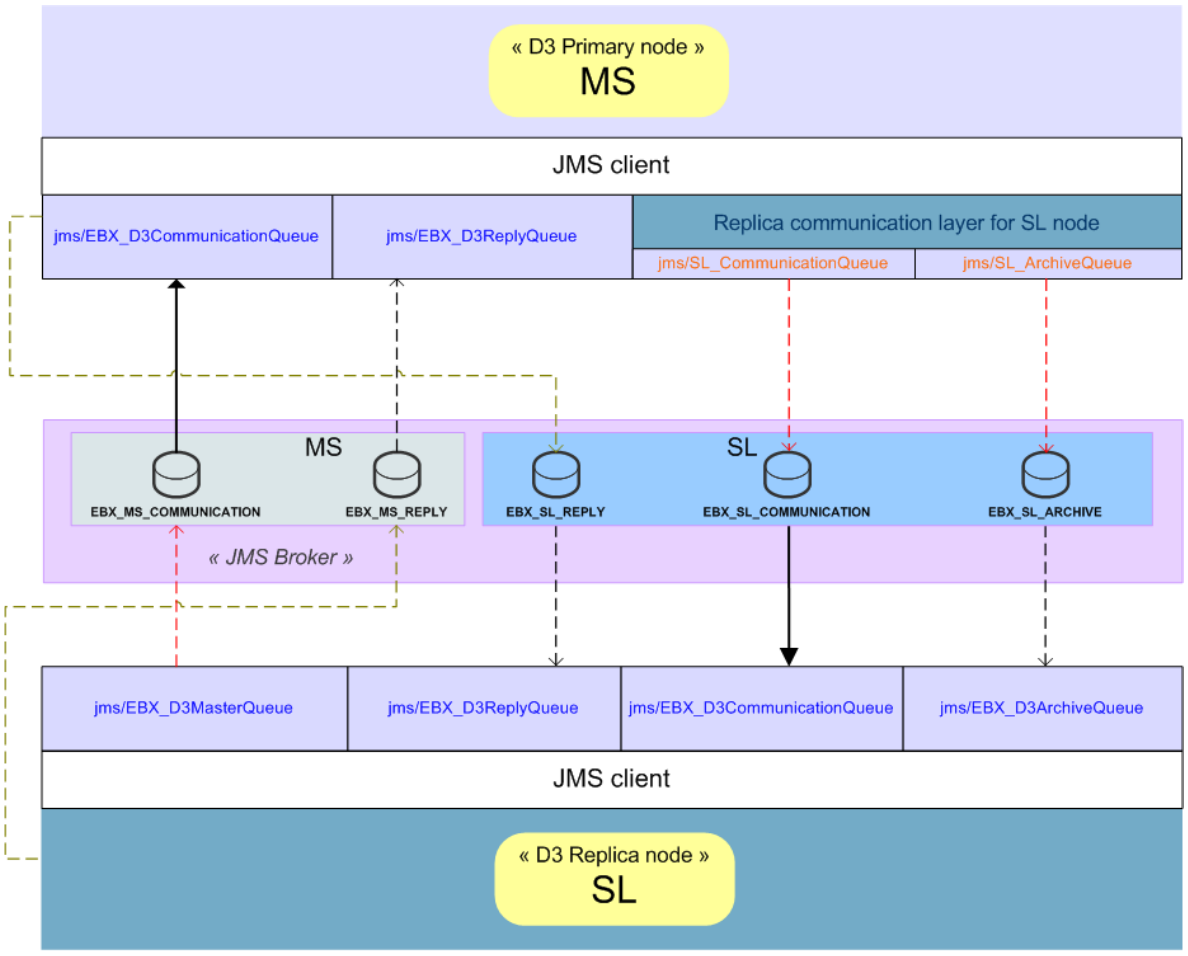
Examples of JMS configuration

	Shared queues	**Specific queues**
Primary-Replica nodes architecture	Between a primary node and two replica nodes with shared queues [p 436]	Between a primary node and a replica node with replica-specific queues [p 437]
Hub-Hub architecture	Between two hub nodes with shared queues [p 438]	Between two hub nodes with replica-specific queues [p 439]

Between a primary node and two replica nodes with shared queues


« D3 Primary Node »
MS
JMS client
jms/EBX_D3ReplyQueue
jms/EBX_D3ArchiveQueue
jms/EBX_D3CommunicationQueue
Message Queueing (D3 / JMS)
« D3 Replica Node »
SL1
JMS client
jms/EBX_D3MasterQueue
jms/EBX_D3ReplyQueue
jms/EBX_D3ArchiveQueue
jms/EBX_D3CommunicationQueue
EBX_D3_REPLY
EBX_D3_ARCHIVE
EBX_D3_COMMUNICATION
SHARED
« JMS Broker»
jms/EBX_D3MasterQueue
jms/EBX_D3ReplyQueue
jms/EBX_D3ArchiveQueue
jms/EBX_D3CommunicationQueue
JMS client
« D3 Replica Node »
SL2
Reserved resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')
Physical name (resource adapter / messaging)
Permanent consumer filtered by repository ID (using JMS type)
Volatile consumer (using JMS correlation Id)
Producer for SOAP request or archive (using queue from JNDI resource)
Producer for SOAP response (using queue from JMS reply to)

Between a primary node and a replica node with replica-specific queues

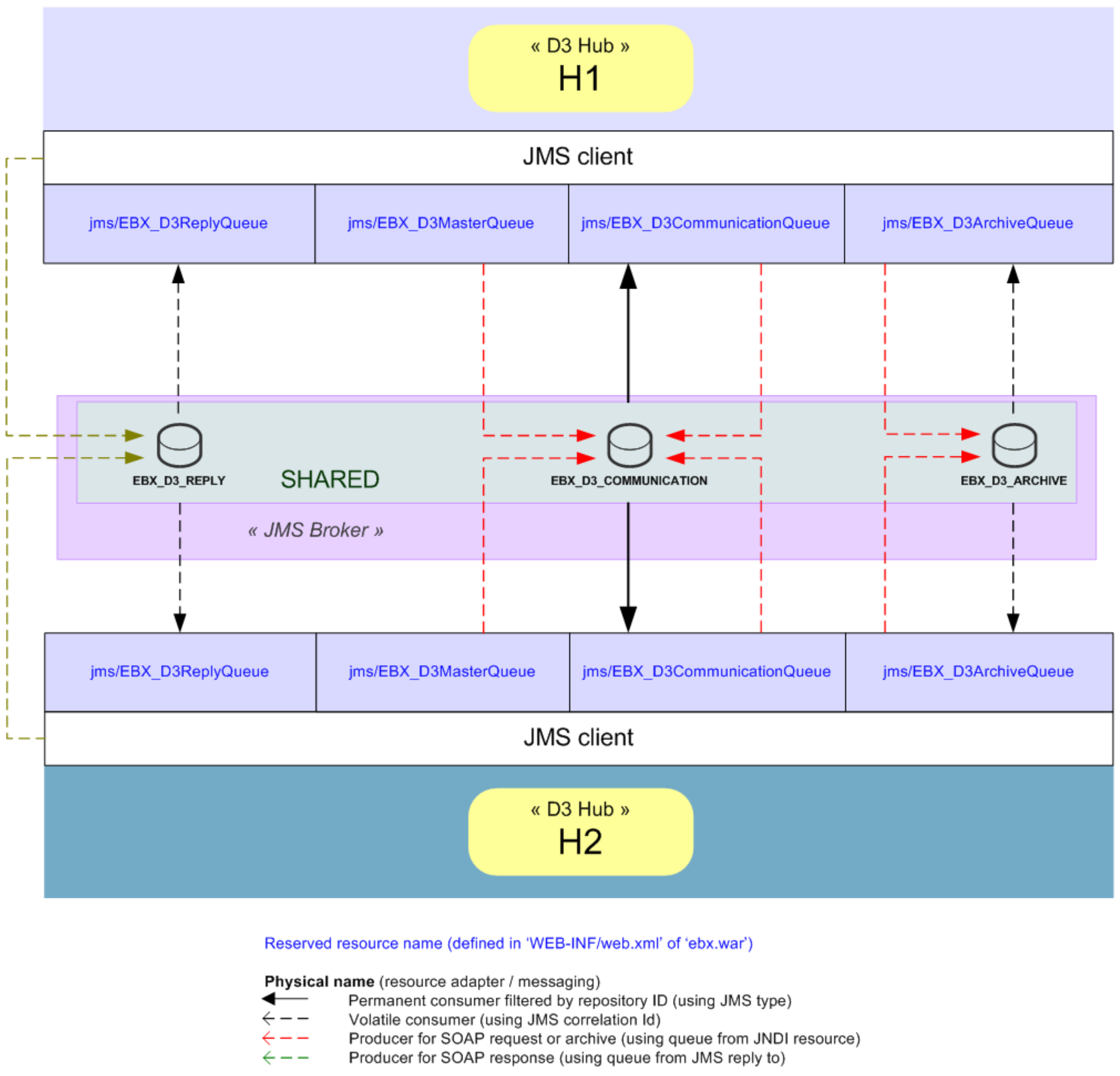


Reserved or custom resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

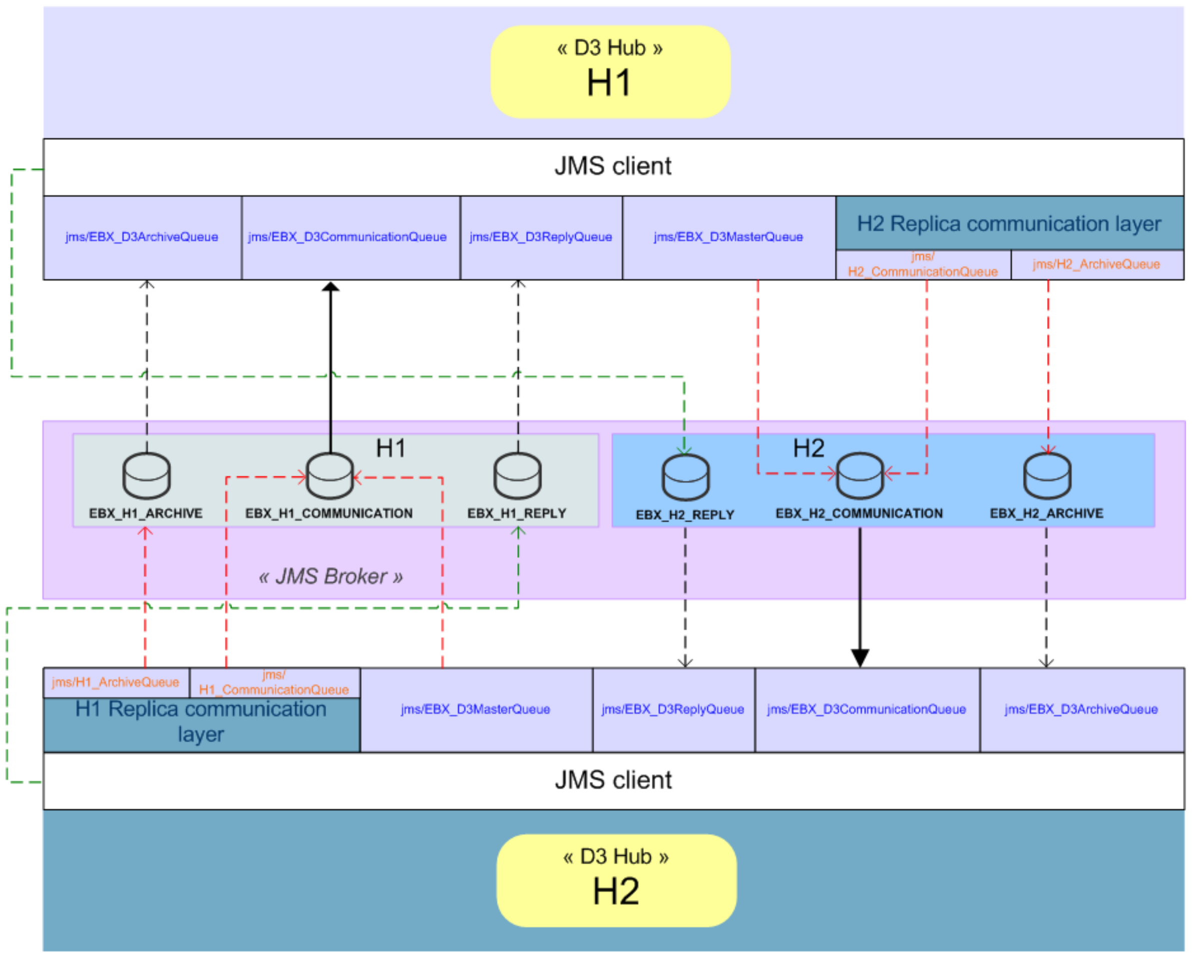
Physical name (resource adapter / messaging)

←	Permanent consumer filtered by repository ID (using JMS type)
← – –	Volatile consumer (using JMS correlation Id)
← – –	Producer for SOAP request or archive (using queue from JNDI resource)
← – –	Producer for SOAP response (using queue from JMS reply to)

Between two hub nodes with shared queues


« D3 Hub »
H1
JMS client
jms/EBX_D3ReplyQueue
jms/EBX_D3MasterQueue
jms/EBX_D3CommunicationQueue
jms/EBX_D3ArchiveQueue
EBX_D3_REPLY
SHARED
EBX_D3_COMMUNICATION
EBX_D3_ARCHIVE
« JMS Broker »
jms/EBX_D3ReplyQueue
jms/EBX_D3MasterQueue
jms/EBX_D3CommunicationQueue
jms/EBX_D3ArchiveQueue
JMS client
« D3 Hub »
H2
Reserved resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')
Physical name (resource adapter / messaging)
Permanent consumer filtered by repository ID (using JMS type)
Volatile consumer (using JMS correlation Id)
Producer for SOAP request or archive (using queue from JNDI resource)
Producer for SOAP response (using queue from JMS reply to)

Between two hub nodes with replica-specific queues



Reserved or custom resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

Physical name (resource adapter / messaging)

←——	Permanent consumer filtered by repository ID (using JMS type)
← – –	Volatile consumer (using JMS correlation Id)
← – –	Producer for SOAP request or archive (using queue from JNDI resource)
← – –	Producer for SOAP response (using queue from JMS reply to)

CHAPTER **74**

D3 administration

This chapter contains the following topics:

74.1 Quick start

This section introduces the configuration of a basic D3 architecture with two TIBCO EBX instances. Before starting, please check that each instance can work properly with its own repository.

> **Note**
>
> Deploy EBX on two different web application containers. If both instances are running on the same host, ensure that all communication TCP ports are distinct.

Declare an existing dataspace on the primary node

The objective is to configure and broadcast an existing dataspace from a *primary* node.

This configuration is performed on the entire D3 infrastructure (primary [p 427] and replica [p 427] nodes included).

Update the `ebx.properties`*primary* node configuration file with:

1. Define D3 mode as `primary` in key `ebx.d3.mode`.

> **Note**
>
> The *primary* node can be started after the configuration.

After authenticating as a built-in administrator, navigate within the administration tab:

1. Prerequisite: Check that the node is configured as a *primary* node (in the 'Actions' menu use 'System information' and check 'D3 mode').
2. Open the '[D3] Primary configuration' administration feature.
3. Add the dataspace to be broadcast to the 'Delivery dataspaces' table, and declare the allowed profile.
4. Add the delivery profile [p 427] to the 'Delivery profiles' table (it must correspond to a logical name) and declare the delivery mode. Possible values are: cluster mode [p 427] or federation mode [p 427].

5. Map the delivery dataspace with the delivery profile into the 'Delivery mapping' table.

> **Note**
>
> The *primary* node is now ready for the replica node(s) registration on the delivery profile.
>
> Check that the D3 broadcast menu appears in the 'Actions' menu of the dataspace or one of its snapshots.

Configure replica node for registration

The objective is to configure and register the *replica* node based on a delivery profile and communications settings.

Update the `ebx.properties` replica node configuration file with:

1. Define D3 mode as `replica` in key `ebx.d3.mode`.
2. Define the delivery profile [p 427] set on the *primary* node in key `ebx.d3.delivery.profiles` (delivery profiles must be separated by a comma and a space).
3. Define the *primary* node user authentication (must have the built-in administrator profile) for node communications in `ebx.d3.master.username` and `ebx.d3.master.password`.
4. Define HTTP/TCP protocols [p 444] for *primary* node communication, by setting a value for the property key `ebx.d3.master.url`

   (for example `http://localhost:8080/ebx-dataservices/connector`).
5. Define the *replica* node user authentication (must have the built-in administrator profile) for node communications in `ebx.d3.slave.username` and `ebx.d3.slave.password`.
6. Define HTTP/TCP protocols [p 444] for *replica* node communication, by setting a value for the property key `ebx.d3.slave.url`

   (for example `http://localhost:8090/ebx-dataservices/connector`).

> **Note**
>
> The *replica* node can be started after the configuration.

After authenticating as a built-in administrator, navigate inside the administration tab:

1. Prerequisite: Check that the node is configured as the *replica* node (in the 'Actions' menu use 'System information' and check 'D3 mode').
2. Open the '[D3] Replica configuration' administration feature.
3. Check the information on the 'Primary information' screen: No field should have the 'N/A' value.

> **Note**
>
> Please check that the model is available before broadcast (from data model assistant, it must be published).
>
> The *replica* node is then ready for broadcast.

74.2 **Configuring D3 nodes**

Runtime configuration of primary and hub nodes through the user interface

The declaration of delivery dataspaces and delivery profiles is done by selecting the '[D3] Primary configuration' feature from the 'Administration' area, where you will find the following tables:

Delivery dataspaces	Declarations of the dataspaces that can be broadcast.
Delivery profiles	Profiles to which replica nodes can subscribe. The delivery mode must be defined for each delivery profile.
Delivery mapping	The association between delivery dataspaces and delivery profiles.

> **Note**
>
> The tables above are read-only while some broadcasts are pending or in progress.

Configuring primary, hub and replica nodes

This section details how to configure a node in its EBX main configuration file.

See also*Overview* *[p 345]*

Primary node

In order to act as a *primary* node, an instance of EBX must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=master` node:

```
##################################################################
## D3 configuration
##################################################################
##################################################################
# Configuration for master, hub and slave
##################################################################
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=master
```

See also*primary node* *[p 427]*

Hub node

In order to act as a *hub* node (combination of primary and replica node configurations), an instance of EBX must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=hub` node:

```
##################################################################
## D3 configuration
##################################################################
##################################################################
# Configuration for master, hub and slave
```

```
####################################################################
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=hub

####################################################################
# Configuration dedicated to hub or slave
####################################################################
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

See also*hub node* *[p 428]*

Replica node

In order to act as a *replica* node, an instance of EBX must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=slave` node:

```
####################################################################
## D3 configuration
####################################################################
####################################################################
# Configuration for master, hub and slave
####################################################################
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=slave

####################################################################
# Configuration dedicated to hub or slave
####################################################################
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

See also*replica node* *[p 427]*

Configuring the network protocol of a node

This section details how to configure the network protocol of a node in its EBX main configuration file.

See also*Overview* *[p 345]*

HTTP(S) and socket TCP protocols

Sample configuration for `ebx.d3.mode=hub` or `ebx.d3.mode=slave` node with HTTP(S) network protocol:

```
####################################################################
```

```
# HTTP(S) and TCP socket configuration for D3 hub and slave
##################################################################
# URL to access the data services connector of the master
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: [protocol]://[master_host]:[master_port]/ebx-dataservices/connector
# Where the possible values of 'protocol' are 'http' or 'https'.
ebx.d3.master.url=

# URL to access the data services connector of the slave
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: [protocol]://[slave_host]:[slave_port]/ebx-dataservices/connector
# Where the possible values of 'protocol' are 'http' or 'https'.
ebx.d3.slave.url=

# Minimum port to use to transfer archives on TCP mode.
# Must be a positive integer above zero and below 65535.
# If not set, a random port will be used.
#ebx.d3.slave.socket.range.min=

# Max port to use on TCP mode to transfer archives.
# Must be a positive integer above ebx.d3.slave.socket.range.min and below 65535.
# Mandatory if ebx.d3.slave.socket.range.min is set.
#ebx.d3.slave.socket.range.max=
```

JMS protocol

If JMS is activated, the following properties can be defined in order to enable JMS functionalities for a D3 node.

Sample configuration for all D3 nodes with JMS network protocol:

```
##################################################################
## JMS configuration for D3
##################################################################
# Taken into account only if Data Services JMS is configured properly
##################################################################
# Configuration for master, hub and slave
##################################################################
# Default is false, activate JMS for D3
## If activated, the deployer must ensure that the entries
## 'jms/EBX_D3ReplyQueue', 'jms/EBX_D3ArchiveQueue' and 'jms/EBX_D3CommunicationQueue'
## are bound in the operational environment of the application server.
## On slave or hub mode, the entry 'jms/EBX_D3MasterQueue' must also be bound.
ebx.jms.d3.activate=false

# Change the default timeout when using reply queue.
# Must be a positive integer that does not exceed 3600000.
# Default is 10000 milliseconds.
#ebx.jms.d3.reply.timeout=10000

# Time-to-live message value expressed in milliseconds.
# This value will be set on each message header 'JMSExpiration' that defines the
# countdown before the message deletion managed by the JMS broker.
# Must be a positive integer equal to 0 or above the value of 'ebx.jms.d3.reply.timeout'.
# The value 0 means that the message does not expire.
# Default is 3600000 (one hour).
#ebx.jms.d3.expiration=3600000

# Archive maximum size in KB for the JMS body message. If exceeds, the message
# is transferred into several sequences messages in a same group, where each one does
# not exceed the maximum size defined.
# Must be a positive integer equals to 0 or above 100.
# Default is 0 that corresponds to unbounded.
#ebx.jms.d3.archiveMaxSizeInKB=

##################################################################
# Configuration dedicated to hub or slave
##################################################################
# Master repository ID, used to set a message filter for the concerned master when sending JMS message
# Mandatory property if ebx.jms.d3.activate=true and if ebx.d3.mode=hub or ebx.d3.mode=slave
#ebx.jms.d3.master.repositoryId=
```

See also *JMS for distributed data delivery (D3)* *[p 433]*

Services on primary nodes

Services to manage a primary node are available in the 'Administration' area of the replica node under '[D3] Primary node configuration' and also in the 'Delivery dataspaces' and 'Registered replica nodes' tables. The services are:

Relaunch replays	Immediately relaunch all replays for waiting federation deliveries.
Delete replica node delivery dataspace	Delete the delivery dataspace on chosen replica nodes and/ or unregister it from the configuration of the D3 primary node. To access the service, select a delivery dataspace from the 'Delivery dataspaces' table on the primary node, then launch the wizard.
Fully resynchronize	Broadcast the full content of the last broadcast snapshot to the registered replica nodes.
Subscribe a replica node	Subscribe a set of selected replica nodes.
Deactivate replica nodes	Remove the selected replica nodes from the broadcast scope and switch their states to 'Unavailable'. **Note** The "in progress" broadcast contexts are rolled back.
Unregister replica nodes	Disconnects the selected replica nodes from the primary node. **Note** The "in progress" broadcast contexts are rolled back.

Note

The primary node services above are hidden while some broadcasts are pending or in progress.

Services on replica nodes

Services are available in the 'Administration' area under *[D3] Configuration of replica* node to manage its subscription to the primary node and perform other actions:

Register replica node	Re-subscribes the replica node to the primary node if it has been unregistered.
Unregister replica node	Disconnects the replica node from the primary node. **Note** The "in progress" broadcast contexts are rolled back.
Close and delete snapshots	Clean up a replica node delivery dataspace. To access the service, select a delivery dataspace from the 'Delivery dataspaces' table on the replica node, then follow the wizard to close and delete snapshots based on their creation dates. **Note:** The last broadcast snapshot is automatically excluded from the selection.

74.3 Supervision

The last broadcast snapshot is highlighted in the snapshot table of the dataspace, it is represented by an icon displayed in the first column.

Primary node management console

Several tables make up the management console of the primary node, located in the 'Administration' area of the primary node, under '[D3] Primary node configuration'. They are as follows:

Registered replica nodes	Replica nodes registered with the primary node. From this table, several services are available on each record.
Broadcast history	History of broadcast operations that have taken place.
Replica node registration log	History of initialization operations that have taken place.
Detailed history	History of archive deliveries that have taken place. The list of associated delivery archives can be accessed from the tables 'Broadcast history' and 'Initialization history' using selection nodes.

Primary node supervision services

Available in the 'Administration' area of the primary node under '[D3] Primary node configuration'. The services are as follows:

Check replica node information	Lists the replica nodes and related information, such as the replica node's state, associated delivery profiles, and delivered snapshots.
Clear history content	Deletes all records in all history tables, such as 'Broadcast history', 'Replica node registration log' and 'Detailed history'.

Replica node monitoring through the Java API

A replica node monitoring class can be created to implement actions that are triggered when the replica node's status switches to either 'Available' or 'Unavailable'. To do so, it must implement the `NodeMonitoring` interface. This class must be outside of any EBX module and accessible from the class-loader of 'ebx.jar' and its full class name must be specified under '[D3] Replica node configuration'.

See also *`NodeMonitoring`* API

Primary node notification

A D3 administrator can set up mail notifications to receive broadcast events:

- On broadcast failure,
- On federation broadcast, if replays exceed a given threshold.

The mail contains a table of events with optional links to further details.

To enable notifications, open the '[D3] Primary node configuration' dataspace from the 'Administration' area and configure the 'Notifications' group under 'Global configuration'.

The 'From email' and 'URL definition' options should also be configured by using the 'Email configuration' link.

Log supervision

The technical supervision can be done through the log category 'ebx.d3', declared in the EBX main configuration file. For example:

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

See also *Configuring the EBX logs* [p 351]

Temporary files

Some temporary files, such as exchanged archives, SOAP messages, broadcast queue, (...), are created and written to the EBX temporary directory. This location is defined in the EBX main configuration file:

```
#################################################
```

```
## Directories for temporary resources.
##################################################
# When set, allows specifying a directory for temporary files different from java.io.tmpdir.
# Default value is java.io.tmpdir
ebx.temp.directory = \\${java.io.tmpdir}

# Allows specifying the directory containing temporary files for cache.
# If unset, the used directory is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.cache.directory = ${ebx.temp.directory}/ebx.platform

# When set, allows specifying the directory containing temporary files for import.
# If unset, the used directory is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.import.directory = ${ebx.temp.directory}/ebx.platform
```

Security Guide

CHAPTER 75

Security Best Practices

Here is a list of best practices that are considered useful to enforce a good security level for the EBX setup. These best practices apply to EBX and to other environments, their configuration, protocols and policies. These are best practices in general, and may not be relevant to your particular infrastructure and security policy.

This chapter contains the following topics:

75.1 Encryption algorithms

Web Server or Application Server may specify encryption algorithms when setting HTTPS parameters. Some recommendations on these algorithms are provided in section HTTPS [p 452]. Password and fields having osd:password as a type are storing hash of their value with SHA_512 as algorithm. That includes the password of users of the default directory.

75.2 HTTPS

Using HTTPS for communication with clients (GUI and REST or SOAP) is recommended. All HTTP traffic should be redirected to HTTPS.

A secure cipher suite and protocols should be used whenever possible. This applies, for example, to Web Servers, Application Servers, and jdbc connections.

TLS v1.2 should be the main protocol because it's the only version that offers modern authenticated encryption (also known as AEAD).

Several obsolete cryptographic primitives must be avoided:

- Anonymous Diffie-Hellman (ADH) suites do not provide authentication,

- NULL cipher suites provide no encryption,
- Export cipher suites are insecure when negotiated in a connection, but they can also be used against a server that prefers stronger suites (the FREAK attack),
- Suites with weak ciphers (typically of 40 and 56 bits) use encryption that can easily be broken,
- RC4 is insecure,
- 3DES is slow and weak,

On the other hand, getting too restrictive on allowed cyphers may prevent some clients to connect as they may not be able to negotiate the HTTPS connection.

The following configuration is compatible with browsers supported by EBX.

- Cipher suites: ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256
- Versions: TLSv1.2

75.3 Installation

Deployed components as Web Server and Application Server should be installed using a non-root or unprivileged user, and following the principle of least privilege whenever possible. For example, only necessary ports and protocols should be opened.

75.4 Web Server

If you have to expose web applications on the Internet, it's a good practice to protect them with a Web Server in a demilitarized zone while EBX and the database server may be in a production zone. Here are some best practices for the configuration.

The secure cipher suite and protocols should be set according to the above section HTTPS [p 452].

It is also a best practice not to use the default configuration, and to remove any banner that may also expose the version and type of web server.

For example, on Apache2, to remove the banner (default page returned at the root), just remove the folder `/var/www/html`.

Also, on Apache2, to remove headers identifying the Web Server, the value of `ServerTokens` and `ServerSignature` from the file `security.conf` should have the following values:

```
# ServerTokens
# This directive configures what you return as the Server HTTP response
# Header. The default is 'Full' which sends information about the OS-Type
# and compiled in modules.
# Set to one of:  Full | OS | Minimal | Minor | Major | Prod
# where Full conveys the most information, and Prod the least.
ServerTokens Prod

# Optionally add a line containing the server version and virtual host
# name to server-generated pages (internal error documents, FTP directory
# listings, mod_status and mod_info output etc., but not CGI generated
# documents or custom error documents).
# Set to "EMail" to also include a mailto: link to the ServerAdmin.
# Set to one of:  On | Off | EMail
ServerSignature Off
```

The Web Server is the recommended way for setting restrictions with HTTP security headers. Be aware that headers related to the origin will impact authorized URLs for all resources returned by EBX. That includes the content of fields of the URL type (example: image of avatar).

Here is a list of security headers and how to set them for EBX. First, EBX should be configured to not set any HTTP security headers. To do so, the property `ebx.security.headers.activated` must be set to 'false'.

X-XSS-Protection

The `x-xss-protection` header is designed to enable the cross-site scripting (XSS) filter built into modern web browsers. Here is what the header should look like.

`x-xss-protection: 1; mode=block`

For version 5.9.4, if the property `ebx.security.headers.activated` is not set or set to `true`, the security header must also be unset beforehand. For version 5.9.4, if the property `ebx.security.headers.activated` is set to `false`, the security header does not need to be unset, so ignore the first line in following snippets. For previous versions, the security header must be unset beforehand.

Enable in Nginx

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

Enable in Apache2

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

x-Frame-Options

The `x-frame-options` header provides clickjacking protection by not allowing iframes to load on the site. Be aware, this may not be compatible with your configuration if EBX is integrated through frames for example. Here is what the header should look like:

`x-frame-options: SAMEORIGIN`

Enable in Nginx

```
 add_header x-frame-options "SAMEORIGIN" always;
```

Enable in Apache2

```
 header always sets x-frame-options "SAMEORIGIN"
```

X-Content-Type-Options

The `x-content-type-options` header prevents Internet Explorer and Google Chrome from sniffing a response away from the declared content-type. This helps reduce the danger of drive-by downloads and helps treat the content properly. Here is what the header looks like.

`x-content-type-options: nosniff`

Enable in Nginx

```
 add_header X-Content-Type-Options "nosniff" always;
```

Enable in Apache2

```
 header always sets X-Content-Type-Options "nosniff"
```

Strict-Transport-Security

The `strict-transport-security` header is a security enhancement that restricts web browsers to access web servers solely over HTTPS. This ensures the connection cannot be established through an insecure HTTP connection which could be vulnerable to attacks. Here is what the header should look like:

```
strict-transport-security: max-age=31536000; includeSubDomains
```

Enable in Nginx

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

Enable in Apache2

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

Content-Security-Policy

The `content-security-policy` HTTP header provides an additional layer of security. This policy helps prevent attacks such as Cross Site Scripting (XSS) and other code injection attacks by defining content sources which are approved and thus allowing the browser to load them. Here is what the header shuould look like. Make sure to adapt it with your domain name (`server.company.com` in the example).

```
content-security-policy: default-src 'self'; font-src * data: server.company.com; img-
src * data: server.company.com; script-src * 'unsafe-inline' 'unsafe-eval'; style-src
 * 'unsafe-inline';
```

Enable in Nginx

```
add_header Content-Security-Policy "default-src 'self'; font-src * data:
server.company.com; img-src * data: server.company.com; script-src * 'unsafe-inline'
'unsafe-eval'; style-src * 'unsafe-inline';" always;
```

Enable in Apache2

```
header always 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';"
```

Referrer-Policy

The Referrer-Policy HTTP header governs which referrer information should be included with requests made. The Referrer-Policy tells the web browser how to handle referrer information that is sent when a user clicks on a link that leads to another page. Here is what it should look like:

```
Referrer-Policy: strict-origin
```

Enable in Nginx

```
add_header Referrer-Policy: "strict-origin" always;
```

Enable in Apache2

```
header always sets Referrer-Policy "strict-origin"
```

75.5 Application Server

As for Web Servers, the same best practice applies: do not expose technical information on the Application Server. For example, for Tomcat, it is recommended to fill the attribute `server` of `connector` in `server.xml` with a generic value as `AppServer`.

```
<Connector port="8080" enableLookups="false" protocol="HTTP/1.1" useBodyEncodingForURI="true"
server="AppServer"/>
```

If the Application Server is exposed through HTTPS, the secure cipher suite and Protocols should be set according to the above section HTTPS [p 452].

If there is a Web Server, it is also recommended to use ports higher than 1024 and let the Web Server do proxy.

If there is no Web Server, security headers should be set by the Application Server as described above.

75.6 **Java**

It is recommended to follow the security best practices from Oracle. Last supported patches should also be applied as soon as they are available especially when they include security patches. Consider using the Server JRE for server systems, such as application servers or other long-running back-end processes. The Server JRE is the same as the regular JRE except that it does not contain the web-browser plugins.

75.7 **Database**

Databases should be encrypted at rest and in transit. If there is a private key for encryption, it should not be stored in the same location as the data files. Regarding the JDBC connection, consider configuring the JDBC driver to use SSL/TLS. Contact your database administrator for detailed instructions. You should always use the last supported version or RDBMS including drivers.

75.8 **User directory and Administration rights**

For production and test platforms, EBX must be integrated with a custom directory [p 402] to enforce the password policy of your company. The default directory can be used only for development platforms.

According to the Separation of Duties best practice, administrators can manage users and grant access but should not have any functional rights.

Developer Guide

Introduction

CHAPTER 76

Packaging TIBCO EBX modules

An EBX module is a standard Java EE web application, packaging various resources such as XML Schema documents, Java classes and static resources.

Since EBX modules are web applications they benefit from features such as class-loading isolation, WAR or EAR packaging, and Web resources exposure.

This chapter contains the following topics:

76.1 Module structure

An EBX module contains the following files:

/WEB-INF/ebx/module.xml	This mandatory document defines the main properties and services of the module. See Module declaration [p 460].
/WEB-INF/web.xml	This is the standard Java EE deployment descriptor. It can perform the registration of the EBX module when the application server is launched. See Module registration [p 460].
/META-INF/MANIFEST.MF	Optional. If present, EBX reports the 'Implementation-Title' and 'Implementation-Version' values to *Administration > Technical configuration > Modules and data models*.
/www/	This optional directory contains all packaged resources, which are accessible via public URL. See Packaged resources [p 462].

Required files for Oracle WebLogic server:

`/WEB-INF/weblogic.xml`	WebLogic deployment descriptor file which activates the `prefer-web-inf-classes` policy, such as the following: `<?xml version="1.0" encoding="UTF-8"?>` `<weblogic-web-app xmlns="http://xmlns.oracle.com/weblogic/weblogic-web-app">` `<container-descriptor>` `<prefer-web-inf-classes>true</prefer-web-inf-classes>` `</container-descriptor>` `</weblogic-web-app>` See weblogic.xml Deployment Descriptor Elements for more information.

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app xmlns="http://xmlns.oracle.com/weblogic/
weblogic-web-app">
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
  </container-descriptor>
</weblogic-web-app>
```

76.2 Module declaration

A module is declared using the document `/WEB-INF/ebx/module.xml`. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:ebx-schemas:module_2.4"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:ebx-schemas:module_2.4 http://schema.orchestranetworks.com/module_2.4.xsd">
 <name>moduleTest</name>
</module>
```

See the associated schema for documentation about each property. The main properties are as follows:

Element	Description	Required
`name`	Defines the unique identifier of the module in the server instance. The module name usually corresponds to the name of the web application (the name of its directory).	Yes.
`publicPath`	Defines a path other than the module's name identifying the web application in public URLs. This path is added to the URL of external resources of the module when computing absolute URLs. If this field is not defined, the public path is the module's `name`, defined above.	No.
`services`	Declares user services using the legacy API. See *Declaration and configuration* of legacy user services. From the version 5.8.0, it is strongly advised to use the new user services [p 563].	No.
`beans`	Declares reusable Java bean components. See the workflow package [p 549].	No.
`ajaxComponents`	Declares Ajax components. See **Declaring an Ajax component in a module** `UIAjaxComponent.declareInModule`API in the Java API.	No.

76.3 Module registration

In order to be identifiable by EBX, a module must be registered at runtime when the application server is launched. For a web application, every EBX module must:

- contain a Java class with the annotation `@WebListener` extending the class `ModuleRegistrationListener`API.

> **Attention**
>
> When using the `@WebListener` annotation, ensure that the application server is configured to activate the servlet 3.0 annotation scanning for the web application. See `JSR 315`: JavaTM Servlet 3.0 Specification for more information.

or:

- contain a Servlet extending the class `ModuleRegistrationServlet`API;
- make a standard declaration of this servlet in the deployment descriptor `/WEB-INF/web.xml`;
- ensure that this servlet will be registered at server startup by adding the following standard element to the deployment descriptor: `<load-on-startup>1</load-on-startup>`.

Additional recommendations and information:

- The method `handleRepositoryStartup` in `ModuleRegistrationServlet`API allows setting the logger associated with the module and defining additional behavior such as common JavaScript and CSS resources.
- The specific class extending `ModuleRegistrationServlet` must be located in the web application (under `/WEB-INF/classes` or `/WEB-INF/lib`; due to the fact that this class is internally used as a hook to the application's class-loader, to load Java classes used by the data models associated with the module).
- The application server startup process is asynchronous and web applications / EBX modules are discovered dynamically. The EBX repository initialization depends on this process and will wait for the registration of all used modules up to an unlimited amount of time. As a consequence, if a used module is not deployed for any reason, it must be declared in the EBX main configuration file. For more information, see the property Declaring modules as undeployed [p 361].
- All module registrations and unregistrations are logged in the `log.kernel` category.
- If an exception occurs while loading a module, the cause is written in the application server log.
- Once the servlet is out of service, the module is unregistered and the data models and associated datasets become unavailable. Note that hot deployment/undeployment is not supported [p 311].

Deployment descriptor example

Here is an example of a Java EE deployment descriptor (`/WEB-INF/web.xml`):

```
<?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
                          http://java.sun.com/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>
```

Registration example

Here is an implementation example of the `ModuleRegistrationServlet`:

```
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_STYLESHEET_URL);
  aContext.addExternalJavaScriptResource(MyCompanyResources.COMMON_JAVASCRIPT_URL);

  aContext.addPackagedStyleSheetResource("myModule.css");
  aContext.addPackagedJavaScriptResource("myModule.js");

 }

 public void handleRepositoryShutdown()
 {
  // Release resources of the current module when the repository is shut down here
  ...
 }

  public void destroyBeforeUnregisterModule()
 {
  // Perform operations when this servlet is being taken out of service here
  ...
 }

}
```

76.4 Packaged resources

The packaged resources are files and documents that can be directly accessed from client browsers and can be managed and specified either as `osd:resource` fields or via the Java API. They have various types and can also be localized.

See also

*`ResourceType`*API

Type `osd:resource` *[p 484]*

Directory structure

The packaged resources must be located under the following directory structure:

1. On the first level, the directory `/www/` must be located at the root of the module (web application).
2. On the second level, the directory must specify the localization. It can be:
   - `common/` should contain all the resources to be used by default, either because they are locale-independent or as the default localization (in EBX, the default localization is `en`, namely English);
   - `{lang}/` when localization is required for the resources located underneath, with `{lang}` to be replaced by the actual locale code; it should correspond to the locales supported by EBX; for more information, see Configuring EBX localization [p 349].

3. On the third level, the directory must specify the resource type. It can be:
    - `jscripts/` for JavaScript resources;
    - `stylesheets/` for Cascading Style Sheet (CSS) resources;
    - `html/` for HTML resources;
    - `icons/` for icon typed resources;
    - `images/` for image typed resources.

Example

In this example, the image `logoWithText.jpg` is the only resource that is localized:

```
/www
 ├── common
 │   ├── images
 │   │   ├── myCompanyLogo.jpg
 │   │   └── logoWithText.jpg
 │   ├── jscripts
 │   │   └── myCompanyCommon.js
 │   └── stylesheets
 │       └── myCompanyCommon.css
 ├── de
 │   └── images
 │       └── logoWithText.jpg
 └── fr
     └── images
         └── logoWithText.jpg
```

CHAPTER 77

Mapping to Java

This chapter contains the following topics:

77.1 How to access data from Java?

Read access

Data can be read from various generic Java classes, mainly `Adaptation`^API and `ValueContext`^API. The getter methods for these classes return objects that are typed according to the mapping rules described in the section Mapping of data types [p 466].

Write access

Data updates must be performed in a well-managed context:

- In the context of a procedure execution, by calling the methods `setValue...` of the interface `ValueContextForUpdate`^API, or
- During the user input validation, by calling the method `setNewValue` of the class `ValueContextForInputValidation`^API.

Modification of mutable objects

According to the mapping that is described in the Mapping of data types [p 466] section, some accessed Java objects are mutable objects. These are instances of `List` ,`Date` or any JavaBean. Consequently, these objects can be locally modified by their own methods. However, such modifications will remain local to the returned object unless one of the above setters is invoked and the current transaction is successfully committed.

77.2 Concurrency and isolation levels

Highest isolation level

The highest isolation level in ANSI/ISO SQL is `SERIALIZABLE`. Three execution methods guarantee the `SERIALIZABLE` isolation level within the scope of a dataspace:

- If the client code is run inside a `Procedure`API container. This is the case for every update, for exports to XML, CSV or archive, and for data services.
- If the client code accesses a dataspace that has been explicitly locked. See `LockSpec`API.
- If the client code accesses data in a snapshot.

> **Note**
>
> For custom read-only transactions that run on a dataspace, it is recommended to use `ReadOnlyProcedure`API.

Default isolation level

If the client code is run outside the contexts that enable `SERIALIZABLE`, its isolation level depends on the persistence mode:

- In semantic mode, the default isolation level is `READ UNCOMMITTED`.
- In relational mode, the default isolation level is the database default isolation level.

See also*Overview of modes* [p 245]

Java access specificities

In a Java application, a record is represented by an instance of the `Adaptation` class. This object is initially linked to the corresponding persisted record. However, unless the client code is executed in a context that enables the SERIALIZABLE [p 466] isolation level, the object can become "disconnected" from the persisted record. If this occurs and concurrent updates have been performed, they will not be reflected in the `Adaptation` object.

Therefore, it is important for the client code to either be in a `SERIALIZABLE` context, or to regularly look up or refresh the `Adaptation` object.

See also

*`AdaptationHome.findAdaptationOrNull`*API

*`AdaptationTable.lookupAdaptationByPrimaryKey`*API

*`Adaptation.getUpToDateInstance`*API

77.3 Mapping of data types

This section describes how XML Schema type definitions and element declarations are mapped to Java types.

Simple data types

Basic rules for simple data types

Each XML Schema simple type corresponds to a Java class, the mapping is documented in the table XML Schema built-in simple types [p 480].

See also *`SchemaNode.createNewOccurrence`* API

Multiple cardinality on a simple element

If the attribute `maxOccurs` is greater than 1, the element is an aggregated list and the corresponding instance in Java is an instance of `java.util.List`.

Elements of the list are instances of the Java class that is determined from the mapping of the simple type (see previous section).

Complex data types

Complex type definitions without a class declaration

By default (no attribute `osd:class`), a terminal node of a complex type is instantiated using an internal class. This class provides a generic JavaBean implementation. However, if a custom client Java code has to access these values, it is recommended to use a custom JavaBean. To do so, use the `osd:class` declaration described in the next section.

It is also possible to transparently instantiate, read and modify the mapped Java object, with or without the attribute `osd:class`, by invoking the methods `SchemaNode.createNewOccurrence` API, `SchemaNode.executeRead` API and `SchemaNode.executeWrite` API.

Mapping of complex types to custom JavaBeans

It is possible to map an XML Schema complex type to a custom Java class. This is done by adding the attribute `osd:class` to the complex node definition. Unless the element has `xs:maxOccurs > 1`, you must also specify the attribute `osd:access` for the node to be considered a *terminal* node. If the element has `xs:maxOccurs > 1`, it is automatically considered to be terminal.

The custom Java class must conform to the JavaBean protocol. This means that each child of the complex type must correspond to a JavaBean property of the class. Additionally, each JavaBean property must be a read-write property, and its implementation must ensure that the value set by the setter method is returned, as-is, by the getter method. Contextual computations are not allowed in these methods.

Example

In this example, the Java class `com.carRental.Customer` must define the methods `getFirstName()` and `setFirstName(String)`.

A JavaBean can have a custom user interface within TIBCO EBX, by using a `UIBeanEditor` API.

```
<xs:element name="customer" osd:access="RW">
  <xs:complexType name="subscriber" osd:class="com.carRental.Customer">
 <xs:sequence>
   <xs:element name="firstName" type="xs:string"/>
   ...
 </xs:sequence>
  </xs:complexType>
</xs:element>
```

Multiple cardinality on a complex element

If the attribute `maxOccurs` is greater than 1, then the corresponding instance in Java is:

- An instance of `java.util.List` for an aggregated list, where every element in the list is an instance of the Java class determined by the mapping of simple types [p 480], or
- An instance of `AdaptationTable`API, if the property `osd:table` is specified.

77.4 Java bindings

Java bindings allow generating Java types that reflect the structure of the data model. The Java code generation can be done in the user interface. See Generating Java bindings [p 471].

Benefits

Ensuring the link between XML Schema structure and Java code provides a number of benefits:

- **Development assistance:** Auto-completion when typing an access path to parameters, if supported by your IDE.
- **Access code verification:** All accesses to parameters are verified at code compilation.
- **Impact verification:** Each modification of the data model impacts the code compilation state.
- **Cross-referencing:** By using the reference tools of your IDE, it is easy to verify where a parameter is used.

Consequently, it is strongly recommended to use Java bindings.

XML declaration

The specification of the Java types to be generated from the data model is included in the main schema.

Each binding element defines a generation target. It must be located at, in XPath notation, `xs:schema/xs:annotation/xs:appinfo/ebxbnd:binding`, where the prefix `ebxbnd` is a reference to the namespace identified by the URI `urn:ebx-schemas:binding_1.0`. Several binding elements can be defined if you have different generation targets.

The attribute `targetDirectory` of the element `ebxbnd:binding` defines the root directory used for Java type generation. Generally, it is the directory containing the project source code, `src`. A relative path is interpreted based on the current runtime directory of the VM, as opposed to the XML schema.

See bindings XML Schema.

XML bindings example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:ebxbnd="urn:ebx-schemas:binding_1.0">
 <xs:annotation>
  <xs:appinfo>
   <!-- The bindings define how this schema will be represented in Java.
   Several <binding> elements may be defined, one for each target. -->
   <ebxbnd:binding
    targetDirectory="../_ebx-demos/src-creditOnLineStruts-1.0/">
    <javaPathConstants typeName="com.creditonline.RulesPaths">
     <nodes root="/rules" prefix="" />
    </javaPathConstants>
    <javaPathConstants typeName="com.creditonline.StylesheetConstants">
     <nodes root="/stylesheet" prefix="" />
    </javaPathConstants>
   </ebxbnd:binding>
  </xs:appinfo>
 </xs:annotation>
 ...
```

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

Java constants can be defined for XML schema paths. To do so, generate one or more interfaces from a schema node, including the root node `/`. The example generates two Java path constant interfaces, one from the node `/rules` and the other from the node `/stylesheet` in the schema. Interface names are described by the element `javaPathConstants` with the attribute `typeName`. The associated node is described by the element `nodes` with the attribute `root`.

CHAPTER **78**

Tools for Java developers

TIBCO EBX provides Java developers with tools to facilitate use of the EBX API, as well as integration with development environments.

This chapter contains the following topics:

78.1 **Activating the development tools**

To activate the development tools, run EBX in *development mode*. This is specified in the EBX main configuration file EBX run mode [p 363] using the property `backend.mode=development`.

78.2 **Data model refresh tool**

When editing the data model directly as an XML Schema document without using the data-modeling tool provided by EBX, you can refresh it without restarting the application server.

In the 'Administration' area, select **Select** > **Technical configuration** > **Development tools** > **Refresh updated data models** (or **Refresh all data models**).

Attention

Since the operation is critical regarding data consistency, refreshing the data models acquires a global exclusive lock on the repository. This means that most other operations (data access and update, validation, etc.) will wait until the completion of the data model refresh.

78.3 **Generating Java bindings**

The Java types specified by Java bindings can be generated from a dataset or a data model, by selecting **Actions** > **Generate Java** in the navigation pane.

See also*Java bindings* *[p 468]*

78.4 **Path to a node**

The field 'Data path' is displayed in the documentation pane of a node. This field indicates the path to the node, which can be useful when writing XPath formulas.

> **Note**
>
> This field is always available to administrators.

78.5 **Web component link generator**

The 'Web component link generator' service is a user interface designed to create HTTP requests that call EBX web components. To launch this service, select **Actions > Web component link generator** in the navigation pane.

CHAPTER 79
Terminology changes

A new TIBCO EBX release can introduce new vocabulary for users. To preserve the backward compatibility, these terminology changes do not usually impact the API. Consequently, Java class names, method names, data services operation names, etc. still use the older version terminology. This chapter purpose is to facilitate the correspondence of the old term in the API to the new terms.

See also*Glossary* *[p 23]*

This chapter contains the following topics:

79.1 Terminology changes in version 5.9

New term	Term prior to version 5.9.0
D3 primary node	D3 master node
D3 replica node	D3 slave node

79.2 **Terminology changes in version 5.0**

The following table summarizes the mappings between the version 5.0.0 terminology and previous terminology:

New term	Term prior to version 5.0.0
Dataset	Adaptation instance
Child dataset	Child adaptation instance
Data model	Data model
Dataspace	Branch
Snapshot	Version
Dataspace or snapshot	Home
Data Workflow	Workflow instance
Workflow model	Workflow definition
Workflow publication	Workflow
Data services	Data services
Field	Attribute
Inherited field	Inherited attribute
Record	Record/occurrence
Validation rule	Constraint
Simple/advanced control	Simple/advanced constraint

Data model

CHAPTER **80**

Introduction

A data model is a structural definition of the data to be managed in the TIBCO EBX repository. Data models contribute to EBX's ability to guarantee the highest level of data consistency and to facilitate data management.

Specifically, the data model is a document that conforms to the XML Schema standard (W3C recommendation). Its main features are as follows:

- A rich library of well-defined simple data types [p 479], such as integer, boolean, decimal, date, time;
- The ability to define additional simple types [p 481] and complex types [p 481];
- The ability to define simple lists of items, called aggregated lists [p 490];
- Validation constraints [p 513] (facets), for example: enumerations, uniqueness constraints, minimum/maximum boundaries.

EBX also uses the extensibility features of XML Schema for other useful information, such as:

- Predefined types [p 482], for example: locale, resource, html;
- Definition of tables [p 493] and foreign key constraints [p 498];
- Mapping data in EBX to Java beans;
- Advanced validation constraints [p 513] (extended facets), such as dynamic enumerations;
- Extensive presentation information [p 531], such as labels, descriptions, and error messages.

> **Note**
>
> EBX supports a subset of the W3C recommendations, as some features are not relevant to Master Data Management.

This chapter contains the following topics:

80.1 Editing the data model

There are two different ways to define a data model:

- The data model can be defined using an XML Schema editor or through the data model assistant. The data model assistant has the advantage of being integrated into the EBX user interface, abstracting the verbose underlying XML. For more information, see Introduction to data models [p 34]. The data model assistant allows using features that are not documented to be used outside of the DMA; e.g. Toolbars and Widgets.
- By using an external XML Schema document editor.

80.2 References

For an introduction to XML Schema, see the W3Schools XML Schema Tutorial.

See also

XML Schema Part 0: Primer

XML Schema Part 1: Structures

XML Schema Part 2: Datatypes

80.3 Relationship between datasets and data models

Each root dataset is associated with a single data model. At the dataspace creation, an associated data model is selected, on which to base the dataset.

See also*Creating a dataset [p 111]*

80.4 Pre-requisite for XML Schemas

In order for an XML Schema to be accepted by EBX, it must include a global element declaration that includes the attribute `osd:access="--"`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
   <xs:import namespace="urn:ebx-schemas:common_1.0"
     schemaLocation="http://schema.orchestranetworks.com/common_1.0.xsd"/>
   <xs:element name="root" osd:access="--">
  ...
  </xs:element>
</xs:schema>
```

80.5 Conventions

By convention, namespaces are always defined as follows:

Prefix	Namespace
`xs:`	`http://www.w3.org/2001/XMLSchema`
`osd:`	`urn:ebx-schemas:common_1.0`
`fmt:`	`urn:ebx-schemas:format_1.0`
`usd:`	`urn:ebx-schemas:userServices_1.0`
`emd:`	`urn:ebx-schemas:entityMappings_1.0`

80.6 Schemas with reserved names

Several data models in EBX have reserved names.

All references to other data models (using the attribute `schemaLocation` for an import, include or redefine) that end with one of the following strings are reserved:

- common_1.0.xsd
- org_1.0.xsd
- coreModel_1.0.xsd
- session_1.0.xsd
- userServices_1.0.xsd
- entityMappings_1.0.xsd

These XSD files correspond to the schemas provided for the module `ebx-root-1.0`, at the path `/WEB-INF/ebx/schemas`. The attribute `schemaLocation` can reference the files at this location or a copy, if the file names are identical. This is useful if you want to avoid a module dependency on `ebx-root-1.0`.

For security reasons, EBX uses an internal definition for these schemas to prevent any modification.

CHAPTER 81

Data types

This chapter details the data types supported by TIBCO EBX.

See also*Tables and relationships* *[p 493]*

This chapter contains the following topics:

81.1 **XML Schema built-in simple types**

The table below lists all the simple types defined in XML Schema that are supported by EBX, along with their corresponding Java types.

XML Schema type	**Java class**	**Notes**
`xs:string`	`java.lang.String`	
`xs:boolean`	`java.lang.Boolean`	
`xs:decimal`	`java.math.BigDecimal`	A `totalDigits` facet with a value equal to 15 is added by default to decimal fields that are contained in a mapped table (relational, historized or replicated table). However, this facet can be overwritten with a greater value in the data model.
`xs:dateTime`	`java.util.Date`	
`xs:time`	`java.util.Date`	The date portion of the returned `Date` is always set to '1970/01/01'.
`xs:date`	`java.util.Date`	The time portion of the returned `Date` is always the beginning of the day, that is, '00:00:00'.
`xs:anyURI`	`java.net.URI`	
`xs:Name` (`xs:string` restriction)	`java.lang.String`	
`xs:int` (`xs:decimal` restriction)	`java.lang.Integer`	
`xs:integer` (`xs:decimal` restriction)	`java.lang.Integer`	This mapping does not comply with the XML Schema recommendation. Although the XML Schema specification states that `xs:integer` has no value space limitation, this value space is, in fact, restricted by the Java specifications of the `java.lang.Integer` object.

The mapping between XML Schema types and Java types are detailed in the section Mapping of data types [p 466].

81.2 **XML Schema named simple types**

Named simple types can be defined when designing a data model for redefining an existing built-in simple type. A *named simple type* can be reused in the data model.

Restrictions:

- In the data model, only the element `restriction` is allowed in a named simple type, and even then, only derivation by restriction is supported. Notably, the elements `list` and `union` are not supported.
- Facet definition is not cumulative. That is, if an element and its named type both define the same kind of facet, then the facet defined in the type is overridden by the local facet definition. However, this restriction does not apply to programmatic facets defined by the element `osd:constraint`. For `osd:constraint`, if an element and its named type both define a programmatic facet with different Java classes, the definition of these facets will be cumulative. Contrary to the XML Schema Specification, EBX is not strict regarding the definition of a facet of the same kind in an element and its named type. That is, the value of a same kind of facet defined in an element is not checked according to the one defined in the named type. However, in the case of static enumerations defined both in an element and its type, the local enumeration will be replaced by the intersection between these enumerations.
- It is not possible to define different types of enumerations on both an element and its named type. For instance, you cannot specify a static enumeration in an element and a dynamic enumeration in its named type.
- It is not possible to simultaneously define a pattern facet in both an element and its named type.

81.3 **XML Schema complex types**

Complex types can be defined when designing a data model. A *named complex type* can be reused in the data model.

Restrictions:

- In the data model, only the element `sequence` is allowed. Notably, attribute definition is not supported.
- Type extensions are not supported in the current version of EBX.

81.4 **Extended simple types defined by EBX**

EBX provides pre-defined simple data types:

XML Schema type	Java class
`osd:text` (`xs:string` restriction)	`java.lang.String`
`osd:html` (`xs:string` restriction)	`java.lang.String`
`osd:email` (`xs:string` restriction)	`java.lang.String`
`osd:password` (`xs:string` restriction)	`java.lang.String`
`osd:color` (`xs:string` restriction)	`java.lang.String`
`osd:resource` (`xs:anyURI` restriction)	internal class
`osd:locale` (`xs:string` restriction)	`java.util.Locale`
`osd:dataspaceKey` (`xs:string` restriction)	`java.lang.String`
`osd:datasetName` (`xs:string` restriction)	`java.lang.String`

The above types are defined by the internal schema `common-1.0.xsd`. They are defined as follows:

`osd:text`	This type represents textual information. For a basic `xs:string`, its default user interface in EBX consists of a dedicated editor with several lines for input and display.
	`<xs:simpleType name="text">`
	` <xs:restriction base="xs:string" />`
	`</xs:simpleType>`

`osd:html`	This represents a character string with HTML formatting. A WYSIWYG editor is provided in EBX.
	`<xs:simpleType name="html">`
	` <xs:restriction base="xs:string" />`
	`</xs:simpleType>`

`osd:email`	This represents an email address as specified by the RFC822 standard.
	`<xs:simpleType name="email">`
	` <xs:restriction base="xs:string" />`
	`</xs:simpleType>`

`osd:password`	This represents a hashed or encrypted password. A specific editor is provided in EBX.
	`<xs:element name="password" type="osd:password" />`
	The default editor performs a hash computation using the SHA-512 algorithm. This encryption function is also available from a Java client using the method `DirectoryDefault.encryptString`API.
	It is also possible for the default editor to use a different encryption mechanism by specifying a class that implements the interface `Encryption`API.
	`<xs:element name="password" type="osd:password">`
	` <xs:annotation>`
	`  <xs:appinfo>`
	`   <osd:uiBean class="com.orchestranetworks.ui.UIPassword">`
	`    <encryptionClass>package.EncryptionClassName</encryptionClass>`
	`   </osd:uiBean>`
	`  </xs:appinfo>`
	` </xs:annotation>`
	`</xs:element>`
	It is possible to specify some salt by referencing a path to another field, and by using a class the implements the interface `HashComputation`API.
	`<xs:element name="password" type="osd:password">`
	` <xs:annotation>`
	`  <xs:appinfo>`
	`   <osd:uiBean class="com.orchestranetworks.ui.UIPassword">`
	`    <encryptionClass>package.HashClassName</encryptionClass>`
	`    <saltPath>../login</saltPath>`
	`   </osd:uiBean>`
	`  </xs:appinfo>`
	` </xs:annotation>`
	`</xs:element>`

osd:locale

This represents a geographical, political or cultural location. The locale type is translated into Java by the class `java.util.Locale`.

```
<xs:simpleType name="locale">
 <xs:restriction base="xs:string">
  <xs:enumeration value="ar" osd:label="Arabic" />
  <xs:enumeration value="ar_AE" osd:label="Arabic (United Arab
 Emirates)" />
  <xs:enumeration value="ar_BH" osd:label="Arabic (Bahrain)" />
  <xs:enumeration value="ar_DZ" osd:label="Arabic (Algeria)" />
  <xs:enumeration value="ar_EG" osd:label="Arabic (Egypt)" />
  <xs:enumeration value="ar_IQ" osd:label="Arabic (Iraq)" />
   ...
  <xs:enumeration value="vi_VN" osd:label="Vietnamese (Vietnam)" /
>
  <xs:enumeration value="zh" osd:label="Chinese" />
  <xs:enumeration value="zh_CN" osd:label="Chinese (China)" />
  <xs:enumeration value="zh_HK" osd:label="Chinese (Hong Kong)" />
  <xs:enumeration value="zh_TW" osd:label="Chinese (Taiwan)" />
 </xs:restriction>
</xs:simpleType>
```

osd:color

This represents a character string with hexadecimal RGB color formatting. A color picker `UIComponent` is provided in EBX.

```
<xs:simpleType name="color">
 <xs:restriction base="xs:string" />
</xs:simpleType>
```

osd:resource

This represents a resource packaged in a module. For more information, see Packaged resources [p 462]. This type requires the definition of the facet FacetOResource [p 518].

```
<xs:simpleType name="resource">
 <xs:restriction base="xs:anyURI" />
</xs:simpleType>
```

osd:dataspaceKey

This type represents a reference to a dataspace.

```
<xs:element name="dataspaceField" type="osd:dataspaceKey" />
```

A specific editor is provided in EBX that displays the dataspaces that can be referenced.

It is possible to specify the dataspaces that can be referenced using the element `osd:dataspaceSet` under `xs:annotation/xs:appInfo`. If the element `osd:dataspaceSet` is not defined, then by default, only open branches can be referenced.

```
<xs:element name="dataspaceField" type="osd:dataspaceKey">
 <xs:annotation>
  <xs:appinfo>
   <osd:dataspaceSet>
    <include>
     <pattern>a pattern</pattern>
     <type>all | branch | version</type>
     <includeDescendants>none | allDescendants |
 allBranchDescendants | allSnapshotDescendants | branchChildren |
 snapshotChildren</includeDescendants>
    </include>
    <exclude>
     <pattern>a pattern</pattern>
     <type>all | branch | version</type>
     <includeDescendants>none | allDescendants |
 allBranchDescendants | allSnapshotDescendants | branchChildren |
 snapshotChildren</includeDescendants>
    </include>
    <filter osd:class="com.foo.MyDataspaceFilter">
     <param1>...</param1>
     <param2>...</param2>
    </filter>
   </osd:dataspaceSet>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

- includes

  Specifies the dataspaces that can be referenced by this field. An `include` must at least be defined.

  `pattern:` Specifies a pattern that filters dataspaces. The pattern is checked against the name of the dataspaces. This property is mandatory.

  `type:` Specifies the type of dataspaces that can be referenced by this field. If not defined, this restriction is applied to branches. If `all` then branches and snapshots are included. If `branch` then only branches are included. If `snapshot` then only snapshots are included. If not set, this property is `branch` by default.

  `includeDescendants:` Specifies if children or descendants of the dataspaces that match the specified pattern are included in the set. If `none` then neither children nor descendants of the dataspaces that match the specified pattern are included. If `allDescendants` then all descendants of the dataspaces that match the specified pattern are included. If `allBranchDescendants` then all descendant branches of the dataspaces that match the specified pattern are included. If `allSnapshotDescendants` then all descendant snapshots of the dataspaces that match the specified pattern are included. If `directBranchChildren` then only direct branches of the dataspaces that match the specified pattern are included. If `directSnapshotChildren` then only direct snapshots of the dataspaces that match the specified pattern are included. If not set, this property is `none` by default.

- excludes

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

  `pattern:` Specifies a pattern that filters dataspaces. The pattern is checked against the name of the dataspaces. This property is mandatory.

  `type:` Specifies the type of dataspaces that can be referenced by this field. If not defined, this restriction is applied to branches. If `all` then branches and snapshots are excluded. If `branch` then only branches are excluded. If `snapshot` then only snapshots are excluded. If not set, this property is `branch` by default.

`includeDescendants:` Specifies if children or descendants of the datasets that match the specified pattern are excluded from the set. If `none` then neither children nor descendants of the dataspaces that match the specified pattern are excluded. If `allDescendants` then all descendants of the dataspaces that match the specified pattern are excluded. If `allBranchDescendants` then all descendant branches of the dataspaces that match the specified pattern are excluded. If `allSnapshotDescendants` then all descendant snapshots of the dataspaces that match the specified pattern are excluded. If `directBranchChildren` then only direct branches of the dataspaces that match the specified pattern are excluded. If `directSnapshotChildren` then only direct snapshots of the dataspaces that match the specified pattern are excluded. If not set, this property is `none` by default.

- `filter`

  Specifies a filter to accept or reject dataspaces in the context of a dataset or a record. This filter is only used in the dedicated input component that is associated to this field. That is, this filter is not used when validating this field. A specific constraint can be used to perform specific controls on this field.

  The attribute `osd:class` specifies a Java bean that implements the interface `DataspaceSetFilter`API.

It is also possible to customize validation messages and the control policy associated with this type using the element `validation` under `xs:annotation/xs:appInfo/osd:dataspaceSet`. See Facet validation message with severity [p 534] and Control policy [p 522] for more information.

`osd:datasetName`

This type represents a reference to a dataset.

```
<xs:element name="dataset" type="osd:datasetName" />
```

A specific editor provided in EBX displays the datasets that can be referenced.

It is also possible to specify the datasets that can be referenced using the element `osd:datasetSet` under `xs:annotation/xs:appInfo`:

```
<xs:element name="datasetField" type="osd:datasetName">
 <xs:annotation>
  <xs:appinfo>
   <osd:datasetSet>
    <branch>productsBranch</branch>
    <version>productsVersion</version>
    <dataspaceSelector>../dataspaceField</dataspaceSelector>
    <pattern>a pattern</pattern>
    <filter osd:class="com.foo.MyDatasetFilter">
     <param1>...</param1>
     <param2>...</param2>
    </filter>
   </osd:datasetSet>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

- `branch`

  Specifies the source branch. Only datasets contained in this branch will be able to be selected by a field of the type Dataset identifier (osd:datasetName).

- `version`

  Specifies the source snapshot. Only datasets contained in this snapshot will be able to be selected by a field of the type Dataset identifier (osd:datasetName).

- `dataspaceSelector`

  Specifies a field in the same data model that defines the dataspace containing the datasets that can be referenced. The specified field must be of type `xs:string` or `osd:dataspaceKey`. The value of this field must comply with the representation of a persistent identifier of a dataspace or snapshot. See `HomeKey.format`API for more information.

  The referred node must respect the restrictions existing for dynamic facets, see Dynamic constraints [p 516].

- `includes`

  Specifies the datasets that can be referenced by this field.

  `pattern`: Specifies a pattern that filters datasets. The pattern is checked against the name of the datasets. This property is mandatory.

  `includeDescendants`: Specifies if children or descendants of the datasets that match the specified pattern are included in the set. If `none` then neither children nor descendants of the datasets that match the specified pattern are excluded. If `directChildren` then only direct children of the datasets that match the specified pattern are excluded. If `allDescendants` then all descendants of the datasets that match the specified pattern are excluded. If not set, this property is `none` by default.

- `excludes`

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

`pattern:` Specifies a pattern that filters datasets. The pattern is checked against the name of the datasets. This property is mandatory.

`includeDescendants:` Specifies if children or descendants of the datasets that match the specified pattern are included in the set. If `none` then neither children nor descendants of the datasets that match the specified pattern are excluded. If `directChildren` then only direct children of the datasets that match the specified pattern are excluded. If `allDescendants` then all descendants of the datasets that match the specified pattern are excluded. If not set, this property is `none` by default.

- `filter`

  Specifies a filter to accept or reject datasets in the context of a dataset or record. This filter is only used in the dedicated input component that is associated to this field. That is, this filter is not used when validating this field. A specific constraint can be used to perform specific controls on this field.

  The attribute `osd:class` specifies a Java bean that implements the interface `DatasetSetFilter`API. A validation message is added to the associated field if an input dataspace reference does not match this filter.

One of the elements `branch`, `version` or `dataspaceSelector` must be defined.

It is also possible to customize validation messages and the control policy associated with this type using the element `validation` under `xs:annotation/xs:appInfo/osd:datasetSet`. See Facet validation message with severity [p 534] and Control policy [p 522] for more information.

81.5 **Complex types defined by EBX**

EBX provides pre-defined complex data types:

XML Schema type	Description
`osd:UDA`	**User Defined Attribute:** This type allows any user, according to their access rights, to define a value associated with an attribute defined in a dictionary called a UDA Catalog.
`osd:UDACatalog`	**Catalog of User Defined Attributes:** This type consists of a table in which attributes can be specified. This catalog is used by all `osd:UDA` elements declared in the same data model.

`osd:UDA`

A User Defined Attribute (UDA) supports both the `minOccurs` and `maxOccurs` attributes, as well as the attribute `osd:UDACatalogPath`, which specifies the path of the corresponding catalog.

```
<xs:element name="firstUDA" type="osd:UDA" minOccurs="0"
 maxOccurs="unbounded" osd:UDACatalogPath="//insuranceCatalog" />
<xs:element name="secondUDA" type="osd:UDA" minOccurs="1"
 maxOccurs="1"
 osd:UDACatalogPath="/root/userCatalog" />
<xs:element name="thirdUDA" type="osd:UDA" minOccurs="0"
 maxOccurs="1"
 osd:UDACatalogPath="//userCatalog" />
```

In the manager, when working with a UDA, the editor will adapt itself to the type of the selected attribute.

`osd:UDACatalog`

Internally, a catalog is represented as a table. The parameters `minOccurs` and `maxOccurs` must be specified.

Several catalogs can be defined in the same data model.

```
<xs:element name="insuranceCatalog" type="osd:UDACatalog"
 minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:documentation xml:lang="en-US">Insurance Catalog.</
xs:documentation>
  <xs:documentation xml:lang="fr-FR">Catalog assurance.</
xs:documentation>
 </xs:annotation>
</xs:element>
<xs:element name="userCatalog" type="osd:UDACatalog" minOccurs="0"
 maxOccurs="unbounded">
 <xs:annotation>
  <xs:documentation xml:lang="en-US">User catalog.</
xs:documentation>
  <xs:documentation xml:lang="fr-FR">Catalogue utilisateur.</
xs:documentation>
 </xs:annotation>
</xs:element>
```

Only the following types are available for creating new attributes:

- xs:string
- xs:boolean
- xs:decimal

- xs:dateTime
- xs:time
- xs:date
- xs:anyURI
- xs:Name
- xs:int
- osd:html
- osd:email
- osd:password
- osd:locale
- osd:text

Restrictions on User Defined Attributes and Catalogs

The following features are unsupported on UDA elements:

- Facets
- Functions using the `osd:function` property
- UI bean editors using the `osd:uiBean` property
- The `osd:checkNullInput` property
- History features
- Replication
- Inheritance features, using the `osd:inheritance` property

As UDA catalogs are internally considered to be tables, the restrictions that apply to tables also exist for `UDACatalog` elements.

81.6 **Aggregated lists**

In XML Schema, the maximum number of times an element can occur is determined by the value of the `maxOccurs` attribute in its declaration. If this value is strictly greater than 1 or is `unbounded`, the data can have multiple occurrences. If no `osd:table` declaration is included, this element is called an *aggregated list*. In Java, it is then represented as an instance of the class `java.util.List`.

The following is an example of an aggregated list that defines the pricing of a loan product, depending on the amount borrowed.

```
<xs:element name="pricing" minOccurs="0" maxOccurs="unbounded"
 osd:access="RW">
 <xs:annotation>
  <xs:documentation>
   <osd:label>Pricing</osd:label>
   <osd:description>Pricing grid </osd:description>
  </xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element name="amount" type="xs:int">
    <xs:annotation>
     <xs:documentation>
      <osd:label>Amount borrowed</osd:label>
     </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="monthly" type="xs:int">
    <xs:annotation>
     <xs:documentation>
      <osd:label>Monthly payment </osd:label>
     </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="cost" type="xs:int">
    <xs:annotation>
     <xs:documentation>
      <osd:label>Cost</osd:label>
     </xs:documentation>
    </xs:annotation>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Aggregated lists have a dedicated editor in EBX. This editor allows you to add or to delete occurrences.

Attention

The addition of an `osd:table` declaration to an element with `maxOccurs > 1` is a very important consideration that must be taken into account during the design process. An aggregated list is severely limited with respect to the many features that are supported by tables. Some features unsupported on aggregated lists that are supported on tables are:

- Performance and memory optimization;
- Lookups, filters and searches;
- Sorting, view and display in hierarchies;
- Identity constraints (primary keys and uniqueness constraints);
- Detailed permissions for creation, modification, deletion and particular permissions at the record level;
- Detailed comparison and merge.

Thus, *aggregated lists should be used only for small volumes of simple data (one or two dozen occurrences), with no advanced requirements*. For larger volumes of data or more advanced functionalities, it is strongly advised to use an `osd:table` declaration.

For more information on table declarations, see Tables and relationships [p 493].

81.7 Including external data models

Including another data model in your current model allows you to use the reusable types that are defined in that data model. You can thus use the inclusion of external data models to share data types between multiple XML Schema Documents.

To include another XML Schema Document in your model, thereby including the data types that it defines, specify the `xs:include` element as follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
   <xs:include  schemaLocation="./schemaToInclude.xsd"/>
 ...
</xs:schema>
```

The attribute `schemaLocation` is mandatory and must specify either an absolute or a relative path to the XML Schema Document to include.

The inclusion of XML Schema Documents is not namespace aware, thus all included data types must belong to the same namespace. As a consequence, including XML Schema Documents that define data types of the same name is not supported.

EBX includes extensions with specific URNs for including embedded data models and data models packaged in modules.

To include an embedded data model in a model, specify the URN defined by EBX. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
 <xs:include schemaLocation="urn:ebx:publication:myPublication"/>
 ...
</xs:schema>
```

To include a data model packaged in a module, specify the specific URN defined by EBX. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
 <xs:include schemaLocation="urn:ebx:module:aModuleName:/WEB-INF/ebx/schema/myDataModel.xsd"/>
 ...
</xs:schema>
```

See `SchemaLocation`API for more information about specific URNs supported by EBX.

Note

If the packaged data model uses Java resources, the class loader of the module containing the data model will be used at runtime for resolving these resources.

CHAPTER **82**

Tables and relationships

This chapter contains the following topics:

82.1 **Tables**

Overview

TIBCO EBX supports the features of relational database tables, including the handling of large volumes of records, and identification by primary key.

Tables provide many benefits that are not offered by aggregated lists [p 490]. Beyond relational capabilities, some features that tables provide are:

- filters and searches;
- sorting, views and hierarchies;
- identity constraints: primary keys, foreign keys [p 498] and uniqueness constraints [p 515];
- specific permissions for creation, modification, and deletion;
- dynamic and contextual permissions at the individual record level;
- detailed comparison and merge;
- ability to have inheritance at the record level (see dataset inheritance [p 272]);
- performance and memory optimization.

See also

Foreign keys [p 498]

Associations [p 501]

Working with existing datasets [p 123]

Simple tabular views [p 116]

Hierarchical views [p 117]

History [p 251]

Declaration

A table element, which is an element with *maxOccurs > 1*, is declared by adding the following annotation:

```
<xs:annotation>
 <xs:appinfo>
  <osd:table>
   <primaryKeys>/pathToField1 /pathToField...n</primaryKeys>
  </osd:table>
 </xs:appinfo>
</xs:annotation>
```

Common properties

Element	Description	Required
`primaryKeys`	Specifies the primary key fields of the table. Each field of the primary key must be denoted by its absolute XPath notation that starts just under the root element of the table. If there are multiple fields in the primary key, the list is delimited by whitespace. **Note:** Whitespaces in primary keys of type `xs:string` are handled differently. See Whitespace handling for primary keys of type string [p 526].	Yes.

`defaultLabel`	Defines the end-user display of records. Multiple variants can be specified:
	• A static non-localized expression is defined using the `defaultLabel` element, for example:
	`<defaultLabel>Product: ${./productCode}</defaultLabel>`
	• Static localized expressions are specified using the `defaultLabel` element with the attribute `xml:lang`, for example:
	`<defaultLabel xml:lang="fr-FR">Produit : ${./productCode}</defaultLabel>`
	`<defaultLabel xml:lang="en-US">Product: ${./productCode}</defaultLabel>`
	• A JavaBean that implements the interface `UILabelRenderer`API and/or the interface `UILabelRendererForHierarchy`API. The JavaBean is specified by means of the attribute `osd:class`, for example:
	`<defaultLabel osd:class="com.wombat.MyLabel"></defaultLabel>`
	Note: The priority of the tags when displaying the user interface is the following:
	1. `defaultLabel` tags with a JavaBean (but it is not allowed to define several renderers of the same type);
	2. `defaultLabel` tags with a static localized expression using the `xml:lang` attribute;
	3. `defaultLabel` tags with a static non-localized expression.
	Attention: Access rights defined on associated datasets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
		No.

`index`	Specifies an index for speeding up requests that match this index (see performances [p 295]). The attribute `name` is mandatory. Each field of the index must be denoted by its absolute XPath notation, which starts just under the root element of the table. If there are multiple fields in the index, the list is delimited by whitespace. **Note:** • Indexing only concerns semantic and relational tables. History and replica tables are not affected. • It is possible to define multiple indexes on a table. • It is not possible to define two indexes with the same name.	No.

Element	Description	Required
	• It is not possible to declare two indexes containing the exact same fields. • An indexed field must be terminal. • An indexed field cannot be a list nor under a list. • A field declared as an *inherited field* cannot be indexed. • A field declared as a function cannot be indexed. For performance purposes, the following nodes are automatically indexed: • Primary keys nodes. See primary keys [p 493]. • Nodes defining a foreign key constraint. See foreign key constraint [p 498]. • Nodes declared as being unique. See uniqueness constraint [p 515]. • Auto-incremented nodes. See auto-incremented values [p 529].	
`recordForm`	Defines a specific component for customizing the record form in a dataset. This component is defined using a JavaBean that extends `UIForm`API or implements `UserServiceRecordFormFactory`API. The JavaBean is specified by means of the attribute `osd:class`, for example: `<recordForm osd:class="com.wombat.MyRecordForm"/>`	No.

Example

Below is an example of a product catalog:

```
<xs:element name="Products" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
   <xs:documentation>
    <osd:label>Product Table </osd:label>
    <osd:description>List of products in Catalog </osd:description>
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
  <xs:annotation>
   <xs:appinfo>
    <osd:table>
     <primaryKeys>./productRange /productCode</primaryKeys>
     <index name="indexProductCode">/productCode</index>
    </osd:table>
   </xs:appinfo>
  </xs:annotation>
   <xs:sequence>
    <xs:element name="productRange" type="xs:string"/><!-- key -->
    <xs:element name="productCode" type="xs:string"/><!-- key -->
    <xs:element name="productLabel" type="xs:string"/>
    <xs:element name="productDescription" type="xs:string"/>
    <xs:element name="productWeight" type="xs:int"/>
    <xs:element name="productType" type="xs:string"/>
    <xs:element name="productCreationDate" type="xs:date"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Catalogs" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
   <xs:documentation>
    <osd:label>Catalog Table</osd:label>
    <osd:description>List of catalogs</osd:description>
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:annotation>
    <xs:appinfo>
     <osd:table>
      <primaryKeys>/catalogId</primaryKeys>
     </osd:table>
    </xs:appinfo>
   </xs:annotation>
   <xs:sequence>
    <xs:element name="catalogId" type="xs:string"/><!-- key -->
    <xs:element name="catalogLabel" type="xs:string"/>
    <xs:element name="catalogDescription" type="xs:string"/>
    <xs:element name="catalogType" type="xs:string"/>
    <xs:element name="catalogPublicationDate" type="xs:date"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

Properties related to dataset inheritance

The following properties are only valid in the context of dataset inheritance:

Element	**Description**	**Required**
`onDelete-deleteOccultingChildren`	Specifies whether, upon record deletion, child records in occulting mode are also to be deleted. Valid values are: `never` or `always`.	No, default is `never`.
`mayCreateRoot`	Specifies whether root record creation is allowed. The expression must follow the syntax below. See definition modes [p 272].	No, default is `always`.
`mayCreateOverwriting`	Specifies whether records are allowed to be overwritten in child datasets. The expression must follow the syntax below. See definition modes [p 272].	No, default is `always`.
`mayCreateOcculting`	Specifies whether records are allowed to be occulted in child datasets. The expression must follow the syntax below. See definition modes [p 272].	No, default is `always`.
`mayDuplicate`	Specifies whether record duplication is allowed. The expression must follow the syntax below.	No, default is `always`.
`mayDelete`	Specifies whether record deletion is allowed. The expression must follow the syntax below.	No, default is `always`.

The `may...` expressions specify when the action is possible, though the ultimate availability of the action also depends on the user access rights. The expressions have the following syntax:

```
expression ::= always | never | <condition>*

condition ::= [root:yes | root:no]

"always": the operation is "always" possible (but user rights may restrict this).

"never": the operation is never possible.

"root:yes": the operation is possible if the record is in a root instance.

"root:no": the operation is not possible if the record is in a root instance.
```

If the record does not define any specific conditions, the default is used.

See also*Dataset inheritance* *[p 271]*

Using toolbars

It is possible to define the toolbars to display in the user interface using the element `defaultView/toolbars` under `xs:annotation/appinfo/osd:table`. A toolbar allows to customize the buttons and menus to display when displaying a table view, a hierarchical view, or a record form.

The table below presents the elements that can be defined under `defaultView/toolbars`.

Element	Description	Required
`tabularViewTop`	Defines the toolbar to use on top of the default table view.	No.
`tabularViewRow`	Defines the toolbar to use on each row of the default table view.	No.
`recordTop`	Defines the toolbar to use in the record form.	No.
`hierarchyViewTop`	Defines the toolbar to use in the default hierarchy view of the table.	No.

See also *Toolbars* *[p 547]*

Example

Below is an example of custom toolbars used by a product catalog:

```
<xs:element name="Products" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
   <xs:documentation>
    <osd:label>Product Table </osd:label>
    <osd:description>List of products in Catalog </osd:description>
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
  <xs:annotation>
   <xs:appinfo>
    <osd:table>
     <primaryKeys>./productRange /productCode</primaryKeys>
    <defaultView>
     <toolbars>
      <tabularViewTop>toolbar_name_for_tabularViewTop</tabularViewTop>
      <tabularViewRow>toolbar_name_for_tabularViewRow</tabularViewRow>
      <recordTop>toolbar_name_for_recordTop</recordTop>
      <hierarchyViewTop>toolbar_name_for_hierarchyViewTop</hierarchyViewTop>
     </toolbars>
    </defaultView>
   </osd:table>
   </xs:appinfo>
  </xs:annotation>
 ...
  </xs:complexType>
</xs:element>
```

> **Note**
>
> If a toolbar does not exist or is not available for a specific location then no toolbar will be displayed in the user interface in the corresponding location.

82.2 Foreign keys

Declaration

A reference to a table [p 493] is defined using the extended facet `osd:tableRef`.

The node holding the `osd:tableRef` declaration must be of type `xs:string`. At the instantiation, any value of the node identifies a record in the target table using its **primary key syntax** `PrimaryKey`.

`syntax`API. This extended facet is also interpreted as an enumeration whose values refer to the records in the target table.

Element	**Description**	**Required**
`tablePath`	XPath expression that specifies the target table.	Yes.
`container`	Reference of the dataset that contains the target table.	Only if the dataspace element is defined. Otherwise, default is the current dataset.
`branch`	Reference of the dataspace that contains the container dataset.	No, default is the current dataspace or snapshot.

`display`	Custom display for presenting the selected foreign key in the current record and the sorted list of possible keys. Two variants can be specified, either pattern-based expressions, or a JavaBean if the needs are very specific:
	• Static expressions are specified using the `display` and `pattern` elements. These static expressions can be localized using the additional attribute `xml:lang` on the `pattern` element, for example:
	`<display>`
	`<pattern>Product : ${./productCode}</pattern>`
	`<pattern xml:lang="fr-FR">Produit : ${./productCode}</pattern>`
	`<pattern xml:lang="en-US">Product: ${./productCode}</pattern>`
	`</display>`
	• A JavaBean that implements the interface `TableRefDisplay`API. It is specified using the attribute `osd:class`. For example:
	`<display osd:class="com.wombat.MyLabel"></display>`
	It is not possible to define both variants on the same foreign key element.
	Attention: Access rights defined on associated datasets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
		No, if the `display` property is not specified, the table's record rendering [p 495] is used.

`filter`	Specifies an additional constraint that filters the records of the target table. Two types of filters are available: • An XPath filter is an XPath predicate in the target table context. It is specified using the `predicate` element. For example: `<filter><predicate>type = ${../refType}</predicate></filter>` A localized validation message can be specified using the element `validationMessage`, which will be displayed to the end-user at the validation time if a record is not accepted by the filter. A specific severity level can be defined in a nested `severity` element. The default severity is 'error'. Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute.	No.

Element	Description	Required
	To specify a default message for unsupported locales, define a `message` element with no `xml:lang` attribute. In the user interface, the XPath filter is applied to filter a table according to the value of a foreign key field. That is, if a foreign key field specifies an XPath filter in a data model, then it will be reused in the filter pane to restrict the set of values in the associated combo-box displayed in the filter pane. However, the predicate used by the filter pane will only take into account the non-contextual parts of the predicate. • A programmatic filter is a JavaBean that implements the interface `TableRefFilter`API. It is specified using the attribute `osd:class`. For example: `<filter osd:class="com.wombat.MyFilter"></filter>` Additional validation messages can be specified during the setup of the programmatic filter. In the user interface, programmatic filters are not applied to filter the set of values in the associated combo-box displayed in the filter pane. However, it is possible to specify an additional XPath predicate that will be used in the filter pane of the user interface. This XPath predicate is specified during the setup of the programmatic filter using the method `TableRefFilterContext.setFilterForSearch`API. **Note:** The attributes `osd:class` and the property `predicate` cannot be set simultaneously. The validation search XPath functions are forbidden on a `tableRef` filter.	
`validation`	Specifies localized validation messages for the `osd:tableRef` and error management policy. A specific severity level can be defined in a nested `severity` element. The default severity is 'error'. An error management policy can be defined in a nested `blocksCommit` element. The error management policy that blocks all operations does not apply to filters. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected, and a validation error will be reported. Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a `message` element with no `xml:lang` attribute.	No.

Attention

You can create a dataset which has a foreign key to a container that does not exist in the repository. However, the content of this dataset will not be available until the container is created. After the creation of the container, a data model refresh is required to make the dataset available. When creating a dataset that refers to a container that does not yet exist, the following limitations apply:

- Triggers defined at the dataset level are not executed.
- Default values for fields that are not contained in tables are not initialized.

- During an archive import, it is not possible to create a dataset that refers to a container that does not exist.

Example

The example below specifies a foreign key in the 'Products' table to a record of the 'Catalogs' table.

```
<xs:element name="catalog_ref" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:tableRef>
     <tablePath>/root/Catalogs</tablePath>
     <display>
       <pattern xml:lang="en-US">Catalog: ${./catalogId}</pattern>
       <pattern xml:lang="fr-FR">Catalogue : ${./catalogId}</pattern>
     </display>
      <validation>
       <severity>error</severity>
       <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
       <message>A default error message</message>
       <message xml:lang="en-US">A localized error message</message>
       <message xml:lang="fr-FR">Un message d'erreur localisé</message>
     </validation>
    </osd:tableRef>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

See also

Table definition *[p 493]*

Primary key syntax `PrimaryKey.syntax`API

Extraction of foreign keys (XPath predicate syntax) *[p 232]*

Associations *[p 501]*

View for advanced selection *[p 541]*

`SchemaNode.getFacetOnTableReference`API

`SchemaFacetTableRef`API

82.3 Associations

Overview

An association provides an abstraction over an existing relationship in the data model, and allows an easy model-driven integration of *associated objects* in the user interface and in data services.

Several types of associations are supported:

- *'By foreign key'* specifies the inverse relationship of an existing foreign key field [p 498].
- *'Over a link table'* specifies a relationship based on an intermediate link table (such tables are often called "join tables"). This link table has to define two foreign keys, one referring to the 'source' table (the table holding the association element) and another one referring to the 'target' table.
- *'By an XPath predicate'* specifies a relationship based on an XPath predicate.

For an association, it is also possible to:

- Filter associated objects by specifying an additional XPath filter.

- Configure a tabular view to define the fields that must be displayed in the associated table.
- Define how associated objects are to be rendered in forms.
- Hide/show associated objects in the data service 'select' operation. See Hiding a field in Data Services [p 540].
- Specify the minimum and maximum number of associated objects that are required.
- Add validation constraints using XPath predicates for restricting associated objects.

See also

*`SchemaNode.getAssociationLink`*API

*`SchemaNode.isAssociationNode`*API

*`AssociationLink`*API

Declaration

Associations are defined in the data model using the XML Schema element `osd:association` under `xs:annotation/appInfo`.

Restrictions:

- An association must be a simple element of type *xs:string*.
- An association can only be defined inside a table.

> **Note**
>
> The "official" cardinality constraints (`minOccurs="0" maxOccurs="0"`) are required because, from an instance of XML Schema, the corresponding node is absent. In other words, an association has no value and is considered as a "virtual" element as far as XML and XML Schema is concerned.

The table below presents the elements that can be defined under `xs:annotation/appInfo/osd:association`.

Element	Description	Required
`tableRefInverse`	Defines the properties of an association that is the inverse relationship of a *foreign key*. Element `fieldToSource` defines the foreign key that refers to the source table of the association. The element `fieldToSource` is mandatory and must specify a foreign key field that refers to the table containing the association.	Yes if the association is the inverse relationship of a *foreign key*, otherwise no.
`linkTable`	Defines the properties of an association over a link table. The element `table` specifies the link table used by the association. The element `table` is mandatory and must refer to an existing table. **Important:** In order to be used by an association, a link table must define a primary key that is composed of auto-incremented fields and/or the foreign key to the source or target table of the association. The element `fieldToSource` defines the foreign key that refers to the source table of the association. The element `fieldToSource` is mandatory and must specify a foreign key field that refers to the table containing the association. The element `fieldToTarget` defines the foreign key that refers to the target table of the association. The element `fieldToTarget` is mandatory and must specify a foreign key field.	Yes if the association is over a link table, otherwise no.
`xpathLink`	Defines the properties of an association that is based on an *XPath predicate*. The `predicate` element specifies the criteria of the association, relative to the current node. Examples: `/root/Products[catalog_ref =${../catalogId}]` or `//Products[catalog_ref = ${../catalogId}]` or `../Products[catalog_ref =${../catalogId}]`. The path to the predicate, for example `../Products`, specifies the target table of the association. This part of the path is resolved with respect to the current table. It is not possible to refer to a table using a relative path if the association targets a table in another dataset. If the association depends on fields of the source table, the XPath expression predicate must include references to the elements on which it depends using the notation `${<relative-path>}` where `relative-path` is a path that identifies the element relative to the association node. See EBX XPath supported syntax [p 227]. **Note**	Yes if the association is based on an *XPath predicate*, otherwise no.

Element	Description	Required
	The validation search XPath functions are forbidden on an XPath link.	
`filter`	Defines an XPath predicate to filter associated objects using the `predicate` element. For example: `<filter><predicate>type = ${../refType}</predicate></filter>` It is only possible to use fields from the source and the target tables when defining an XPath filter. That is, if it is an association over a link table, it is not possible to use fields of the link table in the XPath filter. **Error message on creation**: in the user interface, the record creation is blocked when a user submits a new associated record that does not comply with the filter. The error message can be customized using the element `checkOnAssociatedRecordCreation/message`. Each localized message variant is defined in a nested message element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a message element with no `xml:lang` attribute. See Examples [p 510] for more information on this property. **Note** The validation search XPath functions are forbidden for association filter.	No.
`xpathFilter`	**Note:** Deprecated. This property has been replaced by the property `filter`. Defines an XPath predicate to filter associated objects.	No.
`recordForm`	Defines a specific component for customizing the form of an associated record. This component is defined using a JavaBean that implements `UserServiceAssociationRecordFormFactory`API. The JavaBean is specified by means of the attribute `osd:class`, for example: `<recordForm osd:class="com.wombat.MyRecordFormFactory"/>`	No.

It is possible to refer to another dataset. For that, the following properties must be defined either under the element `tableRefInverse`, `linkTable` or `xpathLink` depending on the type of the association:

Element	Description	Required
`schemaLocation`	Defines the data model containing the fields used by the association. The data model is defined using a specific URN that allows referring to embedded data models and data models packaged in modules. See `SchemaLocation`API for more information about specific URNs supported by EBX.	Yes.
`dataSet`	Defines the dataset used by the association. This dataset must use the data model specified by the element `schemaLocation`.	Yes.
`dataSpace`	Defines the dataspace containing the dataset used by the association.	No.

Important: When creating a dataset, you can create a dataset that defines an association to a container that does not yet exist in the repository. However, the content of this dataset will not be available immediately upon creation. After the absent container is created, a data model refresh is required in order to make the dataset available. When creating a dataset that refers to a container that does not yet exist, the following limitations apply:

- Triggers defined at the dataset level are not executed.
- Default values on fields outside tables are not initialized.
- During an archive import, it is not possible to create a dataset that refers to a container that does not exist.

User interface integration

It is possible to define how associated objects are to be rendered in forms, using the element `osd:defaultView/displayMode` under `xs:annotation/appinfo`.

Possible values are:

- `inline`, specifies that associated records are to be rendered in the form at the same position of the association in the data model.
- `tab`, specifies that associated records are to be rendered in a specific tab.
- `link`, specifies that associated records are to be rendered in a modal window.

By default, associated records are rendered `inline` if this property is not defined.

The following example specifies that associated objects are to be rendered `inline` in the form:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
  <osd:association>
   <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
  </osd:association>
  <osd:defaultView>
   <displayMode>inline</displayMode>
  </osd:defaultView>
   </xs:appinfo>
   </xs:annotation>
```

```
</xs:element>
```

The following example specifies that associated objects are to be rendered in a specific tab:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
  <osd:association>
   <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
  </osd:association>
  <osd:defaultView>
   <displayMode>tab</displayMode>
  </osd:defaultView>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

Using toolbars

It is possible to define the toolbars to display in the user interface using the element `osd:defaultView/toolbars` under `xs:annotation/appinfo`. A toolbar allows to customize the buttons and menus to display when displaying the tabular view of an association.

The table below presents the elements that can be defined under `osd:defaultView/toolbars`.

Element	Description	Required
`tabularViewTop`	Defines the toolbar to use in the default table view of this association.	No.
`tabularViewRow`	Defines the toolbar to use for each row of the default view of this association.	No.

The following example shows how to use toolbars from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
   </osd:association>
   <osd:defaultView>
    <toolbars>
     <tabularViewTop>toolbar_name_for_tabularViewTop</tabularViewTop>
     <tabularViewRow>toolbar_name_for_tabularViewRow</tabularViewRow>
    </toolbars>
   </osd:defaultView>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

> **Note**
>
> It is only possible to use the toolbars defined in the data model containing the target table of the association. That is, if the target table of the association is defined in another data model, then it is only possible to reference a toolbar defined in this data model and not in the one holding the association.

See also *Toolbars* *[p 547]*

Customized view of associated objects

A specific tabular view can be specified to define the fields that must be displayed in the target table. If a tabular view is not defined, all columns that a user is allowed to view, according to the granted access rights, are displayed. A tabular view is defined using the element `osd:defaultView/tabularView` under `xs:annotation/appinfo`.

The table below shows the elements that can be defined under `osd:defaultView/tabularView`.

Element	**Description**	**Required**
`column`	Define a field of the target table to display. The specified path must be absolute from the target table and must refer to an existing field. Several `column` elements can be defined to specify the fields that are to be displayed.	No.
`sort`	Define a field that can be used to sort associated objects. Several`sort` elements can be defined to specify the fields that can be used to sort associated objects. The element `nodePath` defines the path of the field that can be used to sort associated objects. The element `isAscending` specifies whether the sort order is ascending (true) or descending (false).	No.

The following example shows how to define a tabular view from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
   </osd:association>
   <osd:defaultView>
    <tabularView>
     <column>/productRange</column>
     <column>/productCode</column>
     <column>/productLabel</column>
     <column>/productDescription</column>
     <sort>
        <nodePath>/productLabel</nodePath>
        <isAscending>true</isAscending>
     </sort>
    </tabularView>
   </osd:defaultView>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Actions in the user interface

In the user interface, it is possible to perform the following actions:

- **Create**: it allows directly creating an object in the target table of the association. When a new object is created, it is automatically associated with the current record.
- **Duplicate**: allows to duplicate an object in the target table of the association. When a new object is created, it is automatically associated with the current record.

- **Associate**: associates an existing object with the current record. In the case of an association over a link table, a record in the link table is automatically created to materialize the link between the current record and the existing object.
- **Move**: associates the selected objects to a different record than the current one. In the case of an association over a link table, the previous link record is automatically deleted and a new record in the link table is automatically created to materialize the link between the selected objects and their new parent record.
- **Delete**: deletes selected associated objects in the target table of the association.
- **Detach**: breaks the semantic link between the current record and the selected associated objects. In the case of an association over a link table, the records in the link table are automatically deleted, to break the links between the current record and associated objects.

> **Note**
>
> The actions *associate* and *detach* are not available when the association is defined using an **XPath predicate** (element *xpathLink*).

Customized view for actions

A published view, tabular or hierarchical, can be specified to define how objects should be displayed when performing an action through the user interface. A published view is defined using the element `osd:defaultView/associationViews` under `xs:annotation/appinfo`.

The table below shows the elements that can be defined under `osd:defaultView/associationViews`.

Element	Description	Required
`viewForAssociateAction`	Define a published view to be used when displaying the objects in the target table to be associated with the current record. The specified view must be published and created upon the target table of the association.	No.
`viewForMoveAction`	Define a published view to be used when moving an associated object to another record of the current table. The specified view must be published and created upon the current table.	No.

The following example shows how to define views from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
   </osd:association>
   <osd:defaultView>
    <associationViews>
     <viewForAssociateAction>view_name_for_catalogs</viewForAssociateAction>
     <viewForMoveAction>view_name_for_products</viewForMoveAction>
    </associationViews>
   </osd:defaultView>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Validation

Some controls can be defined on associations, in order to restrict associated objects. These controls are defined under the element `osd:association`.

The table below presents the controls that can be defined under xs:annotation/appInfo/osd:association.

Element	Description	Required
`minOccurs`	Specifies the minimum number of associated objects that are required for this association. This minimum number is defined using the element `value` and must be a positive integer.	No, by default the minimum is not restricted.
`maxOccurs`	Specifies the maximum number of associated objects that are allowed for this association. This maximum number is defined by the element `value` and must be either a positive integer or the raw string `unbounded` which indicates that this maximum is not restricted. The maximum number of associated objects must be greater than the minimum number of associated objects.	No, by default the maximum is not restricted.
`constraint`	Defines an XPath predicate for restricting associated records. It is only possible to use fields from the source and the target table when defining an XPath predicate. That is, if it is an association over a link table, it is not possible to use fields of the link table in the XPath predicate. In associated datasets, a validation message of the specified severity is added and displayed to the end-user at the validation time when an associated record does not comply with the specified constraint.	No.
`validation`	A validation message can be defined under the elements `minOccurs`, `maxOccurs` and `constraint`, using the element `validation`. The severity of the validation message is specified using the element `severity`. Possible severities are: `error`, `warning` and `info`. If the severity is not specified then, by default, the severity `error` is used. A localized validation message can be specified using the element `message`, which will be displayed to the end-user at the validation time if an association does not comply with this constraint. Each localized message variant is defined in a nested message element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a message element with no `xml:lang` attribute.	No.

Data services integration

It is possible to define whether associated objects must be hidden in the Data service `select` operation. For this, the property `osd:defaultView/hiddenInDataServices` under `xs:annotation/xs:appinfo` can be set on the association. Setting the property to 'true' will hide associated objects in the Data service `select` operation. If this property is not defined then, by default, associated objects will be shown in the Data service `select` operation.

See also

Hiding a field in Data Services [p 540]

Association field [p 616]

Examples

For example, the product catalog data model defined previously [p 496] specifies that a product belongs to a catalog (explicitly defined by a foreign key in the 'Products' table). The reverse relationship (that a catalog has certain products) is not easily represented in XML Schema, unless the 'Catalogs' table includes the following association that is the inverse of a foreign key:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
   <osd:association>
   <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
   </osd:association>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

For an association over a link table, we can consider the previous example and bring some updates. For instance, the foreign key in the 'Products' table is deleted and the relation between a product and a catalog is redefined by a link table (named 'Catalogs_Products') that has a primary key composed of two foreign keys: one that refers to the 'Products' table (named 'productRef') and another to the 'Catalogs' table (named 'catalogRef'). The following example shows how to define an association over a link table from this new relationship:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
  <osd:association>
   <linkTable>
     <table>/root/Catalogs_Products</table>
     <fieldToSource>./catalogRef</fieldToSource>
     <fieldToTarget>./productRef</fieldToTarget>
   </linkTable>
  </osd:association>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

The following example shows an association that refers to a foreign key in another dataset. In this example, the 'Products' and 'Catalogs' tables are not in the same dataset:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
   <osd:association>
   <tableRefInverse>
    <schemaLocation>urn:ebx:module:aModuleName:/WEB-INF/ebx/schema/products.xsd</schemaLocation>
    <dataSet>Products</dataSet>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
   </osd:association>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

The following example defines an XPath filter to associate only products of the 'Technology' type:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
   <osd:association>
   <tableRefInverse>
    <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
   <filter>
    <predicate>./productType = 'Technology'</predicate>
    <checkOnAssociatedRecordCreation>
     <message>A default message</message>
                    <message xml:lang="en-US">A localized message</message>
                    <message xml:lang="fr-FR">Un message localisé</message>
    </checkOnAssociatedRecordCreation>
   </filter>
   </osd:association>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

The following example specifies the minimum number of products that are required for a catalog:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
      <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
     <minOccurs>
     <value>1</value>
     <validation>
      <severity>warning</severity>
      <message xml:lang="en-US">One product should at least be associated to this catalog.</message>
      <message xml:lang="fr-FR">Un produit doit au moins être associé à ce catalogue.</message>
     </validation>
    </minOccurs>
   </osd:association>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

The following example specifies that a catalog must contain at most ten products:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
    <maxOccurs>
     <value>10</value>
     <validation>
      <severity>warning</severity>
      <message xml:lang="en-US">Too much products for this catalog.</message>
      <message xml:lang="fr-FR">Ce catalogue a trop de produits.</message>
     </validation>
    </maxOccurs>
   </osd:association>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

CHAPTER 83

Constraints, triggers and functions

Facets allow you to define data constraints in your data models. TIBCO EBX supports XML Schema facets and provides extended and programmatic facets for advanced data controls.

This chapter contains the following topics:

83.1 XML Schema supported facets

The tables below show the facets that are supported by different data types.

Key:

- **X -** Supported
- **1 -** The `whiteSpace` facet can be defined, but is not interpreted by EBX
- **2 -** In XML Schema, boundary facets are not allowed on the type `string`. Nevertheless, EBX allows such facets as extensions.
- **3 -** The `osd:resource` type only supports the facet `FacetOResource`, which is required. See Extended Facets [p 516].

- **4** - `osd:dataspaceKey`, `osd:datasetName` and `osd:color` types do not support facets. Only Programmatic constraints [p 520] are supported on these types.

	length	minLength	max Length	pattern	enumeration	white Space
`xs:string`	X	X	X	X	X	1
`xs:boolean`				X		1
`xs:decimal`				X	X	1
`xs:dateTime`				X	X	1
`xs:time`				X	X	1
`xs:date`				X	X	1
`xs:anyURI`	X	X	X	X	X	1
`xs:Name`	X	X	X	X	X	1
`xs:integer`				X	X	1
`osd:resource` [p 482]3						1
`osd:dataspaceKey` [p 484]4						1
`osd:datasetName` [p 486]4						1
`osd:color` [p 484]4						1

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
`xs:string`			2	2	2	2
`xs:boolean`						
`xs:decimal`	X	X	X	X	X	X
`xs:dateTime`			X	X	X	X
`xs:time`			X	X	X	X
`xs:date`			X	X	X	X

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
xs:anyURI						
xs:Name			2	2	2	2
xs:integer	X	X	X	X	X	X
osd:resource [p 482][3]						
osd:dataspaceKey [p 484][4]						
osd:datasetName [p 486][4]						
osd:color [p 484][4]						

Example:

```
<xs:element name="loanRate">
 <xs:simpleType>
  <xs:restriction base="xs:decimal">
   <xs:minInclusive value="4.5" />
   <xs:maxExclusive value="17.5" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Uniqueness constraint

It is possible to define a uniqueness constraint, using the standard XML Schema element xs:unique. This constraint indicates that a value or a set of values has to be unique inside a table.

Example:

In the example below, a uniqueness constraint is defined on the 'publisher' table, for the target field 'name'. This means that no two records in the 'publisher' table can have the same name.

```
<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>
```

A uniqueness constraint has to be defined within a table and has the following properties:

Property	Description	Mandatory
`name` attribute	Identifies the constraint in the data model.	Yes
`xs:selector` element	Indicates the table to which the uniqueness constraint applies using a restricted XPath expression ('..' is forbidden). It can also indicate an element within the table (without changing the meaning of the constraint).	Yes
`xs:field` element	Indicates the field in the context whose values must be unique, using a restricted XPath expression. It is possible to indicate that a set of values must be unique by defining multiple `xs:field` elements.	Yes

Note

Undefined values (null values) are ignored on uniqueness constraints applied to single fields. On multiple fields, undefined values are taken into account. That is, sets of values are considered as being duplicated if they have the same defined and undefined values.

Additional localized validation messages can be defined using the element `osd:validation` under the elements `annotation/appinfo`. If no custom validation messages are defined, a built-in validation message will be used.

Limitations:

1. The target of the `xs:field` element must be in a table.
2. The uniqueness constraint does not apply to fields inside an aggregated list.
3. The uniqueness constraint does not apply to computed fields.

See also ***Uniqueness constraint in the Java API*** `UniquenessConstraint`API

83.2 Extended facets

EBX provides additional constraints that are not specified in XML Schema, but that are useful for managing master data.

In order to guarantee XML Schema conformance, these extended facets are defined under the element `annotation/appinfo/otherFacets`.

Foreign keys

EBX allows to create a reference to an existing table by means of a specific facet. See Foreign keys [p 498] for more information.

Dynamic constraints

Dynamic constraint facets retain the semantics of XML Schema, but the value attribute is replaced with a path attribute that allows fetching the value from another element. The available dynamic constraints are:

- `length`

- `minLength`
- `maxLength`
- `maxInclusive`
- `maxExclusive`
- `minInclusive`
- `minExclusive`

Using these facets, the data model can be modified dynamically.

Example:

```
<xs:element name="amount">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:minInclusive path="/domain/Loan/Pricing/AmountMini/amount" />
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
   ...
</xs:element>
```

In this example, the boundary of the facet `minInclusive` is not statically defined. The value of the boundary comes from the node */domain/Loan/Pricing/AmountMini/amount*.

Restrictions:

- Target field cannot be an aggregated list. That is, it cannot define `maxOccurs` = 1.
- Data type of the target field must be compatible with the facet. That is, it must be:
  - of type `integer` for facets `length`, `minLength` and `maxLength`.
  - compatible with the data type of the field holding the facet for facets `maxInclusive`, `maxExclusive`, `minInclusive` and `minExclusive`.
- Target field cannot be in a table if the field holding the facet is not in a table.
- Target field must be in the same table or outside a table if the field holding the facet is in a table.
- If the target field is under one or more aggregated lists, the field holding the facet must also be under these aggregated lists. That is: the field holding the facet must be in the same list occurrence as the target field, or in a parent occurrence, so that the target field refers to a single value, from an XPath perspective.

FacetOResource constraint

This facet must be defined for every definition using the type `osd:resource`, to specify the subset of available packaged resource files as an enumeration. For more information on this type, see osd:resource type [p 484]. It has the following attributes:

`moduleName`	Indicates, using an alias, the EBX module that contains the resource. If the resource is contained in the current module, the alias must be preceded by "wbp". Otherwise, the alias must be one of the values defined in the element `<dependencies>` in the file `module.xml`.
`resourceType`	Represents the resource type that is one of the following values: 'Image', 'JavaScript', 'Style sheet', 'HTML'.
`relativePath`	Indicates in which directory the resources will be located. This directory must be located under the directory that corresponds to the resource type. For example, for an "Image" type resource, the directory www/common/images/, located at the same level as the directory WEB-INF/ of the target module, will be used and the relative path will have to be defined from this. Furthermore, if a resource is defined in a localized directory (www/fr/ for example), it will only be taken into account if another resource with the same name is defined in the directory www/common/.

This facet has the same behavior as an enumeration facet: the values are collected by recursively listing all the files in the local path in the specified resource type directory in the specified module.

Example:

```
<xs:element name="promotion" type="osd:resource">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:FacetOResource osd:moduleName="wbp"
     osd:resourceType="ext-images" osd:relativePath="promotion/" />
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</ xs:element>
```

For an overview of the standard directory structure of an EBX module (Java EE web application), see Module structure [p 459].

excludeValue constraint

This facet verifies that a value is not the same as the specified excluded value.

In this example, the empty string is excluded from the allowed values.

Example:

```
<xs:element name="roleName">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:excludeValue value="">
     <osd:validation>
```

```
      <severity>error</severity>
       <message>Please select address role(s).</message>
     </osd:validation>
    </osd:excludeValue>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType type="xs:string" />
</xs:element>
```

excludeSegment constraint

This facet verifies that a value is not included in a range of values. Boundaries are excluded.

Example:

In this example, values between 20000 and 20999 are not allowed.

```
<xs:element name="zipCode">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:excludeSegment minValue="20000" maxValue="20999">
     <osd:validation>
      <severity>error</severity>
       <message>Postal code not valid.</message>
     </osd:validation>
    </osd:excludeSegment>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType type="xs:string" />
</xs:element>
```

Enumeration facet defined using another node

By default, an enumeration facet is described statically in XML Schema.

The content of an enumeration facet can also be provided dynamically by a list of simple elements in the data model.

Example:

In this example, the content of an enumeration facet is sourced from the node `CountryList`.

```
<xs:annotation>
 <xs:appinfo>
  <osd:otherFacets>
   <osd:enumeration osd:path="../CountryList" />
  </osd:otherFacets>
 </xs:appinfo>
</xs:annotation>
```

The referred node `CountryList`:

- Must be an aggregated list, that is, `maxOccurs` > 1.
- Must be a list of elements of the same type as the node with the enumeration facet.
- Must be a node outside a table if the node with the enumeration facet is not inside a table.
- Must be a node outside a table or in the same table as the node with the enumeration facet if the node with this enumeration is inside a table.
- If the target field is under one or more aggregated lists, the field holding the facet must also be under these aggregated lists. That is: the field holding the facet must be in the same list occurrence as the target field, or in a parent occurrence, so that the target field refers to a single value, from an XPath perspective.

Example:

```
<xs:element name="FacetEnumBasedOnList">
 <xs:complexType>
```

```
  <xs:sequence>
   <xs:element name="CountryList" maxOccurs="unbounded">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="DE" osd:label="Germany" />
      <xs:enumeration value="AT" osd:label="Austria" />
      <xs:enumeration value="BE" osd:label="Belgium" />
      <xs:enumeration value="JP" osd:label="Japan" />
      <xs:enumeration value="KR" osd:label="Korea" />
      <xs:enumeration value="CN" osd:label="China" />
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
   <xs:element name="CountryChoice" type="xs:string">
    <xs:annotation>
     <xs:appinfo>
      <osd:otherFacets>
       <osd:enumeration osd:path="../CountryList" />
      </osd:otherFacets>
     </xs:appinfo>
    </xs:annotation>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

83.3 **Programmatic facets**

A programmatic constraint can be added to any XML element declaration for a simple type.

In order to guarantee XML Schema conformance, programmatic constraints are specified under the element `annotation/appinfo/otherFacets`.

Programmatic constraints

A programmatic constraint is defined by a Java class that implements the interface `Constraint`API.

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol.

Example:

In the example below, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

```
<xs:element name="amount">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraint class="com.foo.CheckAmount">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraint>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

See also*JavaBean specifications* *`Package com.orchestranetworks.schema.JavaBeans`*API

Programmatic enumeration constraints

An enumeration constraint adds an ordered list of values to a basic programmatic constraint. This facet allows selecting a value from a list. It is defined by a Java class that implements the interface `ConstraintEnumeration`API.

Example:

```
<xs:element name="amount">
 <xs:annotation>
```

```
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraintEnumeration class="com.foo.CheckAmountInEnumeration">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraintEnumeration>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Constraint on 'null' values

In some cases, a value is only mandatory if some conditions are satisfied, for example, if another field has a given value. In this case, the standard XML Schema attribute `minOccurs` is insufficient because it is static.

In order to check if a value is mandatory according to its context, the following requirements must be satisfied:

1. A programmatic constraint must be defined by a Java class (see above).
2. This class must implement the interface `ConstraintOnNull`API.
3. The XML Schema cardinality attributes must specify that the element is optional (`minOccurs="0"` and `maxOccurs="1"`).

> **Note**
>
> By default, constraints on 'null' values are not checked upon user input. In order to enable a check at the input, the 'checkNullInput' property [p 525] must be set. Also, if the element is terminal, the dataset must also be activated.

Example:

```
<xs:element name="amount" minOccurs="0" maxOccurs="1">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraint class="com.foo.CheckIfNull">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraint>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Constraints on table

A constraint on table is defined by a Java class that implements the interface `ConstraintOnTable`API. It can only be defined on table nodes.

As additional parameters can be defined. the implemented Java class must conform to the JavaBean protocol.

Example:

In the example below, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:appinfo>
   <osd:table>
    <primaryKeys>/key</primaryKeys>
   </osd:table>
   <osd:otherFacets>
    <osd:constraint class="com.foo.checkTable">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraint>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Attention

For performance reasons, constraints on tables are only checked when getting the validation report of a dataset or table. This means that these constraints are not checked when updates, such as record insertions, deletions or modifications, occur on tables. However, the internal incremental validation framework will optimize the validation cost of these constraints if dependencies are defined. For more information, see Validation [p 300].

See also***JavaBeans Specifications*** *`Package com.orchestranetworks.schema.JavaBeans`*API

83.4 Control policy

Blocking and non-blocking constraints

When an update in the repository is performed, and this update adds a validation error according to a given constraint, it is possible to specify whether the new error blocks the update (and cancels the transaction) or if it is considered as non-blocking (so that the update can be committed and the error can be corrected later). The element `blocksCommit` within the element `osd:validation` allows this specification, with the following supported values:

`onInsertUpdateOrDelete`	Specifies that the constraint must always remain valid after an operation (dataset update, dataset deletion, record creation, update or deletion). In this case, any operation that would violate the constraint is rejected and the values remain unchanged. This is the default and mandatory policy for primary key constraints, data type conversion constraints (an integer or a date must be well-written) and also structural constraints in mapped tables. This is also the default policy for foreign key constraints that are automatically set in blocking mode (because of a table in relational mode involved in the relationship) and that do not define a control policy.
`onUserSubmit-checkModifiedValues`	Specifies that the constraint must remain valid whenever a user modifies the associated value and submits a form. In this case, any form input that would violate the constraint is rejected and the values remain unchanged. This is the default policy for all blocking constraints mentioned in the previous case. For example, a foreign key constraint is by default not blocking (a record referred to by other records can be deleted, etc.), except in the context of a form submit.
`never`	Specifies that the constraint must never block operations. In this case, any operation that would violate the constraint is allowed. In the context of the user interface, this constraint does not block the form submission if the user sets a value that violates this constraint.

On foreign key constraints, the control policy that blocks all operations does not apply to filtered records. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected and a validation error occurs.

It is not possible to specify a control policy on structural constraints that are defined on relational data models or in mapped tables. That is, this property is not available for fixed length, maximum length, maximum number of digits, and decimal place constraints due to the validation policy of the underlying RDBMS blocking constraints.

This property does not apply to archive imports and when merging dataspaces. That is, all blocking constraints, except structural constraints, are always disabled when importing archives and merging dataspaces.

See also

Facet validation message with severity [p 534]

Foreign keys [p 498]

Relational mode *[p 245]*

XML Schema facet

The control policy is described by the element `osd:validation` in `annotation/appinfo` under the definition of the facet.

Example:

```
<xs:element name="zipCode">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:minInclusive value="1000">
    <xs:annotation>
     <xs:appinfo>
      <osd:validation>
       <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
      </osd:validation>
     </xs:appinfo>
    </xs:annotation>
   </xs:minInclusive>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

XML Schema enumeration facet

The control policy is described by the element `osd:enumerationValidation` in `annotation/appinfo` under the definition of the field.

Example:

```
<xs:element name="Gender">
 <xs:annotation>
  <xs:appinfo>
   <osd:enumerationValidation>
    <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
   </osd:enumerationValidation>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="0" osd:label="male" />
   <xs:enumeration value="1" osd:label="female" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

EBX facet

The control policy is described by the element `osd:validation` under the definition of the facet (which is defined in `annotation/appinfo/otherFacets`).

The control policy with values `onInsertUpdateOrDelete` and `onUserSubmit-checkModifiedValues` is only available on `osd:excludeSegment`, `osd:excludeValue` and `osd:tableRef` EBX facets.

The control policy with the value `never` can be defined on all EBX facets. On programmatic constraints, the control policy with the value `never` can only be set directly during the setup of the corresponding constraint. See `ConstraintContext.setBlocksCommitToNever`API and `ConstraintContextOnTable.setBlocksCommitToNever`API in the Java API for more information.

Example:

```
<xs:element name="price" type="xs:decimal">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:minInclusive path="../priceMin">
     <osd:validation>
      <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
     </osd:validation>
    </osd:minInclusive>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Check 'null' input

According to the EBX default validation policy, in order to allow temporarily incomplete input, a mandatory element is not checked for completion upon user input. Rather, it is verified at the dataset validation only. If completion must be checked immediately upon user input, the element must additionally specify the attribute `osd:checkNullInput="true"`.

> **Note**
>
> A value is mandatory if the data model specifies a mandatory element, either statically, using `minOccurs="1"`, or dynamically, using a constraint on 'null'. For terminal elements, mandatory values are only checked for an activated dataset. For non-terminal elements, the dataset does not need to be activated.

Example:

```
<xs:element name="amount" osd:checkNullInput="true" minOccurs="1">
 ...
</xs:element>
```

See also

Constraint on 'null' [p 521]

Whitespace management [p 525]

Empty string management [p 527]

EBX whitespace management for data types

According to XML Schema (see https://www.w3.org/TR/xmlschema-2/#rf-whiteSpace), whitespace handling must follow one of the procedures *preserve*, *replace* or *collapse*:

preserve	No normalization is performed, the value is unchanged.
replace	All occurrences of `#x9` (tab), `#xA` (line feed) and `#xD` (carriage return) are replaced with `#x20` (space).
collapse	After the processing according to the replace procedure, contiguous sequences of `#x20` are then collapsed to a single `#x20`, and any leading or trailing `#x20`s are removed.

General whitespace handling

EBX complies with the XML Schema recommendation:

- For fields of type `xs:string`, whether a primary key element or not, whitespaces are always preserved and an empty string is never converted to `null`.

- For other fields (non-`xs:string` type), whitespaces are always collapsed and empty strings are converted to `null`.

Attention

Exceptions:

- For fields of type `osd:html` or `osd:password`, whitespaces are always preserved and empty strings are converted to `null`.
- For fields of type `xs:string` that define the property `osd:checkNullInput="true"`, an empty string is interpreted as `null` at user input by EBX.

Whitespace handling upon user input

The rules described in the previous section are applied in the user interface, but leading and trailing whitespaces are removed upon user input. That is, in the user interface, whitespaces are by default always trimmed upon user input. Other input methods (Import XML/CSV, Data services, API updates) are not trimmed from the user interface.

Attention

Exceptions:

- For fields of type `osd:password`, whitespaces are not trimmed upon user input.
- For foreign key fields, whitespaces are not trimmed upon user input.

It is possible to indicate in a data model that whitespaces should not be trimmed upon user input. The attribute `osd:trim="disable"` can be set on the fields that allow leading and trailing whitespaces upon user input.

Example:

```
<xs:element name="field" osd:trim="disable" type="xs:string">
 ...
</xs:element>
```

Whitespace handling for primary keys of type string

For primary key columns of type `xs:string`, a default EBX constraint is defined. This constraint forbids empty strings and non-collapsed whitespace values when creating a record. That is, any record creation that would violate this constraint is rejected.

However, if the primary key node specifies its own `xs:pattern` facet, this facet overrides the default EBX constraint. For example, the specific pattern "`.*`" would accept any string, although this is not recommended.

The default constraint allows handling certain ambiguities. For example, it would be difficult for a user to distinguish between the following strings: "12 34" and "12 34". For generic values, this would not create conflicts, however, errors would occur for primary keys.

See also*Tables and relationships* *[p 493]*

Empty string management

Default conversion

For nodes of type `xs:string`, no distinction is made at user input between an empty string and a `null` value. That is, an empty string value is automatically converted to `null` at user input.

Distinction between empty strings and 'null' value

There are certain cases where the distinction is made between an empty string and the `null` value, such as when:

- A primary key defines a pattern that allows empty strings.
- An element defines a foreign key constraint and the target table has a single primary key defining a pattern that allows empty strings.
- An element defines a static enumeration that contains an empty string.
- An element defines a dynamic enumeration to another element with one of the aforementioned cases.

If the distinction is made between an empty string and a `null` value, this implies the following behaviors:

- An empty string will not be converted to `null` at user input,
- Input fields for nodes of type `xs:string` will display an additional button for setting the value of the node to `null`,
- At validation time, an empty string will be considered to be a compliant value with regard to the `minOccurs="1"` property.

Attention

In relational mode, the Oracle database does not support the distinction between empty strings and `null` values, and these specific cases are not supported.

See also*Relational mode* *[p 245]*

83.5 Triggers and functions

Computed values

By default, data is read and persisted in the XML repository. Nevertheless, data may be the result of a computation and/or external database access, for example, an RDBMS or a central system.

EBX allows taking into account other data in the current dataset context.

This is made possible by defining *functions*.

A function is specified in the data model using the `osd:function` element (see example below).

- The value of the *class* attribute must be the qualified name of a Java class that implements the Java interface `ValueFunction`^API^.

- Additional parameters may be specified at the data model level, in which case the JavaBean convention is applied.

Example:

```
<xs:element name="computedValue">
 <xs:annotation>
  <xs:appinfo>
   <osd:function class="com.foo.ComputeValue">
    <param1>...</param1>
    <param...n>...</param...n>
   </osd:function>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

In some cases, it can be useful to disable the validation of computed values if the execution of a function is time-consuming. Indeed, if the function is attached to a table with N records, then it will be called N times when validating this table. The property `osd:disableValidation= "true"` specified in the data model allows to disable the validation of a computed value (see example below).

Example:

```
<xs:element name="computedValue" osd:disableValidation="true">
 <xs:annotation>
  <xs:appinfo>
   <osd:function class="com.foo.ComputeValue">
    <param1>...</param1>
    <param...n>...</param...n>
   </osd:function>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Triggers

Datasets or table records can be associated with methods that are automatically executed when some operations are performed, such as creations, updates, or deletions.

In the data model, these triggers must be declared under the `annotation/appinfo` element using the `osd:trigger` element.

For dataset triggers, a Java class that extends the abstract class `InstanceTrigger`API must be declared inside the element `osd:trigger`.

In the case of dataset triggers, it is advised to define `annotation/appinfo/osd:trigger` tags just under the `root` element of the data model.

Example:

```
<xs:element name="root" osd:access="--">
   ...
   <xs:annotation>
  <xs:appinfo>
   <osd:trigger class="com.foo.MyInstanceTrigger">
    <param1>...</param1>
    <param...n>...</param...n>
   </osd:trigger>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

For the definition of table record triggers, a Java class that extends the abstract class `TableTrigger`API must be defined inside the `osd:trigger` element. It is advised to define the `annotation/appinfo/osd:trigger` elements just under the element describing the associated table or table type.

Examples:

On a table element:

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:appinfo>
   <osd:table>
    <primaryKeys>/key</primaryKeys>
   </osd:table>
   <osd:trigger class="com.foo.MyTableTrigger" />
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

On a table type element:

```
<xs:complexType name="MyTableType">
   ...
   <xs:annotation>
    <xs:appinfo>
    <osd:trigger class="com.foo.MyTableTrigger">
    <param1>...</param1>
    <param...n>...</param...n>
    </osd:trigger>
    </xs:appinfo>
   </xs:annotation>
   ...
</xs:complexType>
```

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol. In the example above, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

Auto-incremented values

It is possible to define auto-incremented values. Auto-incremented values are only allowed inside tables, and they must be of the type `xs:int` or `xs:integer`.

An auto-increment is specified in the data model using the element `osd:autoIncrement` under the element `annotation/appinfo`.

Example:

```
<xs:element name="autoIncrementedValue" type="xs:int">
 <xs:annotation>
  <xs:appinfo>
   <osd:autoIncrement />
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Also, there are two optional elements, `start` and `step`:

- The `start` attribute specifies the first value for this auto-increment. If this attribute is not specified, then the value `1` is set by default.
- The `step` attribute specifies the step for the next value to be generated by the auto-increment. If this attribute is not specified, then the value *1* is set by default.

Example:

```
<xs:element name="autoIncrementedValue" type="xs:int">
 <xs:annotation>
  <xs:appinfo>
   <osd:autoIncrement>
    <start>100</start>
    <step>5</step>
   </osd:autoIncrement>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

A field specifying an `osd:autoIncrement` has the following behavior:

- The computation and allocation of the field value are performed whenever a new record is inserted and the field value is undefined.
- No allocation is performed if a programmatic insertion already specifies a non-`null` value. For example, if an archive import or an XML import specifies the value, that value is preserved.

  Consequently, the allocation is not performed for a record insertion in occulting or overwriting modes.
- A newly allocated value is, whenever possible, unique in the scope of the repository. More precisely, the uniqueness of the allocation spans over all the datasets of the data model, and it also spans over all the dataspaces. The latter case allows the merge of a dataspace into its parent with a reasonable guarantee that there will be no conflict if the `osd:autoIncrement` is part of the records' primary key.

  This principle has a very specific limitation: when a mass update transaction that specifies values is performed at the same time as a transaction that allocates a value on the same field, it is possible that the latter transaction will allocate a value that will be set by the first transaction (there is no locking between different dataspaces).

Internally, the auto-increment value is stored in the 'Auto-increments' table of the repository. In the user interface, it can be accessed by administrators in the 'Administration' area. This field is automatically updated so that it defines the greatest value ever set on the associated `osd:autoIncrement` field, in any instance or dataspace in the repository. This value is computed, taking into account the max value found in the table being updated.

In certain cases, for example when multiple environments have to be managed (development, test, production), each with different auto-increment ranges, it may be required to avoid this "max value" check. This particular behavior can be achieved using the `disableMaxTableCheck` property. It is generally not recommended to enable this property unless it is absolutely necessary, as this could generate conflicts in the auto-increment values. However, this property can be set in the following ways:

- Locally, by setting a parameter element in the auto-increment declaration: `<disableMaxTableCheck>true</disableMaxTableCheck>`,
- For the whole data model, by setting `<osd:autoIncrement disableMaxTableCheck="true"/>` in the element `xs:appinfo` of the data model declaration, or
- Globally, by setting the property `ebx.autoIncrement.disableMaxTableCheck=true` in the EBX main configuration file.

  See TIBCO EBX main configuration file [p 345].

> **Note**
>
> When this option is enabled globally, it becomes possible to create records in the table of auto-increments, for example by importing from XML or CSV. If this option is not selected, creating records in the table of auto-increments is prohibited to ensure the integrity of the repository.

CHAPTER 84 Labels and messages

TIBCO EBX allows to have custom labels and error messages for data models to be displayed in the interface.

This chapter contains the following topics:

84.1 Label and description

A label and a description can be added to each node in an adaptation model.

In EBX, each adaptation node is displayed with its label. If no label is defined, the name of the element is used.

Two different notations can be used:

Full	The label and description are defined by the elements `<osd:label>` and `<osd:description>` respectively.
Simple	The label is extracted from the text content, ending at the first period ('.'), with a maximum of 60 characters. The description uses the remainder of the text.

The description may also have a hyperlink, either a standard HTML `href` to an external document, or a link to another node of the adaptation within EBX.

- When using the `href` notation or any other HTML, it must be properly escaped.
- EBX link notation is not escaped and must specify the path of the target, for example:

  `<osd:link path="../misc1">Link to another node in the adaptation</osd:link>`

Example:

```
<xs:element name="misc1" type="xs:string">
 <xs:annotation>
  <xs:documentation>
```

```
   Miscellaneous 1. This is the description of miscellaneous element #1.
   Click <a href="https://www.tibco.com" target="_blank">here</a>
   to learn more.
  </xs:documentation>
 </xs:annotation>
</xs:element>
<xs:element name="misc2" type="xs:string">
 <xs:annotation>
  <xs:documentation>
   <osd:label>
    Miscellaneous 2
   </osd:label>
   <osd:description>
    This is the miscellaneous element #2 and here is a
    <osd:link path="../misc1"> link to another node in the
     adaptation</osd:link>.
   </osd:description>
  </xs:documentation>
 </xs:annotation>
</xs:element>
```

If a node points to a named type, then the label of the node replaces the label of the named type. The same mechanism applies to the description of the node (element `osd:description`).

> **Note**
>
> Regarding whitespace management, the label of a node is always *collapsed* when displayed. That is, contiguous sequences of blanks are collapsed to a single blank, and leading and trailing blanks are removed. In descriptions, however, whitespaces are always *preserved*.

Dynamic labels and descriptions

As an alternative to statically defining the localized labels and descriptions for each node, it is possible to specify a Java class that programmatically determines the labels and descriptions for the nodes of the data model. To define the class, include the element `osd:documentation`, with the attribute `class` in the data model. It is possible to pass JavaBean properties using nested parameter elements.

Example:

```
<xs:schema ...>
 <xs:annotation>
  <xs:appinfo>
   <osd:documentation class="com.foo.MySchemaDocumentation">
    <param1>...</param1>
    <param2>...</param2>
   </osd:documentation>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:schema ...>
```

The labels and descriptions that are provided programmatically take precedence over the ones defined locally on individual nodes.

See also *`SchemaDocumentation`*API

84.2 **Enumeration labels**

In an enumeration, a simple, non-localized label can be added to each enumeration element, using the attribute `osd:label`.

> **Attention**
>
> Labels defined for an enumeration element are always `collapsed` when displayed.

Example:

```
<xs:element name="Service" maxOccurs="unbounded">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="1" osd:label="Blue" />
   <xs:enumeration value="2" osd:label="Red" />
   <xs:enumeration value="3" osd:label="White" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

It is also possible to fully localize the labels using the standard `xs:documentation` element. If both non-localized and localized labels are added to an enumeration element, the non-localized label will be displayed in any locale that does not have a label defined.

Example:

```
<xs:element name="access" minOccurs="0">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="readOnly">
    <xs:annotation>
     <xs:documentation xml:lang="en-US">
      read only
     </xs:documentation>
     <xs:documentation xml:lang="fr-FR">
      lecture seule
     </xs:documentation>
    </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value="readWrite">
    <xs:annotation>
     <xs:documentation xml:lang="en-US">
      read/write
     </xs:documentation>
     <xs:documentation xml:lang="fr-FR">
      lecture écriture
     </xs:documentation>
    </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value="hidden">
    <xs:annotation>
     <xs:documentation xml:lang="en-US">
      hidden
     </xs:documentation>
     <xs:documentation xml:lang="fr-FR">
      masqué
     </xs:documentation>
    </xs:annotation>
   </xs:enumeration>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

84.3 **Mandatory error message (osd:mandatoryErrorMessage)**

If the node specifies the attribute `minOccurs="1"` (default behavior), then an error message, which must be provided, is displayed if the user does not complete the field. This error message can be defined specifically for each node using the element `osd:mandatoryErrorMessage`.

Example:

```
<xs:element name="birthDate" type="xs:date">
 <xs:annotation>
  <xs:documentation>
   <osd:mandatoryErrorMessage>
    Please give your birth date.
   </osd:mandatoryErrorMessage>
  </xs:documentation>
 </xs:annotation>
</xs:element>
```

The mandatory error message can be localized:

```
<xs:documentation>
 <osd:mandatoryErrorMessage xml:lang="en-US">
  Name is mandatory
 </osd:mandatoryErrorMessage>
 <osd:mandatoryErrorMessage xml:lang="fr-FR">
  Nom est obligatoire
 </osd:mandatoryErrorMessage>
</xs:documentation>
```

> **Note**
>
> Regarding whitespace management, the enumeration labels are always *collapsed* when displayed.

84.4 Conversion error message

For each predefined XML Schema element, it is possible to define a specific error message if the user entry has an incorrect format.

Example:

```
<xs:element name="email" type="xs:string">
 <xs:annotation>
  <xs:documentation>
   <osd:ConversionErrorMessage xml:lang="en-US">
    Please enter a valid email address.
   </osd:ConversionErrorMessage>
   <osd:ConversionErrorMessage xml:lang="fr-FR">
    Saisissez un e-mail valide.
   </osd:ConversionErrorMessage>
  </xs:documentation>
 </xs:annotation>
</xs:element>
```

84.5 Facet validation message with severity

The validation message that is displayed when the value of a field does not comply with a constraint can define a custom severity, a default non-localized message, and localized message variants. If no severity is specified, the default level is `error`. Blocking constraints *must* have the severity `error`.

XML Schema facet (osd:validation)

The validation message is described by the element `osd:validation` in `annotation/appinfo` under the definition of the facet.

Example:

```
<xs:element name="zipCode">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <!--facet is not localized, but validation message is localized-->
   <xs:minInclusive value="01000">
    <xs:annotation>
     <xs:appinfo>
      <osd:validation>
       <severity>error</severity>
       <message>Non-localized message.</message>
       <message xml:lang="en-US">English error message.</message>
       <message xml:lang="fr-FR">Message d'erreur en français.</message>
      </osd:validation>
     </xs:appinfo>
    </xs:annotation>
   </xs:minInclusive>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

XML Schema enumeration facet (osd:enumerationValidation)

The validation message is described by the element `osd:enumerationValidation` in `annotation/appinfo` under the definition of the field.

Example:

```
<xs:element name="Gender">
 <xs:annotation>
  <xs:appinfo>
   <osd:enumerationValidation>
    <severity>error</severity>
    <message>Non-localized message.</message>
    <message xml:lang="en-US">English error message.</message>
    <message xml:lang="fr-FR">Message d'erreur en français.</message>
   </osd:enumerationValidation>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="0" osd:label="male" />
   <xs:enumeration value="1" osd:label="female" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

EBX facet (osd:validation)

The validation message is described by the element `osd:validation` under the definition of the facet (which is defined in `annotation/appinfo/otherFacets`).

Example:

```
<xs:element name="price" type="xs:decimal">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:minInclusive path="../priceMin">
     <osd:validation>
      <severity>error</severity>
      <message>Non-localized message.</message>
      <message xml:lang="en-US">English error message.</message>
      <message xml:lang="fr-FR">Message d'erreur en français.</message>
     </osd:validation>
    </osd:minInclusive>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

CHAPTER 85

Additional properties

This chapter contains the following topics:

85.1 Default values

In a data model, it is possible to specify a default value for a field using the attribute `default`. This property is used to assign a default value if no value is defined for a field.

The default value is displayed in the user input field at the creation time. That is, the default value will be displayed when creating a new record or adding a new occurrence to an aggregated list.

Example:

In this example, the element specifies a default string value.

```
<xs:element name="fieldWithDefaultValue" type="xs:string" default="aDefaultValue" />
```

85.2 Access properties

The attribute `osd:access` defines the access mode, that is, whether the data of a particular data model node can be read and/or written. This attribute must have one of the following values: `RW`, `R-`, `CC` or `--`.

For each XML Schema node, three types of adaptation are possible:

1. *Adaptation terminal node*

   This node is displayed with an associated value in TIBCO EBX. When accessed using the method `Adaptation.get()`, it uses the adaptation search algorithm.

2. *Adaptation non-terminal node*

This node is a complex type that is only displayed in EBX if it has one child node that is also an adaptation terminal node. It has no value of its own. When accessed using the method `Adaptation.get()`, it returns `null`.

3. *Non-adaptable node*

   This node is not an adaptation terminal node and has no child adaptation terminal nodes. This node is never displayed in EBX. When accessing using the method `Adaptation.get()`, it returns the node default value if one is defined, otherwise it returns `null`.

See also`Adaptation`API

Access mode	Behavior
`RW`	*Adaptation terminal node*: value can be read and written in EBX.
`R-`	*Adaptation terminal node*: value can only be read in EBX.
`CC`	*Cut*: This is not an adaptation terminal node and none of its children are adaptation terminal nodes. This "instruction" has priority over any child node regardless of the value of their *access* attribute.
`--`	If the node is a simple type, it is not adaptable. If the node is a complex type, it is not an adaptation terminal node and does not define any child nodes. The `root` node of a data model must specify this access mode.
Default	If the *access* attribute is not defined: • If the node is a computed value, it is considered to be `R-` • If the node is a simple type and its value is not computed, it is considered to be `RW` • If the node is an aggregated list, it is then a terminal value and is considered to be `RW` • Otherwise, it is not an adaptation terminal node and it does not define anything about its child nodes.

Example:

In this example, the element is adaptable because it is an adaptation terminal node.

```
<xs:element name="proxyIpAddress" type="xs:string" osd:access="RW"/>
```

85.3 Information

The element `osd:information` allows specifying additional information. This information can then be used by the integration code, for any purpose, by calling the method `SchemaNode.getInformation`API.

Example:

```
<xs:element name="misc" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
   <osd:information>
    This is the text information of miscellaneous element.
   </osd:information>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

85.4 **Default view**

Hiding a field or a table in the default view

It is possible for a table or field inside a table to be hidden by default in EBX by using the element `osd:defaultView/hidden`. This property is used to hide elements from the default view of a dataset without defining specific access permissions. That is, elements hidden by default will not be visible in any default forms and views, whether tabular or hierarchical, generated from the structure of the associated data model.

Attention

- If an element is configured to be hidden in the default view of a dataset, then the access permissions associated with this field will not be evaluated.
- It is possible to display a field that is hidden in the default view of a dataset by defining a view. Only in this case will the access permissions associated with this field be evaluated to determine whether the field will be displayed or not.
- It is not possible to display a table that is hidden in the default view of a dataset (in the navigation pane).

Example:

In this example, the element is hidden in the default view of a dataset.

```
<xs:element name="hiddenField" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hidden>true</hidden>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

Hiding groups and fields in views

It is possible for a field or a group to be hidden in all views of a table by using the element `osd:defaultView/hiddenInViews`. This property is used to hide elements from the tabular (including the default tabular view) and hierarchical views of a dataset without defining specific access permissions. That is, hidden elements will not be visible in any views, whether tabular or hierarchical, created from the structure of the associated data model. However, hidden elements in views will be displayed in forms.

To specify whether or not to hide an element in all views, use the `osd:defaultView/hiddenInViews="true|false"` element.

If this property is set to `true`, then the element will not be selectable when creating a custom view. As a consequence, the element will not be displayed in all views of a table in a dataset.

If a group is configured as hidden in views, then all the fields nested under this group will not be displayed respectively in the views of the table.

Hiding a field in search tools

To specify whether or not to hide an element in search tools, use the element `osd:defaultView/hiddenInSearch="true|false|textSearchOnly"`.

If this property is set to `true`, then the field will not be selectable in the text and typed search tools of a dataset.

If this property is set to `textSearchOnly`, then the field will not be selectable only in the text search of a dataset but will be selectable in the typed search.

> **Note**
>
> If a group is configured as hidden in search tools or only in the text search, then all the fields nested under this group will not be displayed respectively in the search tools or only in the text search.

Example:

```
<xs:element name="hiddenFieldInSearch" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hiddenInSearch>true</hiddenInSearch>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the text and typed search tools of a dataset.

```
<xs:element name="hiddenFieldOnlyInTextSearch" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hiddenInSearch>textSearchOnly</hiddenInSearch>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

In this example, the element is hidden only in the text search tool of a dataset.

Hiding a field in Data Services

To specify whether or not to hide an element in data services, use the element `osd:defaultView/hiddenInDataServices`. For more information, see Disabling fields from data model [p 615].

> **Note**
>
> - If a group is configured as being hidden, then all the fields nested under this group will be considered as hidden by data services.

Example:

```
<xs:element name="hiddenFieldInDataService" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hiddenInDataServices>true</hiddenInDataServices>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the Data Service select operation.

Defining a view for the combo box selector of a foreign key

It is possible to specify a published view that will be used to display the target table or the hierarchical view of a foreign key for a smoother selection. If a view has been defined, the selector will be displayed in the user interface in the combo box of this foreign key. The definition of a view can be done by using the XML Schema element `osd:defaultView/widget/viewForAdvancedSelection`.

> **Note**
>
> - This property can only be defined on foreign key fields.
> - The published view must be associated with the target table of the foreign key.
> - If the published view does not exist, then the advanced selection is not available in the foreign key field.

Example:

In this example, the name of a published view is defined to display the target table of a foreign key in the advanced selection.

```
<xs:element name="catalog_ref" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:tableRef>
     <tablePath>/root/Catalogs</tablePath>
    </osd:tableRef>
     </osd:otherFacets>
    <osd:defaultView>
    <widget>
     <viewForAdvancedSelection>catalogView</viewForAdvancedSelection>
    </widget>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

See Combo-box selector [p 57] for more information.

Customizing a default widget

A widget can be defined using the data model assistant. See Default view > Widget [p 57] for more information.

85.5 **Comparison mode**

The attribute `osd:comparison` can be included on a terminal node element in order to set its comparison mode. This mode controls how differences are detected for the element during comparisons. The possible values for the attribute are:

default	Element is visible during comparisons of its data.
ignored	No changes will be detected when comparing two versions of the content in records or datasets. During a merge, the data values of an ignored element are not merged when contents are updated, even if the values are different. For new content, the values of ignored elements are merged. During an archive import, values of ignored elements are not imported when contents are updated. For new content, the values of ignored elements are imported.

> **Note**
>
> - If a group is configured as being ignored during comparisons, then all the fields nested under this group will also be ignored.
> - If a terminal field does not include the attribute `osd:comparison`, then it will be included by default during comparisons.
>
> **Restrictions:**
>
> - This property cannot be defined on non-terminal fields.
> - Primary key fields cannot be ignored during comparison.

Example:

In this example, the first element is explicitly ignored during comparison, the second element is explicitly included.

```
<xs:element name="fieldExplicitlyIgnoredInComparison"
 type="xs:string" minOccurs="0" osd:comparison="ignored"/>
<xs:element name="fieldExplicitlyNotIgnoredInComparison"
 type="xs:string" minOccurs="0" osd:comparison="default"/>
```

85.6 **Apply last modifications policy**

The attribute `osd:applyLastModification` can be defined on a terminal node element in order to specify if this element must be included or not in the 'apply last modifications' service that can be executed in a table of a dataset.

The possible values for the attribute are:

default	Last modifications can be applied to this element.
ignored	This element is ignored from the apply last modifications service. That is, the last modification that has been performed on this element cannot be applied to other records.

Note

- If a group is configured as being ignored by the 'apply last modifications' service, then all fields nested under this group will also be ignored.
- If a terminal field does not include the attribute `osd:applyLastModification`, then it will be included by default in the apply last modifications service.

Restriction:

- This property cannot be defined on non-terminal fields.

Example:

In this example, the first element is explicitly ignored in the 'apply last modifications' service, the second element is explicitly included.

```
<xs:element name="fieldExplicitlyIgnoredInApplyLastModification"
 type="xs:string" minOccurs="0" osd:applyLastModification="ignored"/>
<xs:element name="fieldExplicitlyNotIgnoredApplyLastModification"
 type="xs:string" minOccurs="0" osd:applyLastModification="default"/>
```

85.7 Categories

Categories can be used for "filtering", by restricting the display of data model elements.

To create a category, add the attribute `osd:category` to a table node in the data model XSD.

Filters on data

In the example below, the attribute `osd:category` is added to the node in order to create a category named *mycategory*.

```
<xs:element name="rebate" osd:category="mycategory">
   <xs:complexType>
  <xs:sequence>
    <xs:element name="label" type="xs:string"/>
    <xs:element name="beginDate" type="xs:date"/>
    <xs:element name="endDate" type="xs:date"/>
    <xs:element name="rate" type="xs:decimal"/>
  </xs:sequence>
   </xs:complexType>
</xs:element>
```

To activate a defined category filter on a dataset in the user interface, select **Actions** > **Categories** > **<*category name*>** from the navigation pane.

Predefined categories

Two categories with localized labels are predefined:

- Hidden

  An instance node, including a table node itself, is hidden in the default view, but can be revealed by selecting **Actions > Categories >** ***[hidden nodes]*** from the navigation pane.

  A table record node is always hidden.

- Constraint (deprecated)

Restriction

Categories do not apply to table record nodes, except the category 'Hidden'.

CHAPTER 86 Data services

This chapter details how WSDL operations' names related to a table are defined and managed by TIBCO EBX.

This chapter contains the following topics:

86.1 Definition

EBX generates a WSDL that complies with the W3C Web Services Description Language 1.1 standard. By default, WSDL operations refer to a table using the last element of the table path. A WSDL operation name is composed of the action name (prefix) and the table name (suffix). It is possible to refer to tables in WSDL operations using unique names instead of the last element of their paths by overriding the suffix operations' names.

See also*Data services using the Data Model Assistant [p 43]*

86.2 Configuration

Embedded data model

WSDL suffix operations' names are embedded in EBX's repository and linked to a publication. That is, when publishing an embedded data model, the list of WSDL suffix operations' names can be defined in the data model definition, under the 'Configuration > Data services' table and managed by EBX.

Packaged data model

WSDL suffix operations' names are defined in a dedicated `XML` document file and must be named as the data model and end with the keyword `_entities`. For instance, if a data model is named `catalog.xsd`, then the XML document containing the configuration of the WSDL operations' names overrided will be named `catalog_entities.xml`. This `XML` document must also be located in the same location as the data model. The `XML` document is automatically loaded by EBX if a file that matches this pattern is found when compiling a data model.

86.3 **Publication**

The suffix operations' names are validated at compilation time and contain a list of couples containing `Path` with a unique table name. Checked validation rules are:

- The `path` is not unique,
- The table name contains a syntax error,
- The table name is not unique in the `XML` document.

86.4 **WSDL and table operations**

WSDL Generator

An additional validation rule has been added: a unicity check is systematically applied to table names. The SOAP operation name is composed of the operation type as a prefix and, by default, of the table name (last step of the table path) as a suffix. A dataset can contain several identical table names but with different paths. It is possible to override table names that are not unique in order to guarantee the unicity.

SOAP operations

When an operation request on table has been invoked from the SOAP connector, the target table is retrieved by priority, the name corresponds to:

1. an overridden table name,
2. the last step of the table path.

See also*Data services* *[p 596]*

86.5 **Limitations**

WSDL operations' names are not available with `external` data models.

CHAPTER 87
Toolbars

This chapter details how toolbars are defined and managed by TIBCO EBX.

This chapter contains the following topics:

87.1 Definition

Toolbars allow to customize the buttons and menus to display when accessing a table view, a hierarchical view, or a record form.

Toolbars can only be created and published using the *Data Model Assistant* and are available only on `embedded` and `packaged` data models.

For embedded data models, toolbars are embedded in EBX's repository and linked to a publication. That is, when publishing an embedded data model, the toolbars defined in the data model are embedded with the publication of the data model and managed by EBX.

For packaged data models, toolbars are defined in a dedicated `XML` document and must be named as the data model and end with the keyword `_toolbars`. For instance, if a data model is named `catalog.xsd` then the XML document containing the definition of the toolbars must be named `catalog_toolbars.xml`. This `XML` document must also be placed in the same location as the data model. The toolbar document is automatically loaded by EBX if a file complying with this pattern is found when compiling a data model.

See also

Configuring toolbars using the Data Model Assistant *[p 75]*

Using toolbars in data models *[p 497]*

Toolbar API `ToolbarFactory`API

87.2 Using toolbars

Toolbars can be used on `tables` and `associations.`

On tables, it is possible to specify the toolbar to display:

- On the top of a tabular view
- On each row of a tabular view

- On the top of a record form
- On the top of a hierarchical view.

On associations, it is possible to specify the toolbar to display:

- On top of the tabular view of the association
- On each row of the tabular view of the association

See also

Using toolbars [p 497]

Associations [p 501]

CHAPTER 88

Workflow model

The workflow offers two types of steps: 'library' or 'specific'.

'Library' is a bean defined in `module.xml` and is reusable. Using the 'library' bean improves the ergonomics: parameters are dynamically displayed in the definition screens.

A 'specific' object is a bean defined only by its class name. In this case, the display is not dynamic.

This chapter contains the following topics:

88.1 Bean categories

Step	**Library**	Specific
Scripts	ScriptTaskBean	ScriptTask
Conditions	ConditionBean	Condition
User task	UserTask	

88.2 **Sample of ScriptTask**

Java Code

A script task has to override the method `execute` as in the following example:

```
public class NppScriptTask_CreateWorkingBranch extends ScriptTask
{
    public void executeScript(ScriptTaskContext aContext) throws OperationException
    {
        Repository repository = aContext.getRepository();
        String initialBranchString = aContext.getVariableString("initialBranch");
        AdaptationHome initialBranch = repository.lookupHome(HomeKey.forBranchName(initialBranchString));
        if (initialBranch == null)
            throw OperationException.createError("Null value for initialBranch");

        HomeCreationSpec spec = new HomeCreationSpec();
        spec.setParent(initialBranch);
        spec.setKey(HomeKey.forBranchName("Name"));
        spec.setOwner(Profile.EVERYONE);
        spec.setHomeToCopyPermissionsFrom(initialBranch);
        AdaptationHome newHome = repository.createHome(spec, aContext.getSession());
                //feeds dataContext
        aContext.setVariableString("workingBranch", newHome.getKey().getName());
    }
}
```

See also***com.orchestranetworks.workflow.ScriptTask*** `ScriptTask`API

88.3 **Sample of ScriptTaskBean**

Java Code

A script task bean has to override the method `executeScript` as in the following example:

```
public class ScriptTaskBean_CreateBranch extends ScriptTaskBean
{
    private String initialBranchName;

    private String newBranch;

    public String getInitialBranchName()
    {
        return this.initialBranchName;
    }

    public void setInitialBranchName(String initialBranchName)
    {
        this.initialBranchName = initialBranchName;
    }

    public String getNewBranch()
    {
        return this.newBranch;
    }

    public void setNewBranch(String newBranch)
    {
        this.newBranch = newBranch;
    }

    public void executeScript(ScriptTaskBeanContext aContext) throws OperationException
    {
        final Repository repository = aContext.getRepository();

        String initialBranchName = this.getInitialBranchName();
        final AdaptationHome initialBranch = repository.lookupHome(HomeKey.forBranchName(initialBranchName));
        final HomeCreationSpec spec = new HomeCreationSpec();
        spec.setParent(initialBranch);
        spec.setKey(HomeKey.forBranchName(XsFormats.SINGLETON.formatDateTime(new Date())));
        spec.setOwner(Profile.EVERYONE);
        spec.setHomeToCopyPermissionsFrom(initialBranch);
        final AdaptationHome branchCreate = repository.createHome(spec, aContext.getSession());
        this.setNewBranch(branchCreate.getKey().getName());
    }
}
```

See also ***com.orchestranetworks.workflow.ScriptTaskBean*** `ScriptTaskBean`API

Configuration through module.xml

A script task bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.genericScriptTask.ScriptTaskBean_CreateBranch">
            <documentation xml:lang="fr-FR">
                <label>Créer une branche</label>
                <description>
                    Ce script permet de créer une branche
                </description>
            </documentation>
            <documentation xml:lang="en-US">
                <label>Create a branch</label>
                <description>
                    This script creates a branch
                </description>
            </documentation>
            <properties>
                <property name="initialBranchName" input="true">
                    <documentation xml:lang="fr-FR">
                        <label>Branche initiale</label>
                        <description>
                            Nom de la branche initiale.
                        </description>
                    </documentation>
                    <documentation xml:lang="en-US">
                        <label>Initial branch</label>
                        <description>
                            Initial branch name.
                        </description>
                    </documentation>
                </property>
                <property name="newBranch" output="true">
                    <documentation xml:lang="fr-FR">
                        <label>Nouvelle branche</label>
                        <description>
                            Nom de la branche créée
                        </description>
                    </documentation>
                    <documentation xml:lang="en-US">
                        <label>New branch</label>
                        <description>
                            Created branch name.
                        </description>
                    </documentation>
                </property>
            </properties>
        </bean>
    </beans>
</module>
```

88.4 Samples of UserTask

Service declaration via module.xml

A built-in service can be declared in `module.xml` to be used in the user task definition.

```
 <services>
  <service name="ServiceModule">
     <resourcePath>/service.jsp</resourcePath>
     <type>branch</type>
        <documentation xml:lang="fr-FR">
            <label>Workflow service</label>
               <description>
                Ce service permet de ...
               </description>
  </documentation>
           <documentation xml:lang="en-US">
```

```
            <label>Service workflow</label>
               <description>
                The purpose of this service is ...
               </description>
  </documentation>
            <properties>
            <property name="param1" input="true">
                 <documentation xml:lang="fr-FR">
                     <label>Param1</label>
                     <description>Param1 ...</description>
                 </documentation>
            </property>
            <property name="param2" output="true">
            </property>
        </properties>
 </service>
    <serviceLink serviceName="adaptationService">
        <importFromSchema>
            /WEB-INF/ebx/schema/schema.xsd
        </importFromSchema>
    </serviceLink>
</services>
```

A more complex UserTask

The GUI is quite similar as the example above. The field 'Rule' must be filled to define the class extending the 'UserTask' to invoke.

```
public class NppUserTask_ValidateProduct extends UserTask
{
    public void handleWorkItemCompletion(UserTaskWorkItemCompletionContext context)
        throws OperationException
    {
        if (context.getCompletedWorkItem().isRejected())
        {
            context.setVariableString(NppConstants.VAR_VALIDATION, "KO");
            context.completeUserTask();
        }
        else if (context.checkAllWorkItemMatchStrategy())
        {
            context.setVariableString(NppConstants.VAR_VALIDATION, "OK");
            context.completeUserTask();
        }
    }

    public void handleCreate(UserTaskCreationContext context) throws OperationException
    {
        CreationWorkItemSpec spec = CreationWorkItemSpec.forOfferring(NppConstants.ROLE_PVALIDATOR);
        spec.setNotificationMail("1");
        context.createWorkItem(spec);
        context.setVariableString(NppConstants.VAR_VALIDATION, "validating");
    }
}
```

See also***com.orchestranetworks.workflow.UserTask*** `UserTask`API

88.5 Samples of Condition

Java Code

The method `evaluate` has to be overridden:

```
public class NppCondition_IsValidationOK extends Condition
{
    public boolean evaluateCondition(ConditionContext context) throws OperationException
    {
        String validation = context.getVariableString("validationResult");
        boolean hasError = "KO".equals(validation);
        return !hasError;
    }
}
```

See also***com.orchestranetworks.workflow.Condition*** `Condition` API

88.6 Sample of ConditionBean

Java Code

The method `evaluateCondition` has to be overridden as in the following sample:

```
public class ConditionBean_IsBranchValid extends ConditionBean
{
    private String branchName;

    public String getBranchName()
    {
        return this.branchName;
    }

    public void setBranchName(String branchName)
    {
        this.branchName = branchName;
    }

    public boolean evaluateCondition(ConditionBeanContext aContext) throws OperationException
    {
        final Repository repository = aContext.getRepository();
        Severity severityForValidation = Severity.ERROR;
        String branchToTestName = this.getBranchName();
        final AdaptationHome branchToTest = repository.lookupHome(HomeKey.forBranchName(branchToTestName));
        if (branchToTest.getValidationReportsMap(severityForValidation) != null
            && branchToTest.getValidationReportsMap(severityForValidation).size() > 0)
        {
            return false;
        }
        return true;
    }
}
```

See also***com.orchestranetworks.workflow.ConditionBean*** `ConditionBean` API

Configuration through module.xml

The condition bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.genericScriptTask.ConditionBean_IsBranchValid">
            <documentation xml:lang="fr-FR">
                <label>Branche valide ?</label>
                <description>
                    Ce script permet de tester si une branche est valide.
                </description>
            </documentation>
            <documentation xml:lang="en-US">
                <label>Branch valid ?</label>
                <description>
                    This script allows to check if a branch is valid.
                </description>
            </documentation>
            <properties>
                <property name="branchName" input="true">
                    <documentation xml:lang="fr-FR">
                        <label>Branche à contrôler</label>
                        <description>
                            Nom de la branche à valider.
                        </description>
                    </documentation>
                    <documentation xml:lang="en-US">
                        <label>Branch to check</label>
                        <description>
                            Branch name to check.
                        </description>
                    </documentation>
                </property>
            </properties>
        </bean>
    </beans>
</module>
```

88.7 Sample of SubWorkflowsInvocationBean

Java Code

```
public class MySubWorkflowsInvocationBean extends SubWorkflowsInvocationBean
{
 @Override
 public void handleCreateSubWorkflows(SubWorkflowsCreationContext aContext)
  throws OperationException
 {
  final ProcessLauncher subWorkflow1 = aContext.registerSubWorkflow(
   AdaptationName.forName("validateProduct"),
   "validateProduct1");
  subWorkflow1.setLabel(UserMessage.createInfo("Validate the new product by marketing"));
  subWorkflow1.setInputParameter("workingBranch", aContext.getVariableString("workingBranch"));
  subWorkflow1.setInputParameter("code", aContext.getVariableString("code"));
  subWorkflow1.setInputParameter("service", aContext.getVariableString("marketing"));

  final ProcessLauncher subWorkflow2 = aContext.registerSubWorkflow(
   AdaptationName.forName("validateProduct"),
   "validateProduct2");
  subWorkflow2.setLabel(UserMessage.createInfo("Validate the new product by direction"));
  subWorkflow2.setInputParameter("workingBranch", aContext.getVariableString("workingBranch"));
  subWorkflow2.setInputParameter("code", aContext.getVariableString("code"));
  subWorkflow2.setInputParameter("service", aContext.getVariableString("direction"));

  // Conditional launching.
  if (aContext.getVariableString("productType").equals("book"))
  {
   final ProcessLauncher subWorkflow3 = aContext.registerSubWorkflow(
    AdaptationName.forName("generateISBN"),
    "generateISBN");
   subWorkflow3.setLabel(UserMessage.createInfo("Generate ISBN"));
   subWorkflow3.setInputParameter(
    "workingBranch",
    aContext.getVariableString("workingBranch"));
   subWorkflow3.setInputParameter("code", aContext.getVariableString("code"));
  }

  aContext.launchSubWorkflows();
 }
 @Override
 public void handleCompleteAllSubWorkflows(SubWorkflowsCompletionContext aContext)
  throws OperationException
 {
  aContext.getCompletedSubWorkflows();
  final ProcessInstance validateProductMarketing = aContext.getCompletedSubWorkflow("validateProduct1");
  final ProcessInstance validateProductDirection = aContext.getCompletedSubWorkflow("validateProduct2");
  if (aContext.getVariableString("productType").equals("book"))
  {
   final ProcessInstance generateISBN = aContext.getCompletedSubWorkflow("generateISBN");
   aContext.setVariableString("isbn", generateISBN.getDataContext().getVariableString(
    "newCode"));
  }

  if (validateProductMarketing.getDataContext().getVariableString("Accepted").equals("true")
   && validateProductDirection.getDataContext().getVariableString("Accepted").equals(
    "true"))
   aContext.setVariableString("validation", "ok");
 }
}
```

See also ***com.orchestranetworks.workflow.SubWorkflowsInvocationBean***
*`SubWorkflowsInvocationBean`*API

Configuration through module.xml

`SubWorkflowsInvocationBean` bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.test.MySubWorkflowsInvocationBean"/>
    </beans>
```

```
</module>
```

88.8 Sample of WaitTaskBean

Java Code

```
public class MyWaitTaskBean extends WaitTaskBean
{
 @Override
 public void onStart(WaitTaskOnStartContext aContext)
 {
  Map<String, String> params = new HashMap<String, String>();
  params.put("resumeId", aContext.getResumeId());
  myMethod.callWebService(params);
 }

 @Override
 public void onResume(WaitTaskOnResumeContext aContext) throws OperationException
 {
  // Defines a specific mapping.
  aContext.setVariableString("code", aContext.getOutputParameters().get("isbn"));
  aContext.setVariableString("comment", aContext.getOutputParameters().get("isbnComment"));
 }
}
```

See also*com.orchestranetworks.workflow.WaitTaskBean* *`WaitTaskBean`*API

Configuration through module.xml

`WaitTaskBean` bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.test.MyWaitTaskBean"/>
    </beans>
</module>
```

88.9 Sample of ActionPermissionsOnWorkflow

Java Code

```
package com.orchestranetworks.workflow.test;

import com.orchestranetworks.service.*;
import com.orchestranetworks.workflow.*;
import com.orchestranetworks.workflow.ProcessExecutionContext.*;

/**
 */
public class MyDynamicPermissions extends ActionPermissionsOnWorkflow
{

    public ActionPermission getActionPermission(
        WorkflowPermission aWorkflowAction,
        ActionPermissionsOnWorkflowContext aContext)
    {
        if (WorkflowPermission.VIEW.equals(aWorkflowAction)
            || WorkflowPermission.CREATE_PROCESS.equals(aWorkflowAction))
            return ActionPermission.getEnabled();
        return ActionPermission.getDisabled();
    }

}
```

See also*com.orchestranetworks.workflow.ActionPermissionsOnWorkflow* *`ActionPermissionsOnWorkflow`*API

Configuration through module.xml

`ActionPermissionsOnWorkflow` bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.test.MyDynamicPermissions"/>
    </beans>
</module>
```

88.10 Sample of WorkflowTriggerBean

Java Code

```
public class MyWorkflowTriggerBean extends WorkflowTriggerBean
{
 @Override
 public void handleAfterProcessInstanceStart(
  WorkflowTriggerAfterProcessInstanceStartContext aContext) throws OperationException
 {
  final DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());
  final MailSpec spec = aContext.createMailSpec();
  spec.notify(NotificationType.TO, "supervisor@mail.com");

  spec.setSubject("[TRIGGER] After process instance start");
  spec.setBody("The workflow '"
   + policy.formatUserMessage(aContext.getProcessInstance().getLabel())
   + "' has been created.");

  spec.sendMail(Locale.US);
 }

 @Override
 public void handleBeforeProcessInstanceTermination(
  WorkflowTriggerBeforeProcessInstanceTerminationContext aContext) throws OperationException
 {
  final DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

  final MailSpec spec = aContext.createMailSpec();
  spec.notify(NotificationType.TO, "supervisor@mail.com");

  spec.setSubject("[TRIGGER] Before process instance termination");
  spec.setBody("The workflow '"
   + policy.formatUserMessage(aContext.getProcessInstance().getLabel())
   + "' has been completed. The created product is: '"
   + aContext.getVariableString(NppConstants.VAR_CODE) + "'.");

  spec.sendMail(Locale.US);
 }

 @Override
 public void handleAfterWorkItemCreation(WorkflowTriggerAfterWorkItemCreationContext aContext)
  throws OperationException
 {
  DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

  MailSpec spec = aContext.createMailSpec();
  spec.notify(NotificationType.TO, "supervisor@mail.com");

  spec.setSubject("[TRIGGER] After work item creation");
  WorkItem workItem = aContext.getWorkItem();
  State state = workItem.getState();
  String body = "The work item '" + policy.formatUserMessage(workItem.getLabel())
   + "' has been created. \n The step id is : " + aContext.getCurrentStepId()
   + ". \n The work item is in state : " + policy.formatUserMessage(state.getLabel());

  if (workItem.getOfferedTo() != null)
   body += "\n The role is :" + workItem.getOfferedTo().format();
  if (workItem.getUserReference() != null)
   body += "\n The user is :" + workItem.getUserReference().format();

  spec.setBody(body);

  spec.sendMail(Locale.US);
 }

 @Override
```

```
public void handleBeforeWorkItemStart(WorkflowTriggerBeforeWorkItemStartContext aContext)
 throws OperationException
{
 DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

 MailSpec spec = aContext.createMailSpec();
 spec.notify(NotificationType.TO, "supervisor@mail.com");

 spec.setSubject("[TRIGGER] Before work item start");
 spec.setBody("The work item '"
  + policy.formatUserMessage(aContext.getWorkItem().getLabel())
  + "' has been started. \n  The current step id is : "
  + aContext.getCurrentStepId()
  + ". \n The work item user is: '"
  + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
   aContext.getWorkItem().getUserReference(),
   aContext.getSession().getLocale()) + "'.");

 spec.sendMail(Locale.US);
}

@Override
public void handleBeforeWorkItemAllocation(
 WorkflowTriggerBeforeWorkItemAllocationContext aContext) throws OperationException
{
 DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

 MailSpec spec = aContext.createMailSpec();
 spec.notify(NotificationType.TO, "supervisor@mail.com");

 spec.setSubject("[TRIGGER] Before work item allocation");
 spec.setBody("The work item '"
  + policy.formatUserMessage(aContext.getWorkItem().getLabel())
  + "' has been allocated. \n  The current step id is: "
  + aContext.getCurrentStepId()
  + ". \n  The work item user is: '"
  + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
   aContext.getUserReference(),
   aContext.getSession().getLocale()) + "'.");

 spec.sendMail(Locale.US);
}

@Override
public void handleBeforeWorkItemDeallocation(
 WorkflowTriggerBeforeWorkItemDeallocationContext aContext) throws OperationException
{
 DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

 MailSpec spec = aContext.createMailSpec();
 spec.notify(NotificationType.TO, "supervisor@mail.com");

 spec.setSubject("[TRIGGER] Before work item deallocation");
 spec.setBody("The work item '"
  + policy.formatUserMessage(aContext.getWorkItem().getLabel())
  + "' has been deallocated. \n  The current step id is: "
  + aContext.getCurrentStepId()
  + ". \n  The old work item user is: '"
  + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
   aContext.getWorkItem().getUserReference(),
   aContext.getSession().getLocale()) + ".");

 spec.sendMail(Locale.US);

}

@Override
public void handleBeforeWorkItemReallocation(
 WorkflowTriggerBeforeWorkItemReallocationContext aContext) throws OperationException
{
 DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

 MailSpec spec = aContext.createMailSpec();
 spec.notify(NotificationType.TO, "supervisor@mail.com");

 spec.setSubject("[TRIGGER] Before work item reallocation");
 spec.setBody("The work item '"
  + policy.formatUserMessage(aContext.getWorkItem().getLabel())
  + "' has been reallocated. \n  The current step id is: "
  + aContext.getCurrentStepId()
  + ". \n  The work item user is: '"
  + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
   aContext.getUserReference(),
   aContext.getSession().getLocale())
  + "'. The old work item user is: '"
```

```
    + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
     aContext.getWorkItem().getUserReference(),
     aContext.getSession().getLocale()) + "'.");

   spec.sendMail(Locale.US);

  }
  @Override
  public void handleBeforeWorkItemTermination(
   WorkflowTriggerBeforeWorkItemTerminationContext aContext) throws OperationException
  {
   DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

   MailSpec spec = aContext.createMailSpec();
   spec.notify(NotificationType.TO, "supervisor@mail.com");

   spec.setSubject("[TRIGGER] Before work item termination");
   spec.setBody("The work item '"
    + policy.formatUserMessage(aContext.getWorkItem().getLabel())
    + "' has been terminated. \n  The current step id is: " + aContext.getCurrentStepId()
    + ". \n  The work item has been accepted ? " + aContext.isAccepted());

   spec.sendMail(Locale.US);
  }
}
```

See also ***com.orchestranetworks.workflow.WorkflowTriggerBean*** `WorkflowTriggerBean`API

Configuration through module.xml

`WorkflowTriggerBean` bean must be declared in `module.xml`:

```
<module>
    <beans>
        <bean className="com.orchestranetworks.workflow.test.MyWorkflowTriggerBean"/>
    </beans>
</module>
```

88.11 Sample of trigger starting a process instance

Sample

```
public class TriggerWorkflow extends TableTrigger
{
    public void handleAfterModify(AfterModifyOccurrenceContext aContext) throws OperationException
    {
        ValueContext currentRecord = aContext.getOccurrenceContext();
        String code = (String) currentRecord.getValue(Path.parse("/code"));

        //Get published process
        PublishedProcessKey processPublishedKey = PublishedProcessKey.forName("productProcess");
                //Defines process instance
            ProcessLauncher launcher = ProcessLauncherHelper.createLauncher(
            processPublishedKey,
            aContext.getProcedureContext());
        //initialize Data Context
        launcher.setInputParameter("code", "/root/Client[./code=\"" + code + "\"]");
        launcher.setInputParameter("workingBranch", aContext.getAdaptationHome().getKey().getName());

        //Starts process
        launcher.launchProcess();

    }
      //...
}
```

User interface

CHAPTER 89 Interface customization

The TIBCO EBX graphical interface can be customized through various EBX APIs.

This chapter contains the following topics:

89.1 How to embed a Web Component

EBX can be integrated into any application that is accessible through a supported web browser, thanks to the Web Component API.

To embed all or part of EBX in a web page, the HTML tag `<iframe>` should be used by indicating the URL to EBX. This URL can be specified either manually or by using the `UIHttpManagerComponent` API. A single web page may include several iframes that integrate EBX. It is then possible to create a portal made of tables, forms, hierarchical views, etc., from EBX.

See also*Using TIBCO EBX as a Web Component* *[p 193]*

89.2 User services

A user service is an extension of EBX that provides a graphical user interface (GUI) that allows users to access specific or advanced functions.

Powerful custom user services can be developed using the same visual components and data validation mechanisms as standard EBX user interfaces.

See also*User service overview* *[p 563]*

89.3 **Form layout**

It is possible to override the default layout of forms in the user interface by highly customizing it with the `UIForm` API. This API provides the standard input components from EBX, which give the possibility to customize the layout of a form, while having the same components and standard behavior as record forms using the default layout.

See also

*`UIForm`*API

*`UIFormPaneWriter`*API

*`UIWidget`*API

*`UIFormHeader`*API

*`UIFormBottomBar`*API

89.4 **Custom widgets**

Custom widgets are included in the Java API to allow the development of user interface components for fields or groups of fields. A custom widget (`UICustomWidget`) allows, for a given node, to control the area where the input or display component is located. This allows having an input and display component that is fully customizable in HTML. The standard components (`UIWidgets`) are available and can be used. The custom widget can implement several display aspects: input component in the form, display in the form, display in a table cell. If a custom widget writes its own HTML components, it has the possibility to save the value in the database when submitting the form.

See also*`UICustomWidget`*API

89.5 **Table filter**

A table filter allows, for a given table, to create a criteria input form in order to apply a filter to the table view. The `UITableFilter` API is used to implement a table filter with a custom UI. It provides methods to create a UI that automatically adapts to the underlying data format (for example, by displaying a combo box when applicable).

See also

*`UITableFilter`*API

Properties of data model elements *[p 53]*

*`UILabelRendererForHierarchy`*API

89.6 **Record label**

EBX uses a label to display a reference to a given record (for example a foreign key). Labels are also used in the title of a record form and in hierarchical views. This label can be customized in the model using expressions. It is also possible to customize labels using the `UILabelRenderer` API.

See also*`UILabelRenderer`*API

89.7 **CSS and JavaScript**

It is possible to integrate CSS and JavaScript files in each EBX page by declaring them in the registration module. The inclusion of JavaScript files can be subject to conditions through development depending on the context.

See also

Module registration *[p 460]*

Development recommendations *[p 591]*

*`UIDependencyRegisterer`*API

CHAPTER 90
Overview

A user service is an extension to TIBCO EBX that provides a graphical user interface (GUI) allowing users to access specific or advanced functionalities.

An API is available allowing the development of powerful custom user services using the same visual components and data validation mechanisms as standard EBX user interfaces.

This chapter contains the following topics:

90.1 **Nature**

User services exist in different types called *natures*. The nature defines the minimal elements (dataspace, dataset, table, record...) that need to be selected to execute the service. The following table lists the available natures.

Nature	**Description**
`Dataspace`	The nature of a user service that can be launched from the actions menu of a dataspace (branch or snapshot) or from any context where the current selection implies selecting a dataspace.
`Dataset`	The nature of a user service that can be launched from the actions menu of a dataset or from any context where the current selection implies selecting a dataset.
`TableView`	The nature of a user service that can be launched from the toolbar of a table, regardless of the selected view, or from any context where the current selection implies selecting a table.
`Record`	The nature of a user service that can be launched from the toolbar of a record form or from any context where the current selection implies selecting a *single* record.
`Hierarchy`	The nature of a user service that can be launched from the toolbar of a table when a hierarchy view is selected.
`HierarchyNode`	The nature of a user service that can be launched from the menu of a table hierarchy view node. Currently, only *record* hierarchy nodes are supported.
`Association`	The nature of a user service that can be launched from the target table view of an association or for any context where the current selection implies selecting the target table view of an association.
`AssociationRecord`	The nature of a user service that can be launched from the form of a target record of an association node or from any context where the current selection implies selecting a *single* association target record.

90.2 **Declaration**

A user service can be declared at two levels:

- Module,
- Data model.

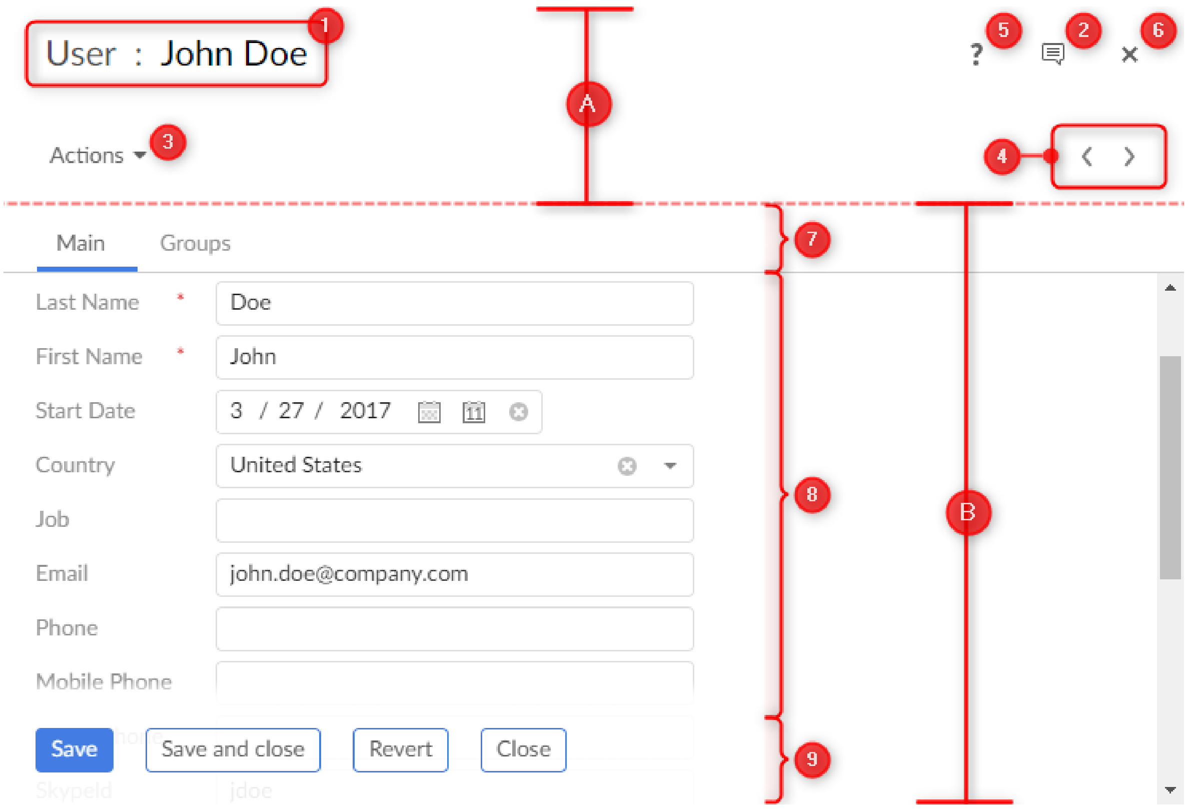
A service declared by a data model can only be launched when the current selection includes a dataset of this model. The user service cannot be of the `Dataspace` nature.

A service declared by a module may be launched for any dataspace or dataset.

The declaration can add restrictions on selections that are valid for the user service.

90.3 Display

On the following figure are displayed the functional areas of a user service.



A. Header B. Form	1. Breadcrumb 2. Message box button 3. Top toolbar 4. Navigation buttons 5. Help button 6. Close button (pop-ups only) 7. Tabs 8. Form pane (one per tab) 9. Bottom buttons

Most areas are optional and customizable. Refer to Quick start [p 567], Implementing a user service [p 571] and Declaring a user service [p 585] for more details.

90.4 Legacy user services

Before the 5.8.0 version, user services were declared in XML and based on Servlet/JSP. Although this type of declaration should no longer be used, the *legacy documentation* is still available.

CHAPTER 91 Quick start

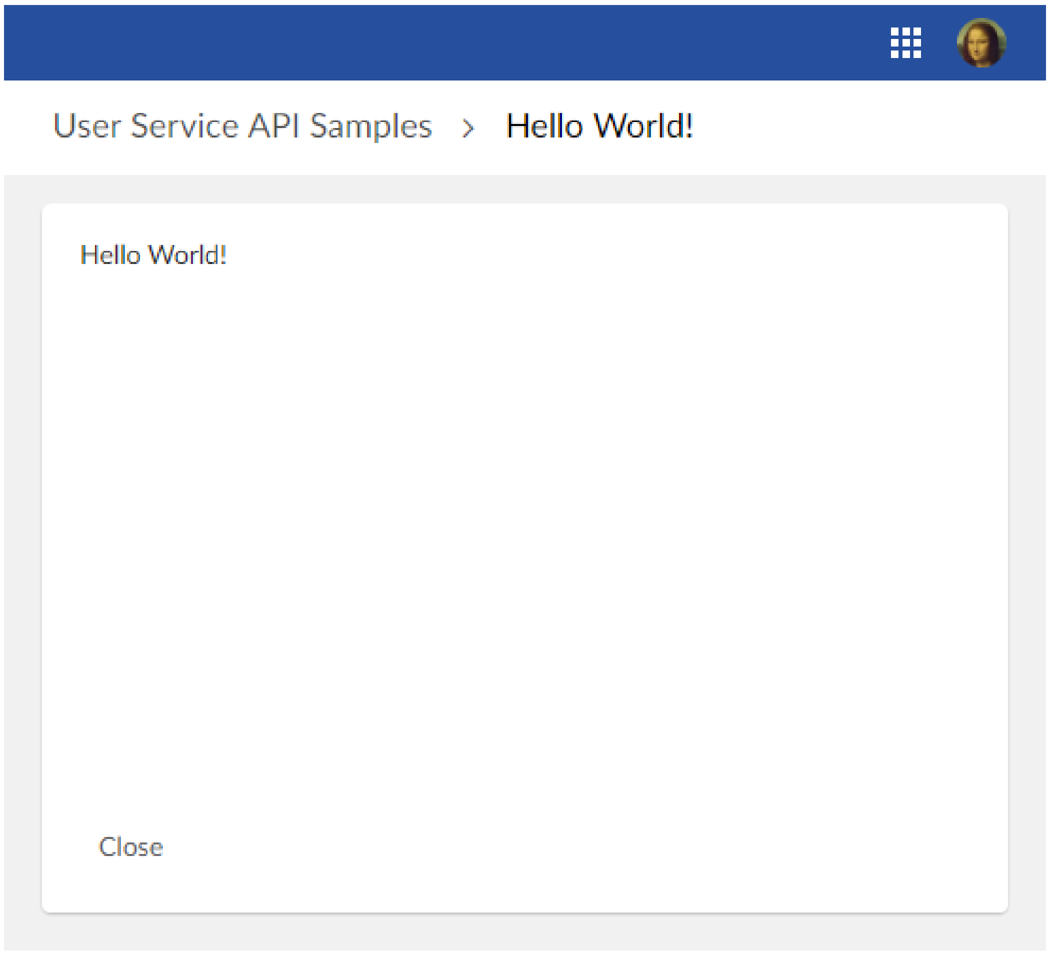
This chapter contains the following topics:

91.1 Main classes

The minimum requirement is to implement two classes, one for the service declaration and one for the implementation itself.

91.2 Hello world

The sample is a dataset user service that simply displays a "hello" message, it can be launched from the action menu of a dataset:



Hello World!

The service implementation class must implement the interface `UserService<DatasetEntitySelection>`:

```
/**
 * This service displays hello world!
 */
public class HelloWordService implements UserService<DatasetEntitySelection>
{
 public HelloWordService()
 {
 }

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DatasetEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  // Set bottom bar
  UIButtonSpecNavigation closeButton = aConfigurator.newCloseButton();
  closeButton.setDefaultButton(true);
  aConfigurator.setLeftButtons(closeButton);

  // Set content callback
  aConfigurator.setContent(this::writeHelloWorld);
 }

 private void writeHelloWorld(
  UserServicePaneContext aPaneContext,
  UserServicePaneWriter aWriter)
 {
  // Display Hello World!

  aWriter.add("<div ");
  aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
  aWriter.add(">");
  aWriter.add("Hello World!");
  aWriter.add("</div>");
 }

 @Override
 public void setupObjectContext(
  UserServiceSetupObjectContext<DatasetEntitySelection> aContext,
  UserServiceObjectContextBuilder aBuilder)
 {
  // No context yet.
 }

 @Override
 public void validate(UserServiceValidateContext<DatasetEntitySelection> aContext)
 {
  // No custom validation is necessary.
 }

 @Override
 public UserServiceEventOutcome processEventOutcome(
  UserServiceProcessEventOutcomeContext<DatasetEntitySelection> aContext,
  UserServiceEventOutcome anEventOutcome)
 {
  // By default do not modify the outcome.
  return anEventOutcome;
 }
}
```

The declaration class must implement the interface `UserServiceDeclaration.OnDataset`:

```
/**
 * Declaration for service hello world!
 */
public class HelloWorldServiceDeclaration implements UserServiceDeclaration.OnDataset
{
 // The service key identifies the user service.
 private static final ServiceKey serviceKey = ServiceKey.forName("HelloWorld");

 public HelloWorldServiceDeclaration()
 {
 }

 @Override
 public ServiceKey getServiceKey()
 {
  return serviceKey;
 }

 @Override
 public UserService<DatasetEntitySelection> createUserService()
 {
  // Creates an instance of the user service.
  return new HelloWordService();
 }

 @Override
 public void defineActivation(ActivationContextOnDataset aContext)
 {
  // The service is activated for all datasets instanciated with
  // the associated data model (see next example).
 }

 @Override
 public void defineProperties(UserServicePropertiesDefinitionContext aContext)
 {
  // This label is displayed in menus that can execute the user service.
  aContext.setLabel("Hello World Service");
 }

 @Override
 public void declareWebComponent(WebComponentDeclarationContext aContext)
 {
 }
}
```

In this sample, the user service is registered by a data model. The data model needs to define a schema extension that implements the following code:

```
public class CustomSchemaExtensions implements SchemaExtensions
{
 @Override
 public void defineExtensions(SchemaExtensionsContext aContext)
 {
  // Register the service.
  aContext.registerUserService(new HelloWorldServiceDeclaration());
 }
}
```

For details on the declaration of schema extensions, see `SchemaExtensions`API.

CHAPTER 92

Implementing a user service

This chapter contains the following topics:

92.1 Implementation interface

The following table lists, per nature, the interface to implement:

Nature	Interface
`Dataspace`	`UserService<DataspaceEntitySelection>`
`Dataset`	`UserService<DatasetEntitySelection>`
`TableView`	`UserService<TableViewEntitySelection>`
`Record`	`UserService<RecordEntitySelection>`
`Hierarchy`	`UserService<HierarchyEntitySelection>`
`HierarchyNode`	`UserService<HierarchyNodeEntitySelection>`
`Association`	`UserService<AssociationEntitySelection>`
`AssociationRecord`	`UserService<AssociationRecordEntitySelection>`

92.2 **Life cycle and threading model**

The user service implementation class is:

- Instantiated at the first HTTP request by a call to its declaration **createUserService()** `UserServiceDeclaration.createUserService`API method.
- Discarded when the current page goes out of scope or when the session times out.

Access to this class is synchronized by TIBCO EBX to make sure that only one HTTP request is processed at a time. Therefore, the class does not need to be thread-safe.

The user service may have attributes. The state of these attributes will be preserved between HTTP requests. However, developers must be aware that these attributes should have moderate use of resources, such as memory, not to overload the EBX server.

92.3 **Object Context**

The object context is a container for objects managed by the user service. This context is initialized and modified by the user service's implementation of the method `UserService.setupObjectContext`API.

An object of the object context is identified by an object key:

```
ObjectKey customerKey = ObjectKey.forName("customer");
```

An object can be:

- A record,
- A dataset,
- A new record not yet persisted,
- A dynamic object.

The object context is maintained between HTTP requests and usually only needs to be set up upon the first request.

Once persisted, a *new record* object is automatically changed to a *plain record* object.

As with **adaptations** `Adaptation`API, **path** `Path`API expressions are used to reference a sub-element of an object.

In the following sample, a pane writer adds a form input mapped to the attribute of an object:

```
// Add an input field for customer's last name.
aWriter.setCurrentObject(customerKey);
aWriter.addFormRow(Path.parse("lastName"));
```

In the following sample, an event callback gets the value of the attribute of an object:

```
// Get value of customer's last name.
ValueContext customerValueContext = aValueContext.getValueContext(customerKey);
String lastName = customerValueContext.getValue(Path.parse("lastName"));
```

A *dynamic object* is an object whose schema is defined by the user service itself. An API is provided to define the schema programmatically. This API allows defining only instance elements (instance nodes). Defining tables is not supported. It supports most other features available with standard EBX data models, such as types, labels, custom widgets, enumerations and constraints, including programmatic ones.

The following sample defines two objects having the same schema:

```
public class SampleService implements UserService<TableViewEntitySelection>
{
 // Define an object key per object:
 private final static ObjectKey _PersonObjectKey = ObjectKey.forName("person");
 private final static ObjectKey _PartnerObjectKey = ObjectKey.forName("partner");

 // Define a path for each property:
 private final static Path _FirstName = Path.parse("firstName");
 private final static Path _LastName = Path.parse("lastName");
 private final static Path _BirthDate = Path.parse("birthDate");

 ...

 // Define and register objects:
 @Override
 public void setupObjectContext(
  UserServiceSetupObjectContext<DataspaceEntitySelection> aContext,
  UserServiceObjectContextBuilder aBuilder)
 {
  if (aContext.isInitialDisplay())
  {
   BeanDefinition def = aBuilder.createBeanDefinition();

   BeanElement firstName = def.createElement(_FirstName, SchemaTypeName.XS_STRING);
   firstName.setLabel("First name");
   firstName.setDescription("This is the given name");
   firstName.setMinOccurs(1);

   BeanElement lastName = def.createElement(_LastName, SchemaTypeName.XS_STRING);
   lastName.setLabel("Last name");
   lastName.setDescription("This is the familly name");
   lastName.setMinOccurs(1);

   BeanElement birthDate = def.createElement(_BirthDate, SchemaTypeName.XS_DATE);
   birthDate.setLabel("Birth date");
   birthDate.addFacetMax(new Date(), false);

   aBuilder.registerBean(_PersonObjectKey, def);
   aBuilder.registerBean(_PartnerObjectKey, def);
  }

  ...
 }
```

92.4 Display setup

The display is set up by the user service's implementation of the method `UserService.setupDisplay`API.

This method is called at each request and can set the following:

- The title (the default is the label specified by the user service declaration),
- The contextual help URL,
- The breadcrumbs,
- The toolbar,
- The bottom buttons.

If necessary, the header and the bottom buttons can be hidden.

The display setup is not persisted and, at each HTTP request, is reset to default before calling the method `UserService.setupDisplay`API.

Bottom buttons

Buttons may be of two types: *action* and *submit*.

An *action* button triggers an *action* event without submitting the form. By default, the user needs to acknowledge that, by leaving the page, the last changes will be lost. This behavior can be customized.

A *submit* button triggers a *submit* event that always submits the form.

More information on events can be found in the following sections.

Content callback

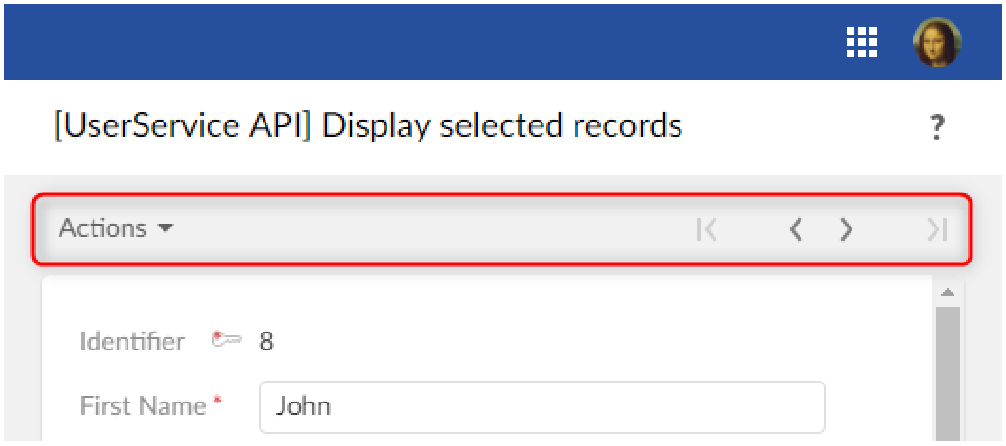
This callback usually implements the interface `UserServicePane`API to render a plain EBX form. The callback can also be an instance of `UserServiceRootTabbedPane`API to render an EBX form with tabs.

For specific cases, the callback can implement `UserServiceRawPane`API. This interface has restrictions but is useful when one wants to implement an HTML form that is not managed by EBX.

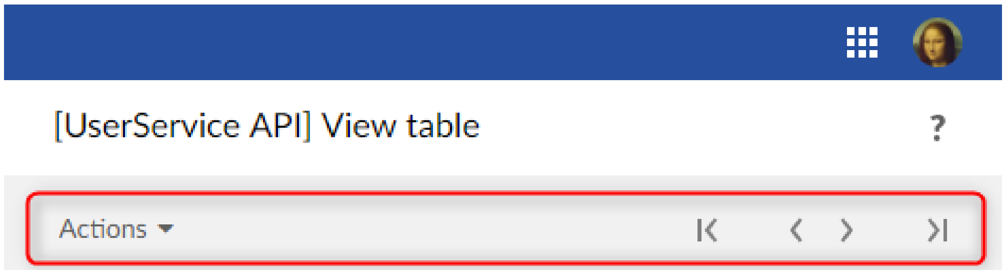
Toolbars

Toolbars are optional and come in two flavors.

The *form* style:



The *table view* style:



The style is automatically selected: toolbars defined for a *record* are of the form style and toolbars defined for a *table* are of the table view style.

Samples

The following sample implements a button that closes the current user service and redirects the user back to the current selection, only if saving the data was successful:

```
public class SampleService implements UserService<...>
{
 private final static ObjectKey _RecordObjectKey = ObjectKey.forName("record");

 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<RecordEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  ...
  // Define a "save and close" button with callback onSave().
  aConfigurator.setLeftButtons(aConfigurator.newSaveCloseButton(this::onSave));
 }

 private UserServiceEventOutcome onSave(UserServiceEventContext anEventContext)
 {
  ProcedureResult result = anEventContext.save(_RecordObjectKey);
  if (result.hasFailed())
  {
   // Save has failed. Redisplay the user message.
   return null;
  }

  // Save has succeded.Close the service.
  return UserServiceNext.nextClose();
 }
}
```

The following sample is compatible with the Java 6 syntax. Only differences with the previous code are shown:

```
public class SampleService implements UserService<...>
{
 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<RecordEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  ...
  // Define a "save and close" button with callback onSave().
  aConfigurator.setLeftButtons(aConfigurator.newSaveCloseButton(new UserServiceEvent() {
   @Override
   public UserServiceEventOutcome processEvent(UserServiceEventContext anEventContext)
   {
    return onSave(anEventContext);
   }
  }));
 }
}
```

The following sample implements a URL that closes the service and redirects the current user to another user service:

```
public class SampleService implements UserService<...>
{
 ...
 private void writePane(UserServicePaneContext aPaneContext, UserServicePaneWriter aWriter)
 {
  // Displays an ULR that redirect current user.
  String url = aWriter.getURLForAction(this::goElsewhere);
  aWriter.add("<a ");
  aWriter.addSafeAttribute("href", url);
  aWriter.add(">Go elsewhere</a");
 }

 private UserServiceEventOutcome goElsewhere(UserServiceEventContext anEventContext)
 {
  // Redirects current user to another user service.
  ServiceKey serviceKey = ServiceKey.forModuleServiceName("CustomerModule", "CustomService");
   return UserServiceNext.nextService(serviceKey);
 }
}
```

The following code is an implementation of the method `UserService.processEventOutcome`API, sufficient for simple user services:

```
public class HelloWordService implements UserService<...>
{
 @Override
 public UserServiceEventOutcome processEventOutcome(
  UserServiceProcessEventOutcomeContext<DatasetEntitySelection> aContext,
  UserServiceEventOutcome anEventOutcome)
 {
  // By default do not modify the outcome.
  return anEventOutcome;
 }
}
```

The following sample is a more complex "wizard" service that includes three steps, each having its own `UserService.setupDisplay`API method:

```
// Custom outcome values.
```

```
public enum CustomOutcome implements UserServiceEventOutcome {
 displayStep1, displayStep2, displayStep3
};

// All steps of the  wizard service implement this interface.
public interface WizardStep
{
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator);
}

// The user service implementation.
public class WizardService implements UserService<...>
{
 // Attribute for current step.
 private WizardStep step = new WizardStep1();

 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  ...

  // Display current step.
  this.step.setupDisplay(aContext, aConfigurator);
 }

 @Override
 public UserServiceEventOutcome processEventOutcome(
  UserServiceProcessEventOutcomeContext<DataspaceEntitySelection> aContext,
  UserServiceEventOutcome anEventOutcome)
 {
  // Custom outcome value processing.

  if (anEventOutcome instanceof CustomOutcome)
  {
   CustomOutcome action = (CustomOutcome) anEventOutcome;
   switch (action)
   {
   case displayStep1:
    this.step = new WizardStep1();
    break;

   case displayStep2:
    this.step = new WizardStep2();
    break;

   case displayStep3:
    this.step = new WizardStep3();
    break;
   }

   // Redisplay the user service.
   return null;
  }

  // Let EBX® process the event outcome.
  return anEventOutcome;
 }
}
```

92.5 **Database updates**

An event callback may update the database.

The following sample saves two objects using a single transaction:

```
public class MultipleObjectsSampleService implements UserService<...>
{
 // This service defines a two objects having same schema.
 private final static ObjectKey _Person1_ObjectKey = ObjectKey.forName("person1");
 private final static ObjectKey _Person2_ObjectKey = ObjectKey.forName("person2");

 ...

 // Save button callback.
 private UserServiceEventOutcome onSave(UserServiceEventContext aContext)
 {
  ProcedureResult result = aContext.save(_Person1_ObjectKey, _Person2_ObjectKey);
  if (result.hasFailed())
  {
   //Save failed. Redisplay the service.
   //The user interface will automatically report error messages.
   return null;
  }

  // Save succeeded. Close the service.
  return UserServiceNext.nextClose();
 }
}
```

The following sample updates the database using a **procedure** `Procedure`API:

```
import com.orchestranetworks.service.*;
import com.orchestranetworks.userservice.*;

public class MultipleObjectsSampleService implements UserService<...>
{
 ...

 // Event callback.
 private UserServiceEventOutcome onUpdateSomething(UserServiceEventContext aContext)
 {
  Procedure procedure = new Procedure()
  {
   public void execute(ProcedureContext aContext) throws Exception
   {
    // Code that updates database should be here.
    ...
   }
  };

  UserServiceTransaction transaction = aContext.createTransaction();
  transaction.add(procedure);

  ProcedureResult result = transaction.execute();
  if (result.hasFailed())
  {
   aContext.addError("Procedure failed");
  }
  else
  {
   aContext.addInfo("Procedure succeeded");
  }

  return null;
}
```

92.6 **Ajax**

A user service can implement Ajax callbacks. An Ajax callback must implement the interface `UserServiceAjaxRequest`API.

The client calls an Ajax callback using the URL generated by: `UserServiceResourceLocator.getURLForAjaxRequest`API.

To facilitate the use of Ajax components, EBX provides the JavaScript prototype `EBX_AJAXResponseHandler` for sending the request and handling the response. For more information on `EBX_AJAXResponseHandler` see `UserServiceAjaxRequest`API.

The following sample implements an Ajax callback that returns partial HTML:

```
public class AjaxSampleService implements UserService<DataspaceEntitySelection>
{
 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  aConfigurator.setLeftButtons(aConfigurator.newCloseButton());
  aConfigurator.setContent(this::writePane);
 }

 /**
  * Displays an URL that will execute the callback
  * and display the returned partial HTML inside a <div> tag.
  */
 private void writePane(UserServicePaneContext aPaneContext, UserServicePaneWriter aWriter)
 {
  // Generate the URL of the Ajax callback.
  String url = aWriter.getURLForAjaxRequest(this::ajaxCallback);

  // The id of the <div> that will display the partial HTML returned by the Ajax callback.
  String divId = "sampleId";

  aWriter.add("<div ");
  aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
  aWriter.add(">");

  // Display the URL that will execute the callback.
  aWriter.add("<a ");
  aWriter.addSafeAttribute("href", "javascript:sample_sendAjaxRequest('" + url + "', '"
   + divId + "')");
  aWriter.add(">");
  aWriter.add("Click to call a user service Ajax callback");
  aWriter.add("</a>");

  // Output the <div> tag that will display the partial HTML returned by the callback.
  aWriter.add("<div ");
  aWriter.addSafeAttribute("id", divId);
  aWriter.add("></div>");

  aWriter.add("</div>");

  // JavaScript method that will send the Java request.
  aWriter.addJS_cr();
  aWriter.addJS_cr("function sample_sendAjaxRequest(url, targetDivId) {");
  aWriter.addJS_cr("  var ajaxHandler = new EBX_AJAXResponseHandler();");

  aWriter.addJS_cr("  ajaxHandler.handleAjaxResponseSuccess = function(responseContent) {");
  aWriter.addJS_cr("    var element = document.getElementById(targetDivId);");
  aWriter.addJS_cr("    element.innerHTML = responseContent;");
  aWriter.addJS_cr("  };");

  aWriter.addJS_cr("  ajaxHandler.handleAjaxResponseFailed = function(responseContent) {");
  aWriter.addJS_cr("    var element = document.getElementById(targetDivId);");
  aWriter.addJS_cr("    element.innerHTML = \"<span class='" + UICSSClasses.TEXT.ERROR
   + "'>Ajax call failed</span>\";");
  aWriter.addJS_cr("  }");

  aWriter.addJS_cr("  ajaxHandler.sendRequest(url);");
  aWriter.addJS_cr("}");
 }

 /**
  * The Ajax callback that returns partial HTML.
  */
 private void ajaxCallback(
  UserServiceAjaxContext anAjaxContext,
  UserServiceAjaxResponse anAjaxResponse)
 {
  UserServiceWriter writer = anAjaxResponse.getWriter();
  writer.add("<p style=\"color:green\">Ajax callback succeeded!</p>");
  writer.add("<p>Current data and time is: ");

  DateFormat format = DateFormat.getDateTimeInstance(
   DateFormat.FULL,
   DateFormat.FULL,
   Locale.US);
  writer.addSafeInnerHTML(format.format(new Date()));

  writer.add("</p>");
 }
}
```

92.7 REST data services

A user service can access REST data services through HTTP requests.

The client should use the URL generated by: `UIResourceLocator.getURLForRest`API. This URL includes required information for the user authentication.

For more information on REST data services see the Built-in RESTful services [p 657].

The following sample implements a REST data service call whose response is printed in a `textarea`:

```
public class RestCallSampleService implements UserService<DataspaceEntitySelection>
{
 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  aConfigurator.setLeftButtons(aConfigurator.newCloseButton());
  aConfigurator.setContent(this::writePane);
 }

 private void writePane(UserServicePaneContext aPaneContext, UserServicePaneWriter aWriter)
 {
  // Generates the URL for REST data service call without additional parameters
  final String url = aWriter.getURLForRest("/ebx-dataservices/rest/{specificPath}", null);

  final String resultAreaId = "restResult";

  // Displays a link for REST data service call
  aWriter.add("<div ");
  aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
  aWriter.add(">");
  aWriter.add("<p>This link will display the response after making a REST call</p>");
  aWriter.add("<a ");
  aWriter.addSafeAttribute("href",
   "javascript:sendRestRequest('" + url + "', '" + resultAreaId + "')");
  aWriter.add(">");
  aWriter.add("Make the call.");
  aWriter.add("</a>");
  aWriter.add("<textarea ");
  aWriter.addSafeAttribute("id", resultAreaId);
  aWriter.add(" readonly=\"readonly\" style=\"width: 100%;\" ></textarea>");
  aWriter.add("</div>");

  // JavaScript method that will send the HTTP REST request
  aWriter.addJS_cr("function sendRestRequest(url, targetId) {");
  aWriter.addJS_cr("  var xhttp = new XMLHttpRequest();");
  aWriter.addJS_cr("  xhttp.open('GET', url, true);");
  aWriter.addJS_cr("  xhttp.setRequestHeader('Content-type', 'application/json');");
  aWriter.addJS_cr("  xhttp.send();");
  aWriter.addJS_cr("  var element = document.getElementById(targetId);");
  aWriter.addJS_cr("  xhttp.onreadystatechange = function() {");
  aWriter.addJS_cr("   if (xhttp.readyState == 4)");
  aWriter.addJS_cr("      element.innerHTML = xhttp.responseText;");
  aWriter.addJS_cr("  }");
  aWriter.addJS_cr("}");
 }
}
```

92.8 File upload

A user service can display forms with file input fields.

The following sample displays a form with two input fields, a title and a file:

```
public class FileUploadService implements UserService<...>
{
 // This service defines a single object named "file".
 private final static ObjectKey _File_ObjectKey = ObjectKey.forName("file");

 // Paths for the "file" object.
 public final static Path _Title = Path.parse("title");
 public final static Path _File = Path.parse("file");

 ...

 @Override
 public void setupObjectContext(
  UserServiceSetupObjectContext<DataspaceEntitySelection> aContext,
  UserServiceObjectContextBuilder aBuilder)
 {
  if (aContext.isInitialDisplay())
  {
   // Create a definition for the "model" object.
```

```
    BeanDefinition def = aBuilder.createBeanDefinition();
    aBuilder.registerBean(_File_ObjectKey, def);

    BeanElement element;

    element = def.createElement(_Title, SchemaTypeName.XS_STRING);
    element.setLabel("Title");
    element.setMinOccurs(1);

    // Type for a file must be BeanDefinition.OSD_FILE_UPLOAD.
    element = def.createElement(_File, BeanDefinition.OSD_FILE_UPLOAD);
    element.setLabel("File");
    element.setMinOccurs(1);
   }
  }

  @Override
  public void setupDisplay(
   UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
   UserServiceDisplayConfigurator aConfigurator)
  {
   aConfigurator.setTitle("File upload service");
   aConfigurator.setLeftButtons(aConfigurator.newSubmitButton("Upload", this::onUpload),
  aConfigurator.newCancelButton());

   // IMPORTANT: Following method must be called to enable file upload.
   // This will set form  encryption type to  "multipart/form-data".
   aConfigurator.setFileUploadEnabled(true);

   aConfigurator.setContent(this::writePane);
  }


  private void writePane(UserServicePaneContext aContext, UserServicePaneWriter aWriter)
  {
   final UIWidgetFileUploadFactory fileUploadFactory = new UIWidgetFileUploadFactory();

   aWriter.setCurrentObject(_File_ObjectKey);

   aWriter.startTableFormRow();

   // Title input.
   aWriter.addFormRow(_Title);

   // File upload input.
   UIWidgetFileUpload widget = aWriter.newCustomWidget(_File, fileUploadFactory);
   // Default filter for file names.
   widget.setAccept(".txt");
   aWriter.addFormRow(widget);

   aWriter.endTableFormRow();
  }

  private UserServiceEventOutcome onUpload(UserServiceEventContext anEventContext)
  {
   ValueContextForInputValidation valueContext = anEventContext.getValueContext(_File_ObjectKey);

   String title = (String) valueContext.getValue(_Title);
   UploadedFile file = (UploadedFile) valueContext.getValue(_File);

   InputStream in;
   try
   {
    in = file.getInputStream();
   }
   catch (IOException e)
   {
    // Should not happen.
    anEventContext.addError("Cannot read file.");
    return null;
   }

   // Do something with title and the input stream.
   return UserServiceNext.nextClose();
  }
 }
```

For more information, see `UIWidgetFileUpload`API.

92.9 **File download**

A user service can display URLs or buttons to download files. The actual downloading of a file is under the control of the user service.

The following sample displays a URL to download a file:

```
public class FileDownloadService implements UserService<DataspaceEntitySelection>
{
 ...

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  aConfigurator.setLeftButtons(aConfigurator.newCloseButton());
  aConfigurator.setContent(this::writePane);
 }

 private void writePane(UserServicePaneContext aContext, UserServicePaneWriter aWriter)
 {
  aWriter.add("<div ");
  aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
  aWriter.add(">");

  // Generate and display the URL for the download.
  String downloadURL = aWriter.getURLForGetRequest(this::processDownloadRequest);

  aWriter.add("<a ");
  aWriter.addSafeAttribute("href", downloadURL);
  aWriter.add(">Click here to download a sample file</a>");

  aWriter.add("</div>");
 }

 private void processDownloadRequest(
  UserServiceGetContext aContext,
  UserServiceGetResponse aResponse)
 {
  // The file is plain text.
  aResponse.setContentType("text/plain;charset=UTF-8");
  // Remove the following statement to display the file directly in the browser.
  aResponse.setHeader("Content-Disposition", "attachment; filename=\"sample.txt\"");

  // Write a text file using UTF-8 encoding.
  PrintWriter out;
  try
  {
   out = new PrintWriter(new OutputStreamWriter(aResponse.getOutputStream(), "UTF-8"));
  }
  catch (IOException ex)
  {
   throw new RuntimeException(ex);
  }

  DateFormat format = DateFormat.getDateTimeInstance(
   DateFormat.FULL,
   DateFormat.MEDIUM,
   Locale.US);
  Date now = new Date();

  out.println("Hello !");
  out.println("This is a sample text file downloaded on " + format.format(now)
   + ", from EBX®.");

  out.close();
 }
}
```

92.10 **User service without display**

A user service may be designed to execute a task without display and return to the previous screen or redirect the user to another screen.

This type of service must implement the interface **UserServiceExtended** `UserServiceExtended`API and method `UserServiceExtended.initialize`API.

The following sample deletes selected records in the current table view:

```
public class DeleteRecordsService implements UserServiceExtended<TableViewEntitySelection>
{
 ...

 @Override
 public UserServiceEventOutcome initialize(
  UserServiceInitializeContext<TableViewEntitySelection> aContext)
 {
  final List<AdaptationName> records = new ArrayList<>();

  // Deletes all selected rows in a single transaction.
  RequestResult requestResult = aContext.getEntitySelection().getSelectedRecords().execute();
  try
  {
   for (Adaptation record = requestResult.nextAdaptation(); record != null; record =
 requestResult.nextAdaptation())
   {
    records.add(record.getAdaptationName());
   }
  }
  finally
  {
   requestResult.close();
  }

  Procedure deleteProcedure = new Procedure()
  {
   @Override
   public void execute(ProcedureContext aContext) throws Exception
   {
    for (AdaptationName record : records)
    {
     aContext.doDelete(record, false);
    }
   }
  };

  UserServiceTransaction transaction = aContext.createTransaction();
  transaction.add(deleteProcedure);

  // Adds an information messages for current user.
  ProcedureResult procedureResult = transaction.execute(true);
  if (!procedureResult.hasFailed())
  {
   if (records.size() <= 1)
   {
    aContext.addInfo(records.size() + " record was deleted.");
   }
   else
   {
    aContext.addInfo(records.size() + " records were deleted.");
   }
  }

  // Do not display the user service and return to current view.
  return UserServiceNext.nextClose();
 }

 @Override
 public void setupObjectContext(
  UserServiceSetupObjectContext<TableViewEntitySelection> aContext,
  UserServiceObjectContextBuilder aBuilder)
 {
  //Do nothing.
 }

 @Override
 public void setupDisplay(
  UserServiceSetupDisplayContext<TableViewEntitySelection> aContext,
  UserServiceDisplayConfigurator aConfigurator)
 {
  //Do nothing.
 }

 @Override
 public void validate(UserServiceValidateContext<TableViewEntitySelection> aContext)
 {
  //Do nothing.
 }

 @Override
 public UserServiceEventOutcome processEventOutcome(
  UserServiceProcessEventOutcomeContext<TableViewEntitySelection> aContext,
  UserServiceEventOutcome anEventOutcome)
 {
  return anEventOutcome;
 }
}
```

Known limitation

If such service is called in the context of a Web component, an association, a perspective action or a hierarchy node, The service will be launched, initialized and closed, but the service's target entity will still be displayed.

CHAPTER 93

Declaring a user service

This chapter contains the following topics:

93.1 Declaration interface

The following table lists, per nature, the interface that the declaration class of a user service must implement:

Nature	Declaration **Interface**
`Dataspace`	`UserServiceDeclaration.OnDataspace`
`Dataset`	`UserServiceDeclaration.OnDataset`
`TableView`	`UserServiceDeclaration.OnTableView`
`Record`	`UserServiceDeclaration.OnRecord`
`Hierarchy`	`UserServiceDeclaration.OnHierarchy`
`HierarchyNode`	`UserServiceDeclaration.OnHierarchyNode`
`Association`	`UserServiceDeclaration.OnAssociation`
`AssociationRecord`	`UserServiceDeclaration.OnAssociationRecord`

93.2 Life cycle and threading model

The user service declaration class is instantiated at the TIBCO EBX startup and must be coded to be thread-safe. This is usually not an issue as most implementations should be immutable classes.

93.3 Registration

A user service declaration must be registered by a module or a data model.

Registration by a module is achieved by the module registration servlet by a code similar to:

```
public class CustomRegistrationServlet extends ModuleRegistrationServlet
{
 @Override
 public void handleServiceRegistration(ModuleServiceRegistrationContext aContext)
 {
  // Register custom user service declaration.
  aContext.registerUserService(new CustomServiceDeclaration());
 }
}
```

For more information on the module registration servlet, see module registration [p 460] and `ModuleRegistrationServlet`API.

Registration by a data model is achieved by a code similar to:

```
public class CustomSchemaExtensions implements SchemaExtensions
{
 @Override
 public void defineExtensions(SchemaExtensionsContext aContext)
 {
  // Register custom user service declaration.
  aContext.registerUserService(new CustomServiceDeclaration());
 }
}
```

For more information on data model extensions, see `SchemaExtensions`API.

93.4 Service properties

The properties of a user service include its *label*, *description*, *confirmation message* and the *group* that owns the service. All are optional but it is a good practice to at least define the label.

For more information, see `UserServiceDeclaration.defineProperties`API.

93.5 Service activation scope

The activation scope defines on which selection the service is available.

Example of a service activation definition:

```
public class CustomServiceDeclaration implements UserServiceDeclaration.OnTableView
{
 ...

 @Override
 public void defineActivation(ActivationContextOnTableView aContext)
 {
  // activates the service in all dataspaces except the "Reference" branch.
  aContext.includeAllDataspaces(DataspaceType.BRANCH);
  aContext.excludeDataspacesMatching(Repository.REFERENCE, DataspaceChildrenPolicy.NONE);

  // activates the service only on tables "table01" and "table03".
  aContext.includeSchemaNodesMatching(
   CustomDataModelPath._Root_Table01.getPathInSchema(),
   CustomDataModelPath._Root_Table03.getPathInSchema());

  // service will be enabled only when at least one record is selected.
  aContext.forbidEmptyRecordSelection();

  // service will not be displayed in hierarchical views (neither in the
  // top toolbar, nor in the hierarchy nodes' menu).
  aContext.setDisplayForLocations(
   ActionDisplaySpec.HIDDEN,
   ToolbarLocation.HIERARCHICAL_VIEW_TOP,
   ToolbarLocation.HIERARCHICAL_VIEW_NODE);

  // service will be considered as disabled if not explicitly enabled
  // via the UI.
  aContext.setDefaultPermission(UserServicePermission.getDisabled());
 }
}
```

For more information about declaring the activation scope, see `UserServiceDeclaration.defineActivation`API.

For more information about the resolution of the user service availability, see Resolving permissions on services [p 287].

93.6 Web component declaration

Parameters declaration and availability in workflows and perspectives

User services are automatically available as web components with a set of built-in parameters depending on the service's nature. To define custom parameters and/or set the service web component as available when configuring a workflow user task, a perspective menu action or a toolbar web component action, `UserServiceDeclaration.declareWebComponent`API must be used.

Example of a web component declaration:

```
public class CustomServiceDeclaration implements UserServiceDeclaration.OnDataset
{
 ...

 @Override
 public void declareWebComponent(WebComponentDeclarationContext aContext)
 {
  // makes this web component available when configuring a workflow user task.
  aContext.setAvailableAsWorkflowUserTask(true);

  // adds a custom input parameter.
  aContext.addInputParameter(
   "source",
   UserMessage.createInfo("Source"),
   UserMessage.createInfo("Source of the imported data."));

  // modifies the built-in "instance" parameter to be "input/output" instead of "input".
  aContext.getBuiltInParameterForOverride("instance").setOutput(true);
 }
}
```

See Using TIBCO EBX as a Web Component [p 193] for more information.

User service extension

It is possible to extend existing user services (built-in or custom) in order to add input/output parameters when using these services as web components.

In order to do so, a user service extension must first be registered by a module or a data model.

Registration by a module is achieved by the module registration servlet by code similar to:

```
public class CustomRegistrationServlet extends ModuleRegistrationServlet
{
 ...

 @Override
 public void handleServiceRegistration(ModuleServiceRegistrationContext aContext)
 {
  // Register user service extension declaration.
  aContext.registerUserServiceExtension(new ServiceExtensionDeclaration());
 }
}
```

For more information on the module registration servlet, see module registration [p 460] and `ModuleRegistrationServlet`API.

Registration by a data model is achieved by a code similar to:

```
public class CustomSchemaExtensions implements SchemaExtensions
{
 ...

 @Override
 public void defineExtensions(SchemaExtensionsContext aContext)
 {
  // Register user service extension declaration.
  aContext.registerUserServiceExtension(new ServiceExtensionDeclaration());
 }
}
```

For more information on the data model extension, see `SchemaExtensions`API.

93.7 User service groups

User service groups are used to organize the display of user services in menus and permission management screens.

The following types of service groups are available:

- Built-in User Service Groups [p 588] provided by EBX,
- Custom User Service Groups [p 589] declared in a module.

The link between groups and services is made upon service declaration. See Associating a service to a group [p 589].

Built-in User Service Groups

Available built-in service groups:

Service Group Key	Description
`@ebx-importExport`	Group containing all built-in import and export services provided by EBX. In the default menus, these services are displayed in an "Import / Export" sub-menu.
`@ebx-views`	Group containing services to display in the 'Views' menu. Unlike other service groups, services associated with this group are not displayed in the default menus, but only in the 'Views' menu displayed in the non-customizable part of the table top toolbar. These services can still be added manually to a custom toolbar.

Declaring a User Service Group

User Service Groups must be declared while registering the module, using the method `ModuleServiceRegistrationContext.registerServiceGroup`API:

```
public class CustomRegistrationServlet extends ModuleRegistrationServlet
{
 ...

 @Override
 public void handleServiceRegistration(ModuleServiceRegistrationContext aContext)
 {
  // In CustomModuleConstants,
  // CUSTOM_SERVICE_GROUP_KEY = ServiceGroupKey.forServiceGroupInModule("customModule", "customGroup")

  // registers CUSTOM_SERVICE_GROUP_KEY service group
  aContext.registerServiceGroup(
   CustomModuleConstants.CUSTOM_SERVICE_GROUP_KEY,
   UserMessage.createInfo("Custom group"),
   UserMessage.createInfo("This group contains services related to..."));
 }
}
```

Associating a service to a group

The association of a service with a group is made at its **declaration** `UserServiceDeclaration`API, using the method `UserServicePropertiesDefinitionContext.setGroup`API:

```
public class CustomServiceDeclaration implements UserServiceDeclaration.OnDataset
{
 ...

 @Override
 public void defineProperties(UserServicePropertiesDefinitionContext aContext)
 {
  // associates the current service to the CUSTOM_SERVICE_GROUP_KEY group
  aContext.setGroup(CustomModuleConstants.CUSTOM_SERVICE_GROUP_KEY);
 }
}
```

A service can be associated with either a built-in or a custom service group. In the latter case, this service will be displayed in this built-in group, just like other built-in services belonging to this group.

CHAPTER 94

Development recommendations

This chapter contains the following topics:

94.1 HTML

It is recommended to minimize the inclusion of specific HTML styles and tags to allow the default styles of TIBCO EBX to apply to custom interfaces. The approach of the API is to automatically apply a standardized style to all elements on HTML pages, while simplifying the implementation process for the developer.

XHTML

EBX is a Rich Internet Application developed in XHTML 1.0 Transitional. It means that the structure of the HTML is strict XML file and that all tags must be closed, including "br" tags. This structure allows for greater control over CSS rules, with fewer differences in browser rendering.

iFrames

Using iFrame is allowed in EBX, especially in collaboration with a URL of a `UIHttpManagerComponent`API. For technical reasons, it is advised to set the `src` attribute of an iFrame using JavaScript only. In this way, the iFrame will be loaded once the page is fully rendered and when all the built-in HTML components are ready.

Example

The following example, developed from any `UIComponentWriter`API, uses a `UIHttpManagerComponent`API to build the URL of an iFrame, and set it in the right way:

```
// using iFrame in the current page requires a sub session component
UIHttpManagerComponent managerComponent = writer.createWebComponentForSubSession();

// [...] managerComponent configuration

String iFrameURL = managerComponent.getURIWithParameters();

String iFrameId = "mySweetIFrame";

// place the iFrame in the page, with an empty src attribute
writer.add("<iframe id=\"").add(iFrameId).add("\" src=\"\" >").add("</iframe>");

// launch the iFrame from JavaScript
```

```
writer.addJS("document.getElementById(\"").addJS(iFrameId).addJS("\").src = \"").addJS(iFrameURL).addJS("\";");
```

94.2 **CSS**

Public CSS classes

The constant catalog `UICSSClasses`API offers the main CSS classes used in the software to style the components. These CSS classes ensure a proper long-term integration into the software, because they follow the background colors, borders, customizable text in the administration; the floating margins and paddings fluctuate according to the variable density; to the style of the icons, etc.

See also *`UICSSUtils`*API

Advanced CSS

EBX allows to integrate to all its pages one or more external Cascading Style Sheet. These external CSS, considered as resources, need to be declared in the Module registration [p 460].

In order to ensure the proper functioning of your CSS rules and properties without altering the software, the following recommendations should be respected. Failure to respect these rules could lead to:

- Improper functioning of the software, both aesthetically and functionally: risk of losing the display of some of the data and some input components may stop working.
- Improper functioning of your CSS rules and properties, since the native CSS rules will impact the CSS implementation.

Reserved prefixes for CSS identifiers and class names

The following prefixes should not be used to create CSS #ids and .classes.

ebx_	Internal built-in
yui	Yahoo User Interface global
ygtv	Yahoo User Interface tree view
layout-doc	Yahoo User Interface layout
cke_	CK editor (used by HTML editor widget)
fa	Font Awesome (icons used by perspectives and toolbars)

CSS classes used internally by EBX

The following CSS classes should never be included in a ruleset that has no contextual selector.

If you do not prefix your CSS selector using one of the CSS classes below, it will cause conflicts and corrupt the UI of EBX.

selected	YUI selected tree node
hd	YUI floating pane header
bd	YUI floating pane body
ft	YUI floating pane footer
container-close	YUI inner popup close button
underlay	YUI inner popup shadow
hastitle	YUI menu group with title
topscrollbar	YUI menu top scroll zone
bottomscrollbar	YUI menu bottom scroll zone
withtitle	YUI calendar
link-close	YUI calendar close button
collapse	YUI layout closed pane indicator
pull-right	Font Awesome parameter
pull-left	Font Awesome parameter

Examples to avoid conflicts

Don't

```
.selected {
 background-color: red;
}
```

Do

```
#myCustomComponent li.selected {
 background-color: red;
}
```

94.3 **JavaScript**

Public JS functions

The catalog of JavaScript functions `JavaScriptCatalog`API offers a list of functions to use directly (through copy-paste) in the JS files.

JavaScript call during page generation in Java

When generating the HTML of a Java component, it is possible to add specific JavaScript code with the API `UIJavaScriptWriter`API.

This JavaScript is executed once the whole page is loaded. It is possible to instantly manage the HTML elements written with `UIBodyWriter.add`API. Setting on-load functions (such as `window.onload = myFunctionToCallOnload;`) is not supported because the execution context comes after the on-load event.

Advanced JavaScript

EBX allows to include one or more external JavaScript files. These external JavaScript files, considered as resources, need to be declared in the Module registration [p 460]. For performance reasons, it is recommended to include the JavaScript resource only when necessary (in a User service or a specific form, for example). The API `UIDependencyRegisterer`API allows a developer to specify the conditions for which the JavaScript resources will be integrated into a given page according to its context.

In order to ensure the proper functioning of your JavaScript resources without altering the software, the following recommendations should be respected. Failure to respect them could lead to:

- Improper functioning of the software: if functions or global variables of the software were to be erased, some input or display components (including the whole screen) may stop working.
- Improper functioning of your JavaScript instructions, since global variables or function names could be erased.

Reserved JS prefixes

The following prefixes are reserved and should not be used to create variables, functions, methods, classes, etc.

ebx_	Internal built-in API
EBX_	Internal built-in API
YAHOO	Yahoo User Interface API

SOAP data services

CHAPTER 95

Introduction

This chapter contains the following topics:

95.1 Overview

Data services allow external systems to interact with the data governed in the TIBCO EBX repository using the SOAP/Web Services Description Language (WSDL) standards.

In order to invoke SOAP operations [p 615], for an integration use case, a WSDL [p 607] must be generated from a data model. It will be possible to perform operations such as:

- Selecting, inserting, updating, deleting, or counting records
- Selecting dataset values
- Getting the differences on a table between dataspaces or snapshots, or between two datasets based on the same data model
- Getting the credentials of records

Other generic WSDLs can be generated and allow performing operations such as:

- Creating, merging, or closing a dataspace
- Creating or closing a snapshot
- Validating a dataset, dataspace, or a snapshot
- Starting, resuming or ending a data workflow

- Administrative operations to manage access to the UI or to system information

> **Note**
>
> See SOAP and REST comparative [p 604].

95.2 Activation and configuration

Data services are enabled by deploying the `ebx-dataservices` web application along with the other EBX modules. See Java EE deployment overview [p 317] for more information.

In case of specific deployment, for example using reverse-proxy mode, see URLs computing [p 356] for more information.

The default method for accessing data services is over HTTP, although it is also possible to use JMS for the SOAP operations. See JMS configuration [p 355] and Using JMS [p 598] for more information.

95.3 Interactions

Input and output message encoding

All input messages must be *exclusively* in UTF-8. All output messages are in UTF-8.

Tracking information

Depending on the data services operation being called, it may be possible to specify session tracking information.

- Example for a SOAP operation, the request header contains:

```
<SOAP-ENV:Header>
 <!-- optional security header here -->
 <m:session xmlns:m="urn:ebx-schemas:dataservices_1.0">
  <trackingInformation>String</trackingInformation>
 </m:session>
</SOAP-ENV:Header>
```

For more information, see `Session.getTrackingInfo`API in the Java API.

Session parameters

Depending on the data services operation being called, it is possible to specify session input parameters. They are defined in the request body.

Input parameters are available on custom Java components with a session object, such as: triggers, access rules, `custom` web services. They are also available on data workflow operations.

- Example for a SOAP operation, the optional request header contains:

```
<SOAP-ENV:Header>
 <!-- optional security header here -->
 <m:session xmlns:m="urn:ebx-schemas:dataservices_1.0">
  <!-- optional trackingInformation header here -->
  <inputParameters>
   <parameter>
    <name>String</name>
    <value>String</value>
   </parameter>
   <!-- for some other parameters, copy complex
        element 'parameter' -->
  </inputParameters>
 </m:session>
</SOAP-ENV:Header>
```

For more information, see `Session.getInputParameterValue`API in the Java API.

Exception handling

In case of unexpected server error upon execution of:

- A SOAP operation, a SOAP exception response is returned to the caller via the `soap:Fault` element. For example:

```
<soapenv:Fault>
 <faultcode>soapenv:java.lang.IllegalArgumentException</faultcode>
 <faultstring />
 <faultactor>admin</faultactor>
 <detail>
  <m:StandardException xmlns:m="urn:ebx-schemas:dataservices_1.0">
   <code>java.lang.IllegalArgumentException</code>
   <label/>
   <description>java.lang.IllegalArgumentException:
    Parent home not found at
    com.orchestranetworks.XX.YY.ZZ.AA.BB(AA.java:44) at
    com.orchestranetworks.XX.YY.ZZ.CC.DD(CC.java:40) ...
   </description>
  </m:StandardException>
 </detail>
</soapenv:Fault>
```

Using JMS

It is possible to access SOAP operations using JMS instead of HTTP. The JMS architecture relies on one JMS request queue (mandatory), on one JMS failure queue (optional), and on JMS response queues, see configuration JMS [p 355]. The mandatory queue is the input queue. Request messages must be put in the input queue, and response messages are put by EBX in the `replyTo` queue of the JMS request. The optional queue is the failure queue which allows you to replay an input message if necessary. If the queue is set and activated in the configuration file and an exception occurs while handling a request message, this input message will be copied in the failure queue.

The relationship between a request and a response is made by copying the `messageId` message identifier field of the JMS request into the `correlId` correlation identifier field of the response.

JMS location points must be defined in the Lineage administration in order to specialize the generated WSDL. If no specific location point is given, the default value will be `jms:queue:jms/EBX_QueueIn`.

95.4 Security

Authentication

Authentication is mandatory to access to data. Several authentication methods are available and described below. The descriptions are ordered by priority (EBX applies the highest priority authentication method first).

- 'Basic Authentication Scheme' method is based on the HTTP-Header `Authorization` in base 64 encoding, as described in RFC 2617 (Basic Authentication Scheme).

```
If the user agent wishes to send the userid "Alibaba" and password "open sesame",
it will use the following header field:
> Authorization: Basic QWxpYmFiYTpvcGVuIHNlc2FtZQ==
```

- 'Standard Authentication Scheme' is based on the HTTP `Request`. User and password are extracted from request parameters. For more information on request parameters, see Parameters [p 610] section.

For more information on this authentication scheme, see `Directory.authenticateUserFromLoginPassword`API.

- The 'SOAP Security Header Authentication Scheme' method is based on the Web Services Security UsernameToken Profile 1.0 specification.

  By default, the type `PasswordText` is supported. This is done with the following SOAP-Header defined in the WSDL:

```
<SOAP-ENV:Header>
 <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
  <wsse:UsernameToken>
   <wsse:Username>String</wsse:Username>
   <wsse:Password Type="wsse:PasswordText">String</wsse:Password>
  </wsse:UsernameToken>
 </wsse:Security>
</SOAP-ENV:Header>
```

  Note

  Only available for SOAP operations [p 615].

- 'Specific authentication Scheme' is based on the HTTP `Request`. An implementation of this method can, for example, extract a password-digest or a ticket from the HTTP request. See `Directory.authenticateUserFromHttpRequest`API for more information.
- The 'SOAP Specific Header Authentication Scheme'.

  For more information, see Overriding the SOAP security header [p 599].

Global permissions

Global access permissions can be independently defined for the SOAP and WSDL connector accesses. For more information see Global permissions [p 383].

Overriding the SOAP security header

It is possible to override the default WSS header in order to define another security authentication mechanism. Such an override is taken into account for both HTTP and JMS. To define and override, use the 'SOAP Header Security declaration' configuration settings under Administration > Lineage, which includes the following fields:

Schema location	The URI of the Security XML Schema to import into the WSDL.
Target namespace	The target namespace of elements in the schema.
Namespace prefix	The prefix for the target namespace.
Message name	The message name to use in the WSDL.
Root element name	The root element name of the security header. The name must be the same as the one declared in the schema.
wsdl:part element name	The name of the `wsdl:part` of the message.

The purpose of overriding the default security header is to change the declaration of the WSDL message matching the security header so that it contains the following:

```
<wsdl:definitions ... xmlns:MyPrefix="MyTargetNameSpace" ...
  ...
  <xs:schema ...>
    <xs:import namespace="MyTargetNameSpace" schemaLocation="MySchemaURI"/>
    ...
  </xs:schema>
  ...
  <wsdl:message name="MySecurityMessage">
    <wsdl:part name="MyPartElementName" element="MyPrefix:MySecurityRootElement"/>
  </wsdl:message>
  ...
  <wsdl:operation name="...">
    <soap:operation soapAction="..." style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
      <soap:header message="impl:MySecurityMessage" part="MyPartElementName" use="literal"/>
  ...
  </wsdl:operation>
</wsdl:definitions>
```

The corresponding XML Schema header declaration would be as follows:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="MyNameSpace"
  xmlns:MyPrefix="MyNameSpace">
  <element name="MySecurityRootElement" type="MyPrefix:SpecificSecurity"/>
  <complexType name="SpecificSecurity">
    <sequence>
      <element name="AuthToken" type="string"/>
    </sequence>
  </complexType>
</schema>
```

A SOAP message using the XML schema and configuration above would have the following header:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Header>
    <m:MySecurityRootElement xmlns:m="MyNameSpace">
      <AuthToken>String</AuthToken>
    </m:MySecurityRootElement>
    ...
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    ...
  </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

To handle this non-default header, you must implement the method: `Directory.authenticateUserFromSOAPHeader`API.

> **Note**
>
> Only available for SOAP operations [p 615].

Lookup mechanism

Because EBX offers several authentication methods, a lookup mechanism based on conditions was set to know which method should be applied for a given request. The method application conditions are evaluated according to the authentication scheme priority. If the conditions are not satisfied, the server evaluates the next method. The following table presents the available authentication methods for each supported protocol and their application conditions. They are ordered from the highest priority to the lowest.

Operation / Protocol	**Authentication methods and application conditions**
SOAP / JMS	SOAP Security Header [p 599] • The SOAP request is received over the JMS protocol. • The SOAP header content must contains a `Security` element. SOAP Specific Header [p 599] • The SOAP request is received over the JMS protocol. • The SOAP header content must not contain a `Security` element.
SOAP / HTTP	Basic [p 598] • The HTTP request must hold an `Authorization` header. • `Authorization` header value must start with the word `Basic`. • No `login` is provided in the URL parameters. Standard [p 598] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters. SOAP Security Header [p 599] • The SOAP header content must contain a `Security` element. • The HTTP request must not hold an `Authorization` header. • No `login` is provided in the URL parameters. Specific [p 599] • The HTTP request must not satisfy the conditions of the previous authentication methods. SOAP Specific Header [p 599] • The SOAP header content must not contain a `Security` element. • The HTTP request must not hold an `Authorization` header. • No `login` is provided in the URL parameters.
WSDL / HTTP	Basic [p 598] • The HTTP request must not hold an `Authorization` header. • `Authorization` header value must start with the word `Basic`. • No `login` is provided in the URL parameters. Standard [p 598] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters. Specific [p 599] • The HTTP request must not satisfy the conditions of the previous authentication methods.

In case of multiple authentication methods present in the same request, EBX will return an HTTP code `401 Unauthorized`.

95.5 Monitoring

Data service events can be monitored through the log category `ebx.dataServices`, as declared in the EBX main configuration file. For example, `ebx.log4j.category.log.dataServices= INFO, ebxFile:dataservices`.

See also

Configuring the EBX logs [p 351]

TIBCO EBX main configuration file [p 345]

95.6 **SOAP and REST comparative**

Operations	**SOAP**	**REST**
Data		
Select or count records (with filter and/or view publication)	X	X
Selector for possible enumeration values (with filter)		X
Insert, update or delete records	X	X
Select or count history records (with filter and/or view publication)		X
Select node values from dataset	X	X
Update node value from dataset		X
Get table or dataset changes between dataspaces or snapshots	X	
Refresh a replication unit	X	
Get credentials for records	X	
Dataspaces		
Create, close, merge a dataspace	X	
Create, close a snapshot	X	
Validate a dataspace or a snapshot	X	
Validate a dataset	X	
Locking a dataspace	X	
Workflow		
Start, resume or end a workflow	X	
Administration		
Manage the default directory content 'Users', 'Roles'... tables.	X	X
Open, close the user interface	X	X

Operations	**SOAP**	**REST**
Select, insert, update, delete operations for administration dataset		X
Select the system information	X	X
Other		
Develop web services from the Java API		X (*)

(*) See REST Toolkit [p 719] for more information.

95.7 Limitations

Date, time & dateTime format

Data services only support the following date and time formats:

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

SOAP naming convention

Due to the naming convention of the data service operations, each table defined within a data model must have a unique name for the WSDL generation.

CHAPTER 96

WSDL generation

This chapter contains the following topics:

96.1 Supported standard

TIBCO EBX generates a WSDL that complies with the W3C Web Services Description Language 1.1 standard.

96.2 **Operation types**

A WSDL can be generated for different types of operations:

Operation type	**WSDL description**
`custom`	WSDL for EBX add-ons.
`dataset`	WSDL for dataset and replication operations.
`directory`	WSDL for default EBX directory operations. It is also possible to filter data using the `tablePaths` [p 611] or `operations` [p 611] parameters.
`repository`	WSDL for dataspace or snapshot management operations.
`tables`	WSDL for operations on the tables of a specific dataset.
`userInterface`	Deprecated since version 5.8.1. This operation type has been replaced by `administration`. While the user interface management operations are still available for backward compatibility reasons, it is recommended to no longer use this type. WSDL for user interface management operations (these operations can only be accessed by administrators).
`administration`	WSDL for administration operations like: • user interface management • system information retrieval These operations can only be accessed by administrators.
`workflow`	WSDL for EBX workflow management operations.

96.3 **WSDL download methods**

EBX supports the following methods:

- from the user interface
- from HTTP protocol

A WSDL can only be downloaded by authorized profiles:

Operation type	Access right permissions
`custom`	All profiles, if at least one web service is registered.
`dataset`	All profiles.
`directory`	All profiles, if the following conditions are valid: • No specific directory implementation is used. (The built-in Administrator role is only subject to this condition). • Global access permissions are defined for the administration. • 'Directory' dataset permissions have writing access for the current profile.
`repository`	All profiles.
`tables`	All profiles.
`userInterface`	Deprecated since version 5.8.1. This operation type has been replaced by `administration`. While the user interface management operations are still available for backward compatibility reasons, it is recommended to no longer use this type. Built-in administrator role or delegated administrator profiles, if all conditions are valid: • Global access permissions are defined for the administration. • 'User interface' dataset permissions have writing access for the current profile.
`administration`	Built-in administrator role or delegated administrator profiles, if all conditions are valid: • Global access permissions are defined for the administration. • 'Administration' dataset permissions have write access for the current profile.
`workflow`	All profiles.

WSDL download from the user interface

An authorized user can download an EBX WSDL from the data services administration area.

> **Note**
>
> See Generating a WSDL for dataspace operations [p 184] in the user guide for more information.

WSDL download from HTTP protocol

An application can download an EBX WSDL using `GET` or `POST` HTTP method. The application has to be authenticated using a profile with appropriate rights.

URL format

```
http[s]://<host>[:<port>]/<ebx-dataservices>/{type}[/{dataspace}[/{dataset}]]?
{queryParameters}
```

Where:

- `<ebx-dataservices>` corresponds to the 'ebx-dataservices.war' web application's path. The path is composed by multiple, or none, URI segments followed by the web application's name.
- `{type}` corresponds to the operation type [p 608].
- `{dataspace}` corresponds to the dataspace or snapshot identifier.
- `{dataset}` corresponds to the dataset name.
- `{queryParameters}` corresponds to common or dedicated operation parameters passed through the URL.

Parameters

A request parameter can be specified by one of the following methods:

- a `PathParam` which corresponds to a path segment from the URL (recommended)

- a `QueryParam` which corresponds to a standard HTTP parameter with value.

Parameter name	`PathParam`	`QueryParam`	Required	Description
`WSDL`	no	yes	yes	Specifies the WSDL download operation. Empty value.
`login`	no	yes	no	Specifies the user identifier. Required when the standard authentication method is used. `String` type value.
`password`	no	yes	no	Specifies the user password. Required when the standard authentication method is used. `String` type value.
`type`	yes	no	yes	Specifies the operation type [p 608]. Possible values are: `custom`, `dataset`, `directory`, `administration`, `userInterface`, `repository`, `tables` or `workflow`. `String` type value.
`branch` `version`	yes	yes	(*)	Specifies the dataspace or the snapshot identifier. (*) required for `tables` and `dataset` types, ignored otherwise. `String` type value.
`instance`	yes	yes	(*)	Specifies the dataset name. `String` type value.
`tablePaths`	no	yes	no	Specifies the list of table paths. Optional for `tables` or `directory` types, ignored otherwise. If not defined, all tables are selected. Each table path is separated by a comma character. `String` type value.
`operations`	no	yes	no	Allows generating a WSDL for an operations subset. Optional for `tables` or `directory` operation types, ignored otherwise. If not defined, all operations for the given type are generated. This parameter's value is a concatenation of one or more of the following characters: • C = Count record(s) • D = Delete record(s) • E = Get credentials • G = Get changes • I = Insert record(s) • U = Update record(s)

Parameter name	PathParam	QueryParam	Required	Description
				• R = Read operations (equivalent to CEGS) • S = Select record(s) • W = Write operations (equivalent to DIU) `String` type value.
`namespaceURI`	yes	yes	(**)	Specifies the unique name space URI of the custom web service. (**)Is required when `type` parameter is set to `custom`, ignored otherwise. `URI` type value.
`attachmentFilename`	no	yes	(***)	Specifies the attachment file name. (***) optional if `isContentInAttachment` parameter is defined and set to `true`, ignored otherwise. `String` type value.
`isContentInAttachment`	no	yes	no	Specifies if the WSDL is downloaded as an attachment. `Boolean` type value. Default value is `false`.
`targetNamespace`	no	yes	no	Overrides the target namespace URI of the WSDL. `URI` type value. Default value is `urn:ebx:ebx-dataservices`.

Message body

No message body is required.

HTTP codes

An HTTP code is always returned. Errors are indicated by a code above 400.

Status code	Information
`200` *(OK)*	The WSDL content was successfully generated and is returned by the request (optionally in an attachment [p 612]).
`400` *(Bad request)*	The request is incorrect. This occurs when: • A request element is incorrect. • The unicity check on table names contains at least one error. **Note** See WSDL and table operations [p 546] for more information.
`401` *(Unauthorized)*	Request requires an authenticated user.
`403` *(Forbidden)*	Request is not allowed for the authenticated user.
`405` *(Method not allowed)*	Request is not allowed in this configuration.
`500` *(Internal error)*	Request generates an error (a stack trace and a detailed error message are returned).

Response body

The response body depends on the returned status code and on the requested WSDL content.

- `200` *(OK)*: the HTTP header `Content-Type` is set to `text/xml;charset=UTF-8`.

  If the content is in attachment, the HTTP header `Content-Disposition` is set to `attachment; filename*=UTF-8''<filename.wsdl>`.
- `4xx`: A detailed message is returned in the body. The HTTP header `Content-Type` is set to `text/html;charset=utf-8`.

96.4 HTTP examples

Some of the following examples are displayed in two methods: `PathParam` and `QueryParam`.

- The WSDL will contain all repository operations, using standard authentication method.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/repository?
  WSDL&login=<login>&password=<password>
  ```
- The WSDL will contain all workflow operations.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/workflow?WSDL
  ```
- The WSDL will contain all tables operations for the `'dataset1'` dataset in `'dataspace1'` dataspace.

  `PathParam`

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables/dataspace1/dataset1?WSDL
```

`QueryParam`

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables?
WSDL&branch=<dataspace1>&instance=<dataset1>
```

- The WSDL will contain all tables with only read operations for the `'dataset1'` dataset in `'dataspace1'` dataspace.

  `PathParam`

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/tables/dataspace1/dataset1?
  WSDL&operations=R
  ```

  `QueryParam`

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/tables?
  WSDL&branch=dataspace1&instance=dataset1&operations=R
  ```

- The WSDL will contain the two selected tables operations for the `'dataset1'` dataset in `'dataspace1'` dataspace.

  `PathParam`

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/tables/dataspace1/dataset1?
  WSDL&tablePaths=/root/table1,/root/table2
  ```

  `QueryParam`

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/tables?
  WSDL&branch=dataspace1&instance=dataset1&tablePaths=/root/table1,/root/table2
  ```

- The WSDL will contain custom web service operations for the dedicated URI.

  `PathParam`

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/custom/urn:ebx-
  test:com.orchestranetworks.dataservices.WSDemo?WSDL
  ```

  `QueryParam`

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/custom?WSDL&namespaceURI=urn:ebx-
  test:com.orchestranetworks.dataservices.WSDemo
  ```

CHAPTER 97

SOAP operations

This chapter contains the following topics:

97.1 Operations generated from a data model

For a data model used in an TIBCO EBX repository, it is possible to dynamically generate a corresponding WSDL, that defines its operations. When using this WSDL, it will be possible to read and/or write in the EBX repository. For example, for a table located at the path `/root/XX/exampleTable`, the generated requests would follow the structure of its underlying data model and include the name of the table `<m:{operation}_exampleTable xmlns:m="urn:ebx-schemas:dataservices_1.0">`.

Attention

Since the WSDL and the SOAP operations tightly depend on the data model structure, it is important to redistribute the up-to-date WSDL after any data model change.

Content policy

Access to the content of records, the presence or absence of XML elements, depend on the resolved permissions [p 275] of the authenticated user session. Additional aspects, detailed below, can impact the content.

Disabling fields from data model

The `hiddenInDataServices` property, defined in the data model, allows always hiding fields in data services, regardless of the user profile. This parameter has an impact on the generated WSDL: any hidden field or group will be absent from the request and response structure.

Modifying the `hiddenInDataServices` parameter value has the following impact on a client which would still use the former WSDL:

- On request, if the data model property has been changed to `true`, and if the concerned field is present in the WSDL request, an exception will be thrown.

- On response, if the schema property has been changed to `false`, WSDL validation will return an error if it is activated.

This setting of "Default view" is defined inside data model.

See also

Hiding a field in Data Services *[p 540]*

Permissions *[p 275]*

Association field

Read-access on table records can export the association fields as displayed in UI Manager. This feature can be coupled with the 'hiddenInDataServices' model parameter.

> **Note**
>
> Limitations: change and update operations do not manage association fields. Also, the select operation only exports the first level of association elements (the content of associated objects cannot contain association elements).

Common request parameters

Several parameters are common to several operations and are detailed below.

Element	Description	Required
branch	The identifier of the dataspace to which the dataset belongs.	Either this parameter or the 'version' parameter must be defined. Required for the 'insert', 'update' and 'delete' operations.
version	The identifier of the snapshot to which the dataset belongs.	Either this parameter or the 'branch' parameter must be defined
instance	The unique name of the dataset which contains the table to query.	Yes
predicate	XPath predicate [p 227] defines the records on which the request is applied. If empty, all records will be retrieved, except for the 'delete' operation where this field is mandatory.	Only required for the 'delete' operation
data	Contains the records to be modified, represented within the structure of their data model. The whole operation is equivalent to an XML import. The details of the operations performed on data content are specified in the section Import [p 215].	Only required for the insert and update operations
viewPublication	This parameter can be combined with the predicate [p 617] parameter as a logical `AND` operation. The behavior of this parameter is described in the section EBX as a Web Component [p 196]. It cannot be used if the 'viewId' parameter is used, and cannot be used on hierarchical views.	No
viewId	*Deprecated since version 5.2.3*. This parameter has been replaced by the parameter 'viewPublication'. While it remains available for backward compatibility, it will eventually be removed in a future version. This parameter cannot be used if the 'viewPublication' parameter is used.	No
blockingConstraintsDisabled	This property is available for all table updates data service operations. If `true`, the validation process disables blocking constraints defined in the data model. If this parameter is not present, the default is `false`. See Blocking and non-blocking constraints [p 522] for more information.	No
details	The `details` element specifies the following option: The optional attribute `locale` (default 'en-US') defines the language of the blockingConstraintsDisabled [p 617] parameter in which the validation messages must be returned.	No

Element	Description	Required
disableRedirectionToLastBroadcast	This property is available for all data service operations. If `true`, access to a delivery dataspace on a D3 primary node is not redirected to the last broadcast snapshot. Otherwise, access to such a dataspace is always redirected to the last snapshot broadcast. If this parameter is not present, the default is `false` (redirection on a D3 master enabled), unless the configuration property ebx.dataservices.disableRedirectionToLastBroadcast.default [p 354] has been set. If the specified dataspace is not a delivery dataspace on a D3 primary node, this parameter is ignored.	No

Select operations

Select request on table

```
<m:select_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <viewPublication>String</viewPublication>
 <exportCredentials>boolean</exportCredentials>
 <pagination>
  <previousPageLastRecordPredicate>String</previousPageLastRecordPredicate>
  <pageSize>Integer</pageSize>
 </pagination>
</m:select_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 617].	
version	See the description under Common parameters [p 617].	
instance	See the description under Common parameters [p 617].	
predicate	See the description under Common parameters [p 617]. This parameter can be combined with the viewPublication [p 617] parameter as a logical `AND` operation.	
viewPublication	See the description under Common parameters [p 617].	

includesTechnicalData	The response will contain technical data if `true`. See also the optimistic locking [p 634] section.
	Each returned record will contain additional attributes for this technical information, for instance:
	`...<table`
	`ebxd:lastTime="2010-06-28T10:10:31.046"`
	` ebxd:lastUser="Uadmin"`
	`ebxd:uuid="9E7D0530-828C-11DF-B733-0012D01B6E76">... .`
		No

exportCredentials	If `true` the select will also return the credentials for each record.	No
pagination	Enables pagination, see child elements below.	No
pageSize (nested under the `pagination` element)	When pagination is enabled, defines the number of records to retrieve.	When pagination is enabled, yes
previousPageLastRecordPredicate (nested under the `pagination` element)	When pagination is enabled, XPath predicate that defines the record after which the page must fetched, this value is provided by the previous response, as the element `lastRecordPredicate`. If the passed record is not found, the first page will be returned.	No
disableRedirectionToLastBroadcast	See the description under Common parameters [p 618].	

Select response on table

```
<ns1:select_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <data>
  <XX>
   <TableName>
    <a>key1</a>
    <b>valueb</b>
    <c>1</c>
    <d>1</d>
   </TableName>
  </XX>
 </data>
 <credentials>
  <XX>
   <TableName predicate="./a='key1'">
    <a>W</a>
    <b>W</b>
    <c>W</c>
    <d>W</d>
   </TableName>
  </XX>
 </credentials>
 <lastRecordPredicate>./a='key1'</lastRecordPredicate>
</ns1:select_{TableName}Response>
```

with:

Element	Description
data	Content of records that are displayed following the table path.
credentials	Contains the access right for each node of each record.
lastRecordPredicate	Only returned if the pagination is enabled, this defines the last records in order to be used on the next request in the element `previousPageLastRecordPredicated`.

See also the optimistic locking [p 634] section.

Select request on dataset

This operation returns dataset content without table.

```
<m:selectInstance xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
</m:selectInstance>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 617].	
version	See the description under Common parameters [p 617].	
instance	See the description under Common parameters [p 617].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 618].	

Select response on dataset

```
<ns1:selectInstanceResponse xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <data>
  <settings>
   <XX>
    <a>key1</a>
    <b>valueb</b>
    <c>1</c>
    <d>true</d>
   </XX>
  </settings>
 </data>
</ns1:selectInstanceResponse>
```

with:

Element	Description
data	Dataset content without table.

Delete operation

Deletes records or, for a child dataset, defines the record state as "occulting" or "inherited" according to the record context. Records are selected by the `predicate` parameter.

Delete request

```
<m:delete_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <includeOcculting>boolean</includeOcculting>
 <inheritIfInOccultingMode>boolean</inheritIfInOccultingMode>
 <checkNotChangedSinceLastTime>dateTime</checkNotChangedSinceLastTime>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
</m:delete_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 617].	
instance	See the description under Common parameters [p 617].	
predicate	See the description under Common parameters [p 617].	
includeOcculting	Includes the records in occulting mode. Default value is `false`.	No
inheritIfInOccultingMode	Inherits the record from its parent if it is in occulting mode. Default value is `false`.	No
occultIfInherit	*Deprecated since version 5.7.0* Occults the record if it is in inherit mode. Default value is `false`.	No
checkNotChangedSinceLastTime	Timestamp used to ensure that the record has not been modified since the last read. Also see the optimistic locking [p 634] section.	No
blockingConstraintsDisabled	See the description under Common parameters [p 617].	
details	See the description under Common parameters [p 617].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 618].	

Delete response

If one of the provided parameters is illegal, if a required parameter is missing, if the action is not authorized or if no record is selected, an exception is returned. Otherwise, the specific response is returned:

```
<ns1:delete_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <blockingConstraintMessage>String</blockingConstraintMessage>
</ns1:delete_{TableName}Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. '95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted.
blockingConstraintMessage	This element is present if the status is equal to '95' with a localized message. The locale of the message is retrieved from the request parameter or from the user session.

Count operation

Count request

```
<m:count_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
</m:count_{TableName}>
```

with:

Element	Description
branch	See the description under Common parameters [p 617].
version	See the description under Common parameters [p 617].
instance	See the description under Common parameters [p 617].
predicate	See the description under Common parameters [p 617].
disableRedirectionToLastBroadcast	See the description under Common parameters [p 618].

Count response

```
<ns1:count_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <count>Integer</count>
</ns1:count_{TableName}Response>
```

with:

Element	Description
count	The number of records that correspond to the predicate in the request.

Update operation

Update request

```
<m:update_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <updateOrInsert>boolean</updateOrInsert>
 <byDelta>boolean</byDelta>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
 <data>
  <XX>
   <TableName>
    <a>String</a>
    <b>String</b>
    <c>String</c>
    <d>String</d>
    ...
   </TableName>
  </XX>
 </data>
</m:update_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 617].	
instance	See the description under Common parameters [p 617].	
updateOrInsert	If `true` and the record does not currently exist, the operation creates the record. `boolean` type, the default value is `false`.	No
byDelta	If `true` and an element does not currently exist in the incoming message, the target value is not changed. If `false` and node is declared `hiddenInDataServices`, the target value is not changed. The complete behavior is described in the sections Insert and update operations [p 216].	No
blockingConstraintsDisabled	See the description under Common parameters [p 617].	
details	See the description under Common parameters [p 617].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 618].	
data	See the description under Common parameters [p 617].	

See also*Optimistic locking* *[p 634]*

Update response

```
<ns1:update_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <blockingConstraintMessage>String</blockingConstraintMessage>
</ns1:update_{TableName}Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. '95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted.
blockingConstraintMessage	This element is present if the status is equal to '95' with a localized message. The locale of the message is retrieved from the request parameter or from the user session.

Insert operation

Insert request

```
<m:insert_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <byDelta>boolean</byDelta>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
 <data>
  <XX>
   <TableName>
    <a>String</a>
    <b>String</b>
    <c>String</c>
    <d>String</d>
    ...
   </TableName>
  </XX>
 </data>
</m:insert_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 617].	
instance	See the description under Common parameters [p 617].	
byDelta	If `true` and an element does not currently exist in the incoming message, the target value is not changed. If `false` and node is declared `hiddenInDataServices`, the target value is not changed. The complete behavior is described in the sections Insert and update operations [p 216].	No
blockingConstraintsDisabled	See the description under Common parameters [p 617].	
details	See the description under Common parameters [p 617].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 618].	
data	See the description under Common parameters [p 617].	

Insert response

```
<ns1:insert_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <blockingConstraintMessage>String</blockingConstraintMessage>
 <inserted>
  <predicate>./a='String'</predicate>
 </inserted>
</ns1:insert_{TableName}Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. '95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted.
blockingConstraintMessage	This element is present if the status is equal to '95' with a localized message. The locale of the message is retrieved from the request parameter or from the user session.
predicate	A predicate matching the primary key of the inserted record. When several records are inserted, the predicates follow the declaration order of the records in the input message.

Get changes operations

Returns changes according to the Content policy [p 615].

Get changes requests

Changes between two datasets:

```
<m:getChangesOnDataSet_{schemaName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <compareWithBranch>String</compareWithBranch>
 <compareWithVersion>String</compareWithVersion>
 <compareWithInstance>String</compareWithInstance>
 <resolvedMode>boolean</resolvedMode>
 <includeInstanceUpdates>boolean</includeInstanceUpdates>
</m:getChangesOnDataSet_{schemaName}>
```

Changes between two tables:

```
<m:getChanges_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <compareWithBranch>String</compareWithBranch>
 <compareWithVersion>String</compareWithVersion>
 <resolvedMode>boolean</resolvedMode>
</m:getChanges_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 617].	
version	See the description under Common parameters [p 617].	
instance	See the description under Common parameters [p 617].	
compareWithBranch	The identifier of the dataspace with which to compare.	One of either this parameter or the 'compareWithVersion [p 627]' parameter must be defined.
compareWithVersion	The identifier of the snapshot with which to compare.	One of either this parameter or the 'compareWithBranch [p 627]' parameter must be defined.
compareWithInstance	The identifier of the dataset with which to compare. If it is undefined, instance [p 627] parameter is used.	No
resolvedMode	Defines whether or not the difference is calculated in resolved mode. Default is `true`. See **Resolved mode** `DifferenceHelper.resolvedMode`API for more information.	No
includeInstanceUpdates	Defines if the content updates of the dataset are included. Default is `false`.	No
pagination	Enables pagination context for the operations `getChanges` and `getChangesOnDataSet`. Allows client to define pagination context size. Each page contains a collection of inserted, updated and/or deleted records of tables according to the maximum size. Get changes persisted context is built at first call according to the page size parameter in request. The pagination context is persisted on the server file system [p 380] and allows invoking the next page until last page or when a timeout is reached. For creation: Defines `pageSize` parameter. For next: Defines `context` element with `identifier` from previous response. Enables pagination, see child elements below.	No
pageSize (nested under `pagination` element)	Defines maximum number of records in each page. Minimal size is `50`.	No (Only for creation)
context (nested under `pagination` element)	Defines content of pagination context.	No (Only for next)

Element	Description	Required
identifier (nested under `context` element)	Pagination context identifier.	Yes
disableRedirectionToLastBroadcast	See the description under Common parameters [p 618].	

Note

If none of the *compareWithBranch* or *compareWithVersion* parameters are specified, the comparison will be made with their parent:

- if the current dataspace or snapshot is a dataspace, the comparison is made with its initial snapshot (includes all changes made in the dataspace);
- if the current dataspace or snapshot is a snapshot, the comparison is made with its parent dataspace (includes all changes made in the parent dataspace since the current snapshot was created);
- returns an exception if the current dataspace is the 'Reference' dataspace.

See also *`DifferenceHelper`* API

Get changes responses

Changes between two datasets:

```
<ns1:getChangesOnDataSet_{schemaName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <updated>
  <changes>
   <path>... Path of changed terminal value ...</path>
   <path>...</path>
  </changes>
  <data>
   ... see the whole content of dataset values (without table) ...
  </data>
 </updated>
 <getChanges_{TableName1}>
  ... see the getChanges between tables response example ...
 </getChanges_{TableName1}>
 <getChanges_{TableName2}>
  ... see the getChanges between tables response example ...
 </getChanges_{TableName2}>
 ...
</ns1:getChangesOnDataSet_{schemaName}Response>
```

Changes between two tables:

```
<ns1:getChanges_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <inserted>
  <XX>
   <TableName>
    <a>AVALUE3</a>
    <b>BVALUE3</b>
    <c>CVALUE3</c>
    <d>DVALUE3</d>
   </TableName>
  </XX>
 </inserted>
 <updated>
  <changes>
   <change predicate="./a='AVALUE2'">
    <path>/b</path>
    <path>/c</path>
   </change>
  </changes>
  <data>
   <XX>
    <TableName>
     <a>AVALUE2</a>
     <b>BVALUE2.1</b>
     <c>CVALUE2.1</c>
     <d>DVALUE2</d>
    </TableName>
   </XX>
  </data>
 </updated>
 <deleted>
  <predicate>./a='AVALUE1'</predicate>
 </deleted>
</ns1:getChanges_{TableName}Response>
```

with:

Element	Description	Required
inserted	Contains inserted record(s) from choice `compareWithBranch` or `compareWithVersion`. Content under this element corresponding to an XML export of inserted records.	No
updated	Contains updated record(s) or dataset content.	No
changes (nested under `updated` element)	Only the group of field have been updated.	Yes
change (nested under `changes` element)	Group of fields have been updated with own `XPath predicate` attribute of the record.	Yes
path (nested under `change` element)	Path in the record.	Yes
path (nested under `changes` element)	Path in the dataset.	Yes
data (nested under `updated` element)	Content under this element corresponding to an XML export of dataset or updated records.	No
deleted	Records have been deleted from context of request. Content corresponding to a list of `predicate` element who contains the XPath predicate of record.	No
pagination	When pagination is enabled on request. Get changes persisted context allows invoking the next page until last page or when the context timeout is reached. Contains a next page: Defines `context` element with `identifier`. Is the last page: Defines `context` element without `identifier`. Enables pagination, see child elements below.	No
context (nested under `pagination` element)	Defines content of pagination context.	Yes (Only for next and last)
identifier (nested under `context` element)	Pagination context identifier. Not defined at last returned page.	No
pageNumber (nested under `context` element)	Current page number in pagination context.	Yes
totalPages (nested under `context` element)	Total pages in pagination context.	Yes

Get changes operation with pagination enabled

Only `pagination` element and sub elements have been described.

For creation:

Extract of request:

```
...
 <pagination>
  <!-- on first request for creation -->
  <pageSize>Integer</pageSize>
 </pagination>
...
```

Extract of response:

```
...
 <pagination>
  <!-- on next request to continue -->
  <context>
   <identifier>String</identifier>
   <pageNumber>Integer</pageNumber>
   <totalPages>Integer</totalPages>
  </context>
 </pagination>
...
```

For next:

Extract of request:

```
...
 <pagination>
  <context>
   <identifier>String</identifier>
  </context>
 </pagination>
...
```

Extract of response:

```
...
 <pagination>
  <!-- on next request to continue -->
  <context>
   <identifier>String</identifier>
   <pageNumber>Integer</pageNumber>
   <totalPages>Integer</totalPages>
  </context>
 </pagination>
...
```

For last:

Extract of request:

```
...
 <pagination>
  <context>
   <identifier>String</identifier>
  </context>
 </pagination>
...
```

Extract of response:

```
...
 <pagination>
  <context>
   <pageNumber>Integer</pageNumber>
   <totalPages>Integer</totalPages>
  </context>
 </pagination>
...
```

Get credentials operation

Get credentials request

```
<m:getCredentials_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <viewPublication>String</viewPublication>
</m:getCredentials_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 617].	
version	See the description under Common parameters [p 617].	
instance	See the description under Common parameters [p 617].	
predicate	See the description under Common parameters [p 617].	
viewPublication	See the description under Common parameters [p 617].	

Get credentials response

```
<ns1:getCredentials_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <XX>
  <TableName>
   <a>R</a>
   <b>W</b>
   <c>H</c>
   <d>W</d>
   ...
  </TableName>
 </XX>
</ns1:getCredentials_{TableName}Response>
```

With the following possible values:

- R: for read-only
- W: for read-write
- H: for hidden

Multiple chained operations

Multiple operations request

It is possible to run multiple operations across tables in the dataset, while ensuring a consistent response. The operations are executed sequentially, according to the order defined on the client side.

All operations are executed in a single transaction with a `SERIALIZABLE` isolation level. If all requests in the multiple operation are read-only, they are allowed to run fully concurrently along with other read-only transactions, even in the same dataspace.

When an error occurs during one operation in the sequence, all updates are rolled back and the client receives a `StandardException` error message with details.

See Concurrency and isolation levels [p 466].

```
<m:multi_ xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
 <request id="id1">
  <{operation}_{TableName}>
  ...
  </{operation}_{TableName}>
 </request>
 <request id="id2">
  <{operation}_{TableName}>
  ...
  </{operation}_{TableName}>
 </request>
</m:multi_>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 617].	
version	See the description under Common parameters [p 617].	
instance	See the description under Common parameters [p 617].	
blockingConstraintsDisabled	See the description under Common parameters [p 617].	
details	See the description under Common parameters [p 617].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 618].	
request	This element contains one operation, like a single operation without branch, version and instance parameters. This element can be repeated multiple times for additional operations. Each request can be identified by an 'id' attribute. In a response, this 'id' attribute is returned for identification purposes. Operations such as `count`, `select`, `getChanges`, `getCredentials`, `insert`, `delete` or `update`.	Yes

Note:

- Does not accept a limit on the number of `request` elements.
- The request `id` attribute must be unique in multi-operation requests.
- If all operations are read only (`count`, `select`, `getChanges`, or `getCredentials`) then the whole transaction is set as read-only for performance considerations.

Limitation:

- The `multi` operation applies to one model and one dataset (parameter `instance`).
- The `select` operation cannot use the pagination parameter.

See also

Procedure^API

Repository^API

Multiple operations response

See each response operation for details.

```
<ns1:multi_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <response id="id1">
  <{operation}_{TableName}Response>
  ...
  </{operation}_{TableName}Response>
 </response>
 <response id="id2">
  <{operation}_{TableName}Response>
  ...
  </{operation}_{TableName}Response>
 </response>
</ns1:multi_Response>
```

with:

Element	Description
response	This element contains the response of one operation. It is be repeated multiple times for additional operations. Each response is identified by an 'id' attribute set in the request or automatically generated. The content of the element corresponds to the response of a single operation, such as `count`, `select`, `getChanges`, `getCredentials`, `insert`, `delete` or `update`.

Optimistic locking

To prevent an update or a delete operation on a record that was read earlier but may have changed in the meantime, an optimistic locking mechanism is provided.

A select request can include technical information by adding the element `includesTechnicalData`:

```
<m:select_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <includesTechnicalData>boolean</includesTechnicalData>
</m:select_{TableName}>
```

The value of the `lastTime` attribute can then be used in the following update request. If the record has been changed since the specified time, the update will be cancelled. The attribute `lastTime` has to be added on the record to prevent the update of a modified record.

```
<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 ebxd:lastTime="2010-06-28T10:10:31.046">
    <a>String</a>
    <b>String</b>
    <c>String</c>
    <d>String</d>
    ...
   </TableName>
  </XX>
 </data>
```

```
</m:update_{TableName}>
```

The value of the `lastTime` attribute can also be used to prevent deletion on a modified record:

```
<m:delete_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <checkNotChangedSinceLastTime>2010-06-28T10:10:31.046</checkNotChangedSinceLastTime>
</m:delete_{TableName}>
```

Note

The element `checkNotChangedSinceLastTime` may be used more than once but only for the same record. This implies that if the `predicate` element returns more than one record, the request will fail.

97.2 Operations on datasets and dataspaces

Parameters for operations on dataspaces and snapshots are as follows:

Element	*Description*	*Required*
branch	Identifier of the target dataspace on which the operation is applied. When not specified, the 'Reference' dataspace is used except for the merge dataspace operation where it is required.	One of either this parameter or the 'version' parameter must be defined. Required for the dataspace merge, locking, unlocking and replication refresh operations.
version	Identifier of the target snapshot on which the operation is applied.	One of either this parameter or the 'branch' parameter must be defined
versionName	Identifier of the snapshot to create. If empty, it will be defined on the server side.	No
childBranchName	Identifier of the dataspace child to create. If empty, it will be defined on the server side.	No
instance	The unique name of the dataset on which the operation is applied.	Required for the replication refresh operation.
ensureActivation	Defines if validation must also check whether this instance is activated.	Yes
details	Defines if validation returns details. The optional attribute `severityThreshold` defines the lowest severity level of message to return. The possible values are, in descending order of severity, 'fatal', 'error', 'warning', or 'info'. For example, setting the value to 'error' will return error and fatal validation messages. If this attribute is not defined, all levels of messages are returned by default. The optional attribute `locale` (default 'en-US') defines the language in which the validation messages are to be returned.	No. If not specified, no details are returned.
owner	Defines the owner. Must respect the inner format as returned by `Profile.format`API.	No
branchToCopyPermissionFrom	Defines the identifier of the dataspace from which to copy the permissions.	No
documentation	Documentation for a dedicated language.	No

Element	*Description*	*Required*
	Multiple `documentation` elements may be used for several languages.	
locale (nested under the `documentation` element)	Locale of the documentation.	Only required when the `documentation` element is used
label (nested under the `documentation` element)	Label for the language.	No
description (nested under the `documentation` element)	Description for the language.	No

Validate a dataspace

Validate dataspace request

```
<m:validate xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
</m:validate>
```

Validate dataspace response

```
<ns1:validate_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <validationReport>
  <instanceName>String</instanceName>
  <fatals>boolean</fatals>
  <errors>boolean</errors>
  <infos>boolean</infos>
  <warnings>boolean</warnings>
 </validationReport>
</ns1:validate_Response>
```

Validate a dataset

Validate dataset request

```
<m:validateInstance xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <ensureActivation>boolean</ensureActivation>
 <details severityThreshold="fatal|error|warning|info" locale="Locale"/>
</m:validateInstance>
```

Validate dataset response

```
<ns1:validateInstance_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <validationReport>
  <instanceName>String</instanceName>
  <fatals>boolean</fatals>
  <errors>boolean</errors>
  <infos>boolean</infos>
  <warnings>boolean</warnings>
  <details>
   <reportItem>
    <severity>{fatal|error|warning|info}</severity>
    <message>
     <internalId />
     <text>String</text>
    </message>
    <subject>
     <table>Path</table>
     <predicate>String</predicate>
     <path>Path</path>
    </subject>
   </reportItem>
  </details>
 </validationReport>
</ns1:validateInstance_Response>
```

Create a dataspace

Create dataspace request

```
<m:createBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <owner>String</owner>
 <branchToCopyPermissionFrom>String</branchToCopyPermissionFrom>
 <documentation>
  <locale>Locale</locale>
  <label>String</label>
  <description>String</description>
 </documentation>
 <childBranchName>String</childBranchName>
</m:createBranch>
```

Create dataspace response

```
<ns1:createBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <childBranchName>String</childBranchName>
</ns1:createBranch_Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully.

Create a snapshot

Create snapshot request

```
<m:createVersion xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <versionName>String</versionName>
 <owner>String</owner>
 <documentation>
  <locale>Locale</locale>
  <label>String</label>
  <description>String</description>
 </documentation>
</m:createVersion>
```

Create snapshot response

```
<ns1:createVersion_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <versionName>String</versionName>
</ns1:createVersion_Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully.

Locking a dataspace

Lock dataspace request

```
<m:lockBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <durationToWaitForLock>Integer</durationToWaitForLock>
 <message>
  <locale>Locale</locale>
  <label>String</label>
 </message>
</m:lockBranch>
```

with:

Element	*Description*	*Required*
durationToWaitForLock	This parameter defines the maximum duration (in seconds) that the operation waits for a lock before aborting.	No, does not wait by default
message	User message of the lock. Multiple `message` elements may be used.	No
locale (nested under the `message` element)	Locale of the user message.	Only required when the `message` element is used
label (nested under the `message` element)	The user message.	No

Lock dataspace response

```
<ns1:lockBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:lockBranch_Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully. '94' indicates that the dataspace has been already locked by another user. Otherwise, a SOAP exception is thrown.

Unlocking a dataspace

Unlock dataspace request

```
<m:unlockBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
</m:unlockBranch>
```

Unlock dataspace response

```
<ns1:unlockBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:unlockBranch_Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully. Otherwise, a SOAP exception is thrown.

Merge a dataspace

Merge dataspace request

```
<m:mergeBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <deleteDataOnMerge>boolean</deleteDataOnMerge>
 <deleteHistoryOnMerge>boolean</deleteHistoryOnMerge>
</m:mergeBranch>
```

with:

Element	*Description*	*Required*
deleteDataOnMerge	This parameter is available for the merge dataspace operation. Sets whether the specified dataspace and its associated snapshots will be deleted upon merge. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.dataDeletionOnCloseOrMerge.default` in the EBX main configuration file [p 354]. See Deleting data and history [p 378] for more information.	No
deleteHistoryOnMerge	This parameter is available for the merge dataspace operation. Sets whether the history associated with the specified dataspace will be deleted upon merge. Default value is `false`. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.historyDeletionOnCloseOrMerge.default` in the EBX main configuration file [p 354]. See Deleting data and history [p 378] for more information.	No

Note

The merge decision step is bypassed during merges performed through data services. In such cases, the data in the child dataspace automatically overrides the data in the parent dataspace.

Merge dataspace response

```
<ns1:mergeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:mergeBranch_Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully.

Close a dataspace or snapshot

Close dataspace or snapshot request

Close dataspace request:

```
<m:closeBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <deleteDataOnClose>boolean</deleteDataOnClose>
 <deleteHistoryOnClose>boolean</deleteHistoryOnClose>
</m:closeBranch>
```

Close snapshot request:

```
<m:closeVersion xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <version>String</version>
 <deleteDataOnClose>boolean</deleteDataOnClose>
</m:closeVersion>
```

with:

Element	*Description*	*Required*
deleteDataOnClose	This parameter is available for the close dataspace and close snapshot operations. Sets whether the specified snapshot, or dataspace and its associated snapshots, will be deleted upon closure. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine this default value by specifying the property `ebx.dataservices.dataDeletionOnCloseOrMerge.default` in the EBX main configuration file [p 354]. See Deleting data and history [p 378] for more information.	No
deleteHistoryOnClose	This parameter is available for the close dataspace operation. Sets whether the history associated with the specified dataspace will be deleted upon closure. Default value is `false`. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.historyDeletionOnCloseOrMerge.default` in the EBX main configuration file [p 354]. See Deleting data and history [p 378] for more information.	No

Close dataspace or snapshot response

Close dataspace response:

```
<ns1:closeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:closeBranch_Response>
```

Close snapshot request:

```
<ns1:closeVersion_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:closeVersion_Response>
```

Replication refresh

Replication refresh request

```
<m:replicationRefresh_${schema} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <instance>String</instance>
 <unitName>String</unitName>
</m:replicationRefresh_${schema}>
```

with:

Element	*Description*	*Required*
branch	See the description under Common parameters [p 636].	Yes
instance	See the description under Common parameters [p 636].	Yes
unitName	Name of the replication unit. **See also** *Replication refresh information* [p 260]	Yes

Replication refresh response

```
<ns1:replicationRefresh_${schema}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:replicationRefresh_${schema}Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully.

97.3 **Operations on data workflows**

Parameters for data workflows operations are retrieved from the SOAP header in the session.

Deprecated since version 5.7.0 to define parameters in the SOAP message body.

See session parameters [p 597] for more information.

Element	Description	Required
parameters	*Deprecated since version 5.7.0* While it remains available for backward compatibility, it will eventually be removed in a future major version. **Note** The `parameters` element is ignored if at least one session parameter has been defined.	No
parameter (nested under the `parameters` element). Multiple `parameter` elements may be used.	An input parameter for the workflow.	No
name (nested under the `parameter` element)	Name of the parameter.	Yes
value (nested under the `parameter` element)	Value of the parameter.	No

Start a workflow

Start a workflow from a workflow launcher. It is possible to start a workflow with localized documentation and specific input parameters (with name and optional value).

> **Note**
>
> The workflow creator is initialized from the session and the workflow priority is retrieved from the last published version.

Sample request:

```
<m:workflowProcessInstanceStart xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <publishedProcessKey>String</publishedProcessKey>
 <documentation>
  <locale>Locale</locale>
  <label>String</label>
  <description>String</description>
 </documentation>
</m:workflowProcessInstanceStart>
```

with:

Element	Description	Required
publishedProcessKey	Identifier of the workflow launcher.	Yes
documentation	See the description under Common parameters [p 636].	No
parameters	*Deprecated since version 5.7.0* See the description under Common parameters [p 643].	No

Sample response:

```
<m:workflowProcessInstanceStart_Response xmlns:m="urn:ebx-schemas:dataservices_1.0">
```

```
 <processInstanceKey>String</processInstanceKey>
</m:workflowProcessInstanceStart_Response>
```

with:

Element	Description	Required
processInstanceId	*Deprecated since version 5.6.1* This parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No
processInstanceKey	Workflow identifier.	No

Resume a workflow

Resume a workflow in a wait step from a resume identifier. It is possible to define specific input parameters (with name and optional value).

Sample request:

```
<m:workflowProcessInstanceResume xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <resumeId>String</resumeId>
</m:workflowProcessInstanceResume>
```

with:

Element	Description	Required
resumeId	Resume identifier of the waiting task.	Yes
parameters	*Deprecated since version 5.7.0* See the description under Common parameters [p 643].	No

Sample response:

```
<m:workflowProcessInstanceResume_Response xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <processInstanceKey>String</processInstanceKey>
</m:workflowProcessInstanceResume_Response>
```

with:

Element	Description	Required
status	'00' indicates that the operation has been executed successfully. '20' indicates that the workflow has not been found. '21' indicates that the event has already been received.	Yes
processInstanceKey	Identifier of the workflow. This parameter is returned if the operation has been executed successfully.	No

End a workflow

End a workflow from its identifier.

Sample request:

```
<m:workflowProcessInstanceEnd xmlns:m="urn:ebx-schemas:dataservices_1.0">
```

```
 <processInstanceKey>String</processInstanceKey>
</m:workflowProcessInstanceEnd>
```

with:

Element	Description	Required
processInstanceKey	Identifier of the workflow.	Either this parameter or 'publishedProcessKey' and 'processInstanceId' parameters must be defined.
publishedProcessKey	*Deprecated since version 5.6.1* Due to a limitation this parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No
processInstanceId	*Deprecated since version 5.6.1* Due to a limitation this parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No

Sample response:

```
<m:workflowProcessInstanceEnd_Response xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</m:workflowProcessInstanceEnd_Response>
```

with:

Element	Description	Required
status	'00' indicates that the operation has been executed successfully.	Yes

97.4 Administrative services

Directory services

The services on directory provide operations on the 'Users' and 'Roles' tables of the default directory. To execute an operation related to these services, the authenticated user must be a member of the built-in role 'Administrator'.

The technical dataspace and dataset must be set to `ebx-directory`. For all SOAP operation syntaxes, see Operations generated from a data model [p 615] for more information.

Create a user in the directory

This example of a SOAP insert request adds a user to the EBX directory.

```
<m:insert_user xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>ebx-directory</branch>
 <instance>ebx-directory</instance>
 <data>
 <directory>
  <users>
   <login>login</login>
   <lastName>lastname</lastName>
   <firstName>firstname</firstName>
   <email>firstname.lastname@email.com</email>
   <password>***</password>
   <passwordMustChange>true</passwordMustChange>
   <builtInRoles>
    <administrator>false</administrator>
    <readOnly>false</readOnly>
   </builtInRoles>
   <comments>a comment</comments>
  </users>
 </directory>
 </data>
</m:insert_user>
```

For the insert SOAP response syntax, see insert response [p 625] for more information.

User interface operations

See Application locking [p 387] for more information.

Parameters for operations on the user interface are as follows:

Element	Description	Required
closedMessage	Message to be displayed to users when the user interface is closed to access.	No

Close user interface request

The close operation removes all user sessions that are not acceptable in this mode.

```
<m:close xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <closedMessage>Access is temporarily forbidden.</closedMessage>
</m:close>
```

Close user interface response

```
<ns1:close_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:close_Response>
```

Open user interface request

```
<m:open xmlns:m="urn:ebx-schemas:dataservices_1.0"/>
```

Open user interface response

```
<ns1:open_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:open_Response>
```

System information operation

This operation returns the EBX system information. The information returned is the same as the information contained in the log header `kernel.log` or in the UI tab 'Administration' > 'System Information'. The response contains several keys, labels, and values representing the configuration and status of EBX. To execute this operation, the authenticated user must be a member of the built-in role 'Administrator'.

Parameters

The following parameter is applicable.

Parameter	Description	Required
`details`	Defines attributes that must be applied to response messages. The attribute `locale` (default: EBX default `locale`) defines the language in which the system item messages must be returned.	No, but if specified, the `locale` attribute must be provided.

System information request

This SOAP request will return all EBX instance's system information and format them using "`en_US`" `locale`.

```
<m:systemInformation xmlns:m="urn:ebx-schemas:dataservices_1.0">
    <details locale="en_US" />
</m:systemInformation>
```

System information response

```
<ns1:systemInformation_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
    <bootInfoEBX>
        <label>String</label>
        <infoItem>
            <key>String</key>
            <label>String</label>
            <content>String</content>
            <content>String</content>
            ...
        </infoItem>
        ...
    </bootInfoEBX>
    <repositoryInfo>
        <label>String</label>
        <infoItem>
            <key>String</key>
            <label>String</label>
            <content>String</content>
            <content>String</content>
            ...
        </infoItem>
        ...
    </repositoryInfo>
    <bootInfoVM>
        <label>String</label>
        <infoItem>
            <key>String</key>
            <label>String</label>
            <content>String</content>
            ...
        </infoItem>
        ...
    </bootInfoVM>
</ns1:systemInformation_Response>
```

REST data services

CHAPTER **98**

Introduction

This chapter contains the following topics:

98.1 Overview

REST data services allow external systems to interact with data governed in the TIBCO EBX repository using the RESTful built-in services.

The request and response syntax for built-in services are described in the chapter Built-in RESTful services [p 657].

Built-in REST data services allow to perform operations such as:

- Selecting, inserting, updating, deleting, or counting records
- Selecting or counting history records
- Selecting, updating, or counting dataset values
- Administrative operations to manage access to the UI or to system information

> **Note**
>
> See SOAP and REST comparative [p 655].

98.2 Activation and configuration

REST and SOAP Data services are activated by deploying the `ebx-dataservices` web application along with the other EBX modules. See Java EE deployment overview [p 317] for more information.

In case of specific deployment, for example using reverse-proxy mode, see URLs computing [p 356] for more information.

Currently only protocol HTTP(S) is supported.

98.3 **Interactions**

Input and output message encoding

All input and output messages must be *exclusively* in UTF-8 for REST built-in.

Tracking information

Depending on the data services operation being called, it may be possible to specify session tracking information.

- Example for a RESTful operation, the JSON request contains:

```
{
 "procedureContext": // JSON Object (optional)
 {
    "trackingInformation": "String" // JSON String (optional)
 }, ...
}
```

For more information, see `Session.getTrackingInfo`API in the Java API.

Session parameters

Depending on the data services operation being called, it is possible to specify session input parameters. They are defined in the request body.

Input parameters are available on custom Java components with a session object, such as: triggers, access rules, `custom` web services. They are also available on data workflow operations.

- Example for a RESTful operation, the JSON request contains:

```
{
 "procedureContext": // JSON Object (optional)
 {
    "trackingInformation": "String", // JSON String (optional)
    "inputParameters": // JSON Array (optional)
    [
     // JSON Object for each parameter
     {
      "name" : "String" // JSON String (required)
      "value" : "String" // JSON String (optional)
     },
     ...
    ]
 }, ...
}
```

For more information, see `Session.getInputParameterValue`API in the Java API.

Exception handling

When an error occurs, a JSON exception response is returned to the caller. For example:

```
{
  "code": 999, // JSON Number, HTTP status code
  "errors": [
   {
    "severity": "...",     // JSON String, severity (optional)
    "rowIndex": 999,       // JSON Number, request row index (optional)
    "userCode": "...",     // JSON String, user code (optional)
    "message": "...",      // JSON String, message
    "details": "...",      // JSON String, URL (optional)
    "pathInRecord": "...", // JSON String, Path in record (optional)
    "pathInDataset": "..." // JSON String, Path in dataset (optional)
   }
  ]
}
```

The response contains an HTTP status code and a table of errors. The severity of each error is specified by a character, with one of the possible values (`F`: fatal, `E`: error, `W`: warning, `I`: information).

The HTTP error `422` *(Unprocessable entity)* corresponds to a functional error. It contains a user code under the `userCode` key and is a JSON `String` type.

See also *HTTP codes* [p 663]

See also *`Severity.toParsableString`*API

98.4 Security

Authentication

Authentication is mandatory to access built-in services. Several authentication methods are available and described below. The descriptions are ordered by priority (EBX applies the highest priority authentication method first).

- 'Token Authentication Scheme' method is based on the HTTP-Header `Authorization`, as described in RFC 2617.

  ```
  > Authorization: <tokenType> <accessToken>
  ```

  For more information on this authentication scheme, see Token authentication operations [p 666].

  See also *HTTP Authorization header policy* [p 356]

- 'Basic Authentication Scheme' method is based on the HTTP-Header `Authorization` in base 64 encoding, as described in RFC 2617 (Basic Authentication Scheme).

  ```
  If the user agent wishes to send the userid "Alibaba" and password "open sesame",
  it will use the following header field:
  > Authorization: Basic QWxpYmFiYTpvcGVuIHNlc2FtZQ==
  ```

  > **Note**
  >
  > The `WWW-Authenticate [p 662]` header can be valued with this method.

  See also *HTTP Authorization header policy* [p 356]

- 'Standard Authentication Scheme' is based on the HTTP `Request`. User and password are extracted from request parameters. For more information on request parameters, see Parameters [p 610] section.

  For more information on this authentication scheme, see `Directory.authenticateUserFromLoginPassword`API.

- The 'REST Forward Authentication Scheme' is used only when calling a REST service from a user service [p 560], that reuses the current authenticated session.

  For more information, see Implementing a user service [p 571] making a call to REST data services [p 578].

- 'Specific authentication Scheme' is based on the HTTP `Request`. For example, an implementation can extract a password-digest or a ticket from the HTTP `Request`. See `Directory.authenticateUserFromHttpRequest`API for more information.

- 'Anonymous authentication Scheme' is used only to access the REST services handling the authentication operations. The credentials acquisition, password changes, etc. imply that the user cannot be known yet.

Global permissions

Global access permissions can be independently defined for the REST built-in services access. For more information see Global permissions [p 383].

Lookup mechanism

Because EBX offers several authentication methods, a lookup mechanism based on conditions was set to know which method should be applied for a given request. The method application conditions are evaluated according to the authentication scheme priority. If the conditions are not satisfied, the server evaluates the next method. The following table presents the available authentication methods for each supported protocol and their application conditions. They are ordered from the highest priority to the lowest.

Operation / Protocol	Authentication methods and application conditions
REST / HTTP	Token [p 652] • The HTTP request must hold an `Authorization` header. • `Authorization` header value must start with the word `EBX`. • No `login` is provided in the URL parameters. Basic [p 652] • The HTTP request must hold an `Authorization` header. • `Authorization` header value must start with the word `Basic`. • No `login` is provided in the URL parameters. Standard [p 652] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters. Rest forward [p 652] • The HTTP request must not contain an `Authorization` header. • No `login` is provided in the URL parameters. Specific [p 652] • The HTTP request must not satisfy the conditions of the previous authentication methods. Anonymous [p 653] • None of the previous authentication methods can be applied. • The requested REST service is handling an authentication operation.

In case of multiple authentication methods present in the same request, EBX will return an HTTP code `401 Unauthorized`.

98.5 Monitoring

Data service events can be monitored through the log category `ebx.dataServices`, as declared in the EBX main configuration file. For example, `ebx.log4j.category.log.dataServices= INFO, ebxFile:dataservices`.

See also

Configuring the EBX logs *[p 351]*

TIBCO EBX main configuration file *[p 345]*

98.6 **SOAP and REST comparative**

Operations	**SOAP**	**REST**
Data		
Select or count records (with filter and/or view publication)	X	X
Selector for possible enumeration values (with filter)		X
Insert, update or delete records	X	X
Select or count history records (with filter and/or view publication)		X
Select node values from dataset	X	X
Update node value from dataset		X
Get table or dataset changes between dataspaces or snapshots	X	
Refresh a replication unit	X	
Get credentials for records	X	
Dataspaces		
Create, close, merge a dataspace	X	
Create, close a snapshot	X	
Validate a dataspace or a snapshot	X	
Validate a dataset	X	
Locking a dataspace	X	
Workflow		
Start, resume or end a workflow	X	
Administration		
Manage the default directory content 'Users', 'Roles'... tables.	X	X
Open, close the user interface	X	X

Operations	**SOAP**	**REST**
Select, insert, update, delete operations for administration dataset		X
Select the system information	X	X
Other		
Develop web services from the Java API		X (*)

(*) See REST Toolkit [p 719] for more information.

98.7 Limitations

Date, time & dateTime format

Data services only support the following date and time formats:

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

JMS

- JMS protocol is not supported.

CHAPTER **99**

Built-in RESTful services

This chapter contains the following topics:

99.1 Introduction

The architecture used is called ROA (Resource-Oriented Architecture), it can be an alternative to SOA (Service-Oriented Architecture). The chosen resources are readable and/or writable by third-party systems, according to the request content.

The HATEOAS approach of the built-in RESTful services also allows to experience an intuitive and straightforward navigation, which implies that the data details could be obtained through a link.

> **Note**
>
> All operations are stateless.

99.2 Request

This chapter describes the elements to use in order to build a conform REST request, such as: the HTTP method, the URL format, the header fields and the message body.

See also

Interactions *[p 651]*

Security *[p 652]*

HTTP method

Considered HTTP methods for built-in RESTful services, are:

- `GET`: used to select master data defined in the URL (the URL size limit depends on the application server or on the browser, that must be lower than or equal to 2KB).
- `POST`: used to insert one or more records in a table or to select the master data defined in the URL (the size limit is 2MB or more depending on the application server. Each parameter is limited to a value containing 1024 characters).
- `PUT`: used to update the master data defined in the URL.
- `DELETE`: used to delete either the record defined in the URL or multiple records defined with the table URL and the record table in the message body.

URL

REST URL contains:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/{categoryVersion}/
{specificPath}[:{extendedAction}]?{queryParameters}
```

Where:

- `<ebx-dataservices>` corresponds to the 'ebx-dataservices.war' web application's path. The path is composed by multiple, or none, URI segments followed by the web application's name.
- `{category}` corresponds to the operation category [p 659].
- `{categoryVersion}` corresponds to the category version: current value is `v1`.
- `{specificPath}` corresponds to a specific path inside the category.
- `{extendedAction}` corresponds to the extended action name (optional).
- `{queryParameters}` corresponds to common [p 661] or dedicated operation parameters passed by the URL.

Operation category

It specializes the operation, it is added in the path of the URL in `{category}` and it takes one of the following values:

`admin`	Administration operations reserved to administrators. For more information, see: Administration operations [p 664].
`auth`	Manage token authentication method. For more information, see: Token authentication operations [p 666] and Token Authentication Scheme [p 652].
`data`	Lists dataset content, requests a table, a record or a field record content, including modified operations on dataset node, table, record and record field. For more information, see: Data operations [p 669].
`history`	Lists history dataset content, requests a history table, a history of a record or a history record. For more information, see: Data operations [p 669]. **See also***History* *[p 251]*

Header fields

These header field definitions are used by TIBCO EBX.

`Accept`	Used to specify content types by order of preference to be used in the response, the first supported one will be chosen and specified in the response header `Content-Type`. Currently, the only supported one is `application/json`. If none is supported, the result depends on the `ebx.dataservices.rest.request.checkAccept` property: • If `true`, an HTTP error response is returned with code `406`. • If `false`, the response is returned with the default content type, that is `application/json`. **See also***Configuring data services* *[p 354]*
`Accept-Language`	Used for specifying the preferred locale for the response. The supported locales are defined in the schema model. If none of the preferred locale are supported, the default locale for the current model is used.
`Authorization`	Used for 'Basic Authentication Scheme' and 'Token Authentication Scheme' methods, otherwise the request is rejected. **See also***Authentication* *[p 652]*
`Content-Type`	Used to specify the request body media type. The supported types are `application/json` and `application/x-www-form-urlencoded`. The request value is checked and if it is not supported, then an HTTP error message is returned with the code `415` *(Unsupported media type)*. **See also***Configuring data services* *[p 354]*
`X-Requested-With`	If present and in case of authentication failure, prevents the addition of the `WWW-Authenticate` header in the response. **See also***Response header `WWW-Authenticate`* *[p 662]*

See RFC2616 for more information about HTTP Header Field Definitions.

Common parameters

These optional parameters are available for all data service operations.

Parameter	Description
`disableRedirectionToLastBroadcast`	This parameter only has impact on a D3 architecture. If `true`, access to a delivery dataspace on a D3 primary node is not redirected to the last broadcast snapshot. Otherwise, access to such a dataspace is always redirected to the last broadcast snapshot. If the specified dataspace is not a delivery dataspace on a D3 primary node, this parameter is ignored. `Boolean` type value. If this parameter is not present, the default is `false` (redirection to a D3 master enabled), unless the configuration property ebx.dataservices.disableRedirectionToLastBroadcast.default [p 354] has been set. **See also***Primary node* *[p 427]*
`indent`	Specifies if the response should be indented, to be easier to read for a human. `Boolean` type, default value is `false`.

Message body

It contains the request data using the JSON format, see JSON Request body [p 694].

> **Note**
>
> Requests may define a message body only when using `POST` or `PUT` HTTP methods.

99.3 Response

This chapter describes the responses returned by built-in RESTful services.

- See Exception handling [p 651] for details on standard error handling (where the HTTP code is greater than or equal to `300`).

Header fields

These header field definitions are used by EBX.

`Content-Language`	Indicates the locale used in the response for labels and descriptions.
`Content-Type`	Indicates the response body content type.
`Location`	If a new record has been successfully inserted, the query URL for this record is returned by this field.
`WWW-Authenticate`	This header field is added to the HTTP response when authentication fails with the 401 *(Unauthorized)* HTTP code. Its value consists of a list with at least one authentication method applicable to the request URI. It is present if and only if the following conditions are verified: • the 'Basic Authentication Scheme' method is enabled and • the `X-Requested-With` HTTP header is not present. If the client is able to interpret the authentication method, it is possible to resubmit the request providing the appropriate credentials. The administration property ebx.dataservices.rest.auth.tryBasicAuthentication [p 354] must be set to `true`. **See also** *Request header `X-Requested-With` [p 660]* *Authentication [p 652]*

HTTP codes

HTTP code	Description
`200` *(OK)*	The request has been successfully handled.
`201` *(Created)*	A new record has been created, in this case, the header field `Location` is returned with its resource URL.
`204` *(No content)*	The request has been successfully handled but no response body is returned.
`400` *(Bad request)*	the request URL or body is not well-formed or contains invalid content.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	Permission was denied to read or modify the specified resource for the authenticated user. This error is also returned when the user: • is not allowed to modify a field mentioned in the request message body. • is not allowed to access the REST connector. For more details, see Global permissions [p 653].
`404` *(Not found)*	The resource specified in the URL cannot be found.
`406` *(Not acceptable)*	Content type defined in the request's `Accept` parameter is not supported. This error can be returned only if the EBX property `ebx.rest.request.checkAccept` is set to `true`.
`409` *(Conflict)*	A concurrent modification has occurred. **See also***Optimistic locking* [p 689]
`415` *(Unsupported media type)*	The request content is not supported, the request header value `Content-Type` is not supported by the operation.
`422` *(Unprocessable entity)*	The new resource's content cannot be accepted for semantic reasons.
`500` *(Internal error)*	Unexpected error thrown by the application. Error details can usually be found in EBX logs.

Message body

The response body content's format depends on the HTTP code value:

- HTTP codes from `200` included to `300` excluded: the content format depends on the associated request (JSON [p 697] samples).

  With the exception of code `204` *(No content)*.
- HTTP codes greater than or equal to `300`: the content describes the error. See JSON [p 651] for details on the format.

99.4 Administration operations

Administration operations are related to:

- the administration category.
- the administration dataspaces accessible through the `data` category.

> **Note**
>
> administration category and administration dataspaces can only be used by administrators.

Directory operations

The EBX default directory configuration is manageable with built-in RESTful services. The users and roles tables, the mailing lists and other objects are concerned. For more information, see Users and roles directory [p 399].

> **Note**
>
> Triggers are present on the directory's tables, ensuring the data consistency.

The URL format is:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/Bebx-directory/ebx-directory
```

Directory configuration operations

The EBX default directory configuration is manageable like dataset nodes. It can be accessed and modified through the `data` category operations. Each field is self-described when metadata is requested.

See select [p 669] and update [p 682] operations for more information.

Mailing lists operations

There are two default mailing lists that can be configured in the EBX directory: one for everybody and one for administrators. These lists can be handled like dataset nodes through the `data` category operations.

See select [p 669] and update [p 682] operations for more information.

Directory users operations

Users can be manipulated like records of the `data` category using the operations of the latter. For security purposes, an administrator cannot delete himself. The user's salutation must be chosen among the ones available in the 'salutations' table.

See select [p 669], update [p 682], insert [p 677] and delete [p 684] operations for more information.

Directory roles operations

Roles are records of the `data` category and can be managed with its operations. EBX roles are assigned to users through the 'usersRoles' association table. 'usersRoles' is automatically fed when the directory is administered through the user interface. However, it is not the case through data services and role assignments require manual operations. Roles inclusions are specified in the 'rolesInclusions' association table. As for the 'usersRoles' table, the management of roles inclusions requires manual operations. Each table is self-descriptive when metadata is requested.

See select [p 669], update [p 682], insert [p 677] and delete [p 684] operations for more information.

User interface operations

The EBX user interface can be opened or closed to users for maintenance needs. Handled information is similar to what is contained in the UI tab 'Administration' > 'User interface configuration' > 'Advanced perspective' > 'Graphical interface configuration' > 'Application locking'.

URL format is:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/Bebx-manager/ebx-manager/
domain/toolStatus
```

See also*Application locking [p 387]*

Retrieve user interface state

User interface status and the unavailability message are accessible like dataset nodes.

See Select operation [p 669] and the JSON [p 704] example, for more information.

Open or close user interface

User interface status and the unavailability message can be modified like dataset nodes using the update operation. To open the user interface set the content of `toolStatus` to `true`, or to `false` to close it.

See Update operation [p 682] and the JSON [p 697] examples, for more information.

System information operation

This operation returns system information on the EBX server. This is accepted for `GET` and `POST` HTTP methods. Warning: no update will be possible in the `POST` HTTP method because the request body is ignored. The information returned is the same as the information contained in the log header `kernel.log` or in the UI tab 'Administration' > 'System Information'. The response contains several keys, labels, and values representing the configuration and status of EBX. The mode of representation of the response may be flat or hierarchical.

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/admin/v1/systemInformation
```

See also

TIBCO EBX main configuration file [p 345]

Repository administration [p 372]

Parameters

The following parameter is applicable.

Parameter	Description
`systemInformationMode`	Specifies the returned mode: • `flat`: A flat representation under the following information groups: `bootInfoEBX`, `repositoryInfo` and `bootInfoVM`. • `hierarchical`: A hierarchical representation. `String` type, default value is `flat`.

HTTP codes

HTTP code	Description
`200` *(OK)*	The system information was successfully returned.
`400` *(Bad request)*	The request is not correct, it contains one of the following errors: • the HTTP method is not `GET` nor `POST`, • the HTTP parameter `systemInformationMode` is not correct, • the operation is not supported. • the request path is invalid.
`403` *(Forbidden)*	The user is not an administrator.

Response body

It is returned, if and only if, the HTTP code is `200 (OK)`. The content structure depends on the provided parameter `systemInformationMode` or its default value.

See the JSON [p 698] example of the flat representation.

See the JSON [p 698] example of the hierarchical representation.

99.5 Token authentication operations

These operations allow to create or revoke an authentication token. Authentication tokens have a timeout period. If a token is not used to access the EBX server within this period, it will automatically be revoked. This timeout period is refreshed on each access to EBX server.

> **Note**
>
> The token timeout is modifiable through the administration property ebx.dataservices.rest.auth.token.timeout [p 354] (the default value is 30 minutes).

Create token operation

This operation requires using the `POST` HTTP method with a request containing the user's credentials and, optionally, session parameters [p 651].

URL format is:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/token:create
```

Message body

A message body must be defined in the HTTP request. It necessarily contains one of the following set of data:

- A `login` and a `password` value. Both JSON attributes are mandatory and of `String` types.

  See `Directory.authenticateUserFromLoginPassword`API for more information.

- The `specific` JSON attribute set to `true`. When activated, this flag allows to performed a user authentication against the whole HTTP request. Warning, even if `login` and `password` attributes are defined in the JSON request's body, setting `specific` to `true` lead to a specific user authentication.

  See `Directory.authenticateUserFromHttpRequest`API for more information.

See the JSON [p 694] examples of a token creation request.

HTTP codes

HTTP code	Description
`200` *(OK)*	The token was successfully created.
`400` *(Bad request)*	For one of the following reasons: • the syntax is not correct, • the HTTP method is not `POST`, • the operation is not supported.
`401` *(Unauthorized)*	For one of the following reasons: • The `login` and/or `password` is/are incorrect. • Authentication data for other methods is defined.
`422` *(Unprocessable entity)*	For one of the following reasons: • `PasswordMustChange`: The password must be changed (only available with the default directory). See Change password operation [p 668]. • `RestrictedAccess`: User access is closed for maintenance or other actions (reserved to administrators).

Response body

If the HTTP code is `200 (OK)`, the body holds the token value and its type.

See the JSON [p 699] example of a token creation response.

The token can later be used to authenticate a user by setting the HTTP-Header `Authorization` accordingly.

See also *'Token authentication Scheme' method [p 652]*

Change password operation

This operation modifies the password of an existing user account. It can be used in an authenticated context: `login` parameter, if present, is checked against the current session or taken from it, if absent. It could also be used in an unauthenticated context, for example when the Create token operation [p 666] aborts with the HTTP code `422` *(Unprocessable entity)* with reason: `PasswordMustChange`.

It requires the use of:

- the EBX default directory
- the `POST` HTTP method
- the message body containing the structure specified below

URL format is:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/user:changePassword
```

Message body

The message body must be defined in the request. It necessarily contains a `password` and a `passwordNew`, the `login` is optional (all are `String`).

See the JSON [p 694] example of a password change and token creation request.

HTTP codes

HTTP code	**Description**
`204` *(No content)*	The password has been changed.
`400` *(Bad request)*	For one of the following reasons: • the EBX default directory is required, • the syntax is not correct, • the HTTP method is not `POST`, • the provided `login` and the user's session one mismatch • the operation is not supported.
`401` *(Unauthorized)*	For the following reason: • The `login` and/or `password` is/are incorrect.
`422` *(Unprocessable entity)*	For one of the following reasons: • `PasswordChangeAbort`: `passwordNew` is empty. • `PasswordChangeAbort`: `passwordNew` is equal to `password`. • `PasswordChangeAbort`: `password` is incorrect. • `RestrictedAccess`: User access is closed for maintenance or other actions (reserved to administrators).

Response body

If HTTP code `204 (No content)` is returned, then the password has been modified.

Revoke token operation

This operation requires using the `POST` HTTP method. No message body is needed.

URL format is:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/auth/v1/token:revoke
```

Header fields

`Authorization`	This field is required, `tokenType` and `accessToken` fields must have the values returned from the "token create" operation. `> Authorization: <tokenType> <accessToken>`

HTTP codes

HTTP code	**Description**
`204` *(No content)*	The token has been revoked successfully.
`400` *(Bad request)*	For one of the following reasons: • the configuration is not activated, • the syntax is incorrect, • the HTTP method is not `POST`, • the operation is not supported.
`401` *(Unauthorized)*	Authentication has failed.

99.6 **Data operations**

Operations from the `data` operation category concern the datasets, the dataset fields, tables, records or record fields.

Operations from the `history` category and concern historized content from datasets, tables, records or the record fields.

Select operation

Select operation may use one of the following methods:

- `GET` HTTP method,
- `POST` HTTP method without message body or
- `POST` HTTP method with message body and optionally session parameters [p 651].

URL formats are:

- **Dataset tree**, depending on operation category [p 659]:

  The `data` category returns the hierarchy of the selected dataset, this includes group and table nodes.

The `history` category returns the hierarchy of the selected history dataset, this includes the pruned groups for history table nodes only.

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
{dataset}
```

> **Note**
>
> Terminal nodes and sub-nodes are not included.

- **Dataset node**: the `data` category returns the terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
  {pathInDataset}
  ```

  > **Note**
  >
  > Not applicable with the `history` category.

- **Table**, depending on operation category [p 659]:

  the `data` category returns the table content and/or metadata, current page records and URLs for pagination.

  The `history` category returns the history table content and/or metadata, current page records and URLs for pagination.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}
  ```

  See also*Count operation [p 686]*

- **Record**, depending on operation category [p 659]:

  the `data` category returns the record content and/or metadata.

  The `history` category returns history record content and/or metadata.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}/{encodedPrimaryKey}
  ```

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}?primaryKey={xpathExpression}
  ```

  > **Note**
  >
  > The record access by the primary key (`primaryKey` parameter) is limited to its root node. It is recommended to use the encoded primary key, available in the `details` field in order to override this limitation. Similarly, for a history record, use the encoded primary key, available in the `historyDetails` field.

- **Field**, depending on operation category [p 659]:

  the `data` category returns the field record content where structure depends on its type.

The `history` category returns the field history record content where structure depends on its type.

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
{dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
```

> **Note**
>
> The field must be either an association node, a selection node, a terminal node or above.

Where:

- `{category}` corresponds to the operation category [p 659].
- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the dataset node, that can be a group node or a table node.
- `{encodedPrimaryKey}` corresponds to the encoded representation of the primary key.

  See also *`RESTEncodingHelper`* API

- `{xpathExpression}` corresponds to the record primary key, using the XPath expression.
- `{pathInRecord}` corresponds to the path starting from the table node.

Parameters

The following parameters are applicable to the select operation.

Parameter	Description
`includeContent`	Includes the `content` field with the content corresponding to the selection. `Boolean` type, default value is `true`.
`includeDetails`	Includes the `details` field in the metadata and the response, for each indirect reachable resource. The returned value corresponds to its URL resource. Type `Boolean`, default value is `true`. **See also***includeMeta* [p 672]
`includeHistory`	Includes those fields for a historized content: • The `history` property in the metadata. The returned value corresponds to a boolean value. • The `historyDetails` property in the content and for each indirectly reachable resource. This point is coupled to the includeDetails [p 672] parameter. The returned value corresponds to its URL resource. `Boolean` type, default value is `false`. **Note** The `includeHistory` parameter is ignored in the `history` category, the default value is `true`. **See also** *includeMeta* [p 672] *includeDetails* [p 672] *`AdaptationTable.getHistory`*API
`includeLabel`	Includes the `label` field associated with each simple type `content`. Possible values are: • `yes`: the `label` is included for the foreign key, enumeration, record and selector values. • `all`: the `label` field is included, as for the `yes` value and also for the Content of simple type [p 711]. **See also***`SchemaNode.displayOccurrence`*API • `no`: the `label` field is not included (integration use case). `String` type, default value is `yes`. **Note** The `label` field is not included if it is equal to the `content` field.
`includeMeta`	Includes the `meta` field corresponding to the description of the structure returned in the `content` field. `Boolean` type, default value is `false`.

Parameter	Description
	See also *includeHistory* [p 672] *includeDetails* [p 672]
`includeMergeInfo`	Includes the `merge_info` corresponding to a field of the technical data of an history transaction and has a potentially high access cost. `Boolean` type, default value is `true`. **Note** This parameter is ignored with the `data` category. **See also***REST access to history table* [p 303]
`includeSelector`	Includes the `selector` field in the response, for each indirect reachable resource. The returned value corresponds to its URL resource. Type `Boolean`, default value is `true`. **See also***selector* [p 676]
`includeSortCriteria`	Includes the `sortCriteria` field corresponding to the list of sort criteria applied. The sort criteria parameters are added by using: • sort [p 675] • sortOnLabel [p 675] • viewPublication [p 676] `Boolean` type, default value is `false`. Example JSON [p 707]
`includeTechnicals`	Includes the internal technical data. `Boolean` type, default value is `false`. **See also***Technical data* [p 716] **Note** This parameter is ignored with the `history` category.
`includeValidation`	Includes the validation report corresponding to the selection. `Boolean` type, default value is `false`. **Note** This parameter is ignored with the `history` category. **See also***includeDetails* [p 672]

Table parameters

The following parameters are applicable to tables, associations and selection nodes.

Parameter	Description
`filter`	XPath predicate [p 227] expression defines the field values to which the request is applied. If empty, all records will be retrieved. `String` type value. **Note** The history code operation value is usable with `ebx-operationCode` path field from the `meta` section associated with this field.
`historyMode`	Specifies the filter context applied on table. `String` type, possible values are: • `CurrentDataSpaceOnly`: history in current dataspace • `CurrentDataSpaceAndAncestors`: history in current dataspace and ancestors • `CurrentDataSpaceAndMergedChildren`: history in current dataspace and merged children • `AllDataSpaces`: history in all dataspaces The default value is `CurrentDataSpaceOnly`. **See also***`history`* *[p 659]* **Note** This parameter is ignored with the `data` category.
`includeOcculting`	Includes the records in occulting mode. `Boolean` type, default value is `false`.
`primaryKey`	Search a record by a primary key, using the XPath expression. The XPath predicate [p 227] expression should only contain field(s) of the primary key and all of them. Fields are separated by the operator `and`. A field is represented by one of the following possibilities according to its simple type: • For the `date`, `time` or `dateTime` types: use the `date-equal(path, value)` • For other types: indicate the `path`, the `=` operator and the `value`. Example with a composed primary key: `./pk1i=1 and date-equal(./pk2d,'2015-11-13')` The response will only contain the corresponding record, otherwise an error is returned. Consequently, the other table parameters are ignored (as filter [p 674], viewPublication [p 676], sort [p 675], etc.) `String` type value.
`pageFirstRecordFilter`	*Deprecated since version 5.9.0*, replaced by `pageRecordFilter`
`pageRecordFilter`	Specifies the record XPath predicate [p 227] expression filter of the page. `String` type value.

Parameter	Description
`pageAction`	Specifies the pagination action to perform from page defined by `pageRecordFilter`. `String` type, possible values are: • `first` • `previous` • `next` • `last` **See also** *`RequestPagination`* API
`pageSize`	Specifies the number of records per page. `Integer` type, default value is based on the user preferences. `String` type, the `unbounded` value can be defined to return all records. Only for this case, no pagination context is returned.
`sort`	Specifies that the operation result will be sorted according to the specified criteria. The criteria are composed of one or more criteria, the result will be sorted by priority from the left. A criterion is composed of the field path and, optionally, the sorting order (ascending or descending, on value or on label). This parameter can be combined with: 1. the sortOnLabel [p 675] parameter as a new criteria added after the `sort`. 2. the viewPublication [p 676] parameter as a new criteria added after the `sort`. The value structure is as follows: `<path1>:<order>;...;<pathN>:<order>` Where: • `<path1>` corresponds to the field path in priority `1`. • `<order>` corresponds to the sorting order, with one of the following values: ◦ `asc`: ascending order on value (default), ◦ `desc`: descending order on value, ◦ `lasc`: ascending order on label, ◦ `ldesc`: descending order on label. `String` type, the default value orders according to the primary key fields (ascending order on value). **Note** The history code operation value is usable with the `ebx-operationCode` path field from the `meta` section associated with this field. **See also** *`Request.setSortCriteria`* API
`sortOnLabel`	Specifies that the operation result will be sorted according to the record label. This parameter can be combined with: 1. the sort [p 675] parameter as a new criteria added before the `sortOnLabel`. 2. the viewPublication [p 676] parameter as a new criteria added after the `sortOnLabel`. The value structure is as follows: `<order>`

Parameter	Description
	Where: • `<order>` corresponds to the sorting order, with one of the following values: • `lasc`: ascending order on label, • `ldesc`: descending order on label. The behavior of this parameter is described in the section defaultLabel [p 495]. `String` type value. **See also***Limitation* [p 692]
`viewPublication`	Specifies the name of the published view. This parameter can be combined with: 1. the filter [p 674] parameter as the logical `and` operation. 2. the sort [p 675] parameter as a new criteria added before the `viewPublication`. 3. the sortOnLabel [p 675] parameter as new criteria added before the `viewPublication`. The behavior of this parameter is described in the section EBX as a Web Component [p 196]. `String` type value.

Selector parameters

The following parameters are only applicable to fields that return an enumeration, foreign key or `osd:resource` (Example JSON [p 714]). By default, a pagination mechanism is enabled.

Parameter	Description
`selector`	Specifies whether: • `true`: returns all possible values (includes their labels) • `false`: returns the current values for the current field. `Boolean` type, default value is `false`. **Note** This parameter is ignored with the `history` category.
`firstElementIndex`	Specifies the index of the first element returned by the selector. Must be an integer higher than or equal to `0`. `Integer` type, default value is `0`.
`pageSize`	Specifies the number of elements per page. `Integer` type, default value is based on the user preferences. `String` type, the `unbounded` value can be defined to return all values. Only for this case, no pagination context is returned.
`selectorFilter`	Specifies the filter of the selector. `String` type value, the syntax complies with the Text search [p 114].

HTTP codes

HTTP code	Description
`200` *(OK)*	The selected resource is successfully retrieved.
`400` *(Bad request)*	The request is incorrect. This occurs when: • The selected field in a record or a dataset is sub-terminal, • The XPath predicate of the `filter` parameter is malformed or contains unfilterable nodes. • The XPath predicate of the `primaryKey` parameter is malformed or is not a record primary key. • The sort criteria of the `sort` parameter have an invalid syntax or contain unsortable nodes. • `pageAction` parameter value is not included in allowed values, or `pageRecordFilter` is malformed or non-existent when selecting next or previous page. • `pageSize` parameter value is below 2, or is a string different from `unbounded`. • The table view for the `viewPublication` parameter is either hierarchical, non-existent or non-published. • The `selector` parameter is used for a non-enumerated node, or the `firstElementIndex` is negative, higher than or equal to the number of values.
`403` *(Forbidden)*	The selected resource is hidden for the authenticated user.
`404` *(Not found)*	The selected resource is not found.

Response body

After a successful dataset, table, record or field selection, the result is returned in the response body. The content depends on the provided parameters and selected data.

Example: JSON [p 699].

Insert operation

Insert operation uses the `POST` HTTP method. A body message is required to specify data. This operation supports the insertion of one or more records in a single transaction. Moreover, it is also possible to update record(s) through parameterization.

- **Record**: insert a new record or modify an existing one in the selected table.
- **Record table**: insert or modify one or more records in the selected table, while securing a consistent answer. Operations are executed sequentially, in the order defined on the client side. When an error occurs during a table operation, all updates are cancelled and the client receives an error message with detailed information.

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
{pathInDataset}
```

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.

- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the table node.

Parameters

The following parameters are applicable with the insert operation.

Parameter	Description
`includeDetails`	Includes the `details` field in the answer for access to the details of the data. The returned value corresponds to its URL resources. Type `Boolean`, the default value is `false`. **Note** Only applicable on the **record table**.
`includeForeignKey`	Includes the `foreignKey` field in the answer for each record. The returned value corresponds to the value of a foreign key field that was referencing this record. `Boolean` type, the default value is `false`. **Note** Only applicable on the **record table**.
`includeLabel`	Includes the `label` field in the answer for each record. Possible values are: • `yes`: the `label` field is included. • `no`: the `label` field is not included (use case: integration). `String` type, the default value is `no`. **Note** Only applicable on the **record table**.
`updateOrInsert`	Specifies the behavior when the record to insert already exists: • If `true`: the existing record is updated with new data. For a request on a **record table**, the `code` field is added to the report in order to specify if this is an insert `201` or an update `204`. • If `false` (default value): a client error is returned and the operation is aborted. `Boolean` type value.
`byDelta`	Specifies the behavior for setting value of nodes that are not defined in the request body. This is described in the Update modes [p 716] section. `Boolean` type, the default value is `true`. **Note** Applicable on record in update mode and if the updateOrInsert [p 679] parameter is `true`.
`blockingConstraintsDisabled`	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. `Boolean` type, default value is `false`.

Parameter	Description
	See Blocking and non-blocking constraints [p 522] for more information.

Message body

The request must define a message body. The format depends on the inserted object type:

- **Record**: similar to the select operation of a record but without the record's header (example JSON [p 695]).
- **Record table**: Similar to the select operation on a table but without the pagination information (example JSON [p 696]).

See also*Inheritance [p 690]*

HTTP codes

HTTP code	Description
`200`*(OK)*	If the request relates to a **record table**. The insert request was applied successfully, an optional report is returned in the response body.
`201` *(Created)*	If the request relates to a **record**. A new record has been created, in this case, the header field `Location` is returned with its resource URL.
`204` *(No content)*	If the request relates to a **record**. Only available if `updateOrInsert` is `true`, an existing record has been successfully updated, in this case, the header field `Location` is returned with its resource URL.
`400` *(Bad request)*	The request is incorrect. This occurs when the body message structure does not comply with what was mentioned in Message body [p 680].
`403` *(Forbidden)*	Authenticated user is not allowed to create a record or the request body contains a read-only field.
`404` *(Not found)*	The selected resource is not found.
`409` *(Conflict)*	Concurrent modification, only available if `updateOrInsert` is `true`, the Optimistic locking [p 689] is activated and the content has changed in the meantime, it must be reloaded before update.
`422` *(Unprocessable entity)*	The request cannot be processed. This occurs when: • A blocking validation error occurs (only available if `blockingConstraintsDisabled` is `false`). • The record cannot be inserted because a record with the same primary key already exists (only available if `updateOrInsert` is `false`). • The record cannot be inserted because the definition of the primary key is either non-existent or incomplete. • The record cannot be updated because the value of the primary key cannot be modified.

Response body

The response body format depends on the inserted object type:

- **Record**: is empty if the operation was executed successfully. The header field `Location` is returned with its URL resource.
- **Record table**: (optional) contains a table of element(s), corresponding to the insert operation report (example JSON [p 714]). This report is automatically included in the response body, if at least one of the following options is set:
  - `includeForeignKey`
  - `includeLabel`
  - `includeDetails`

See also *Inheritance* [p 690]

Update operation

This operation allows the modification of a single dataset or record. The `PUT` HTTP method must be used. Available URL formats are:

- **Dataset node**: modifies the values of terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
  {pathInDataset}
  ```

- **Record**: modifies the content of selected record.

  > **Note**
  >
  > Also available for `POST` HTTP methods. In this case, the URL must correspond to the table by setting the parameter `updateOrInsert` to `true`.

- **Field**: modifies the field content.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
  {pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
  ```

  > **Note**
  >
  > The field must be either a terminal node or above.

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the dataset node:
  - For dataset node operations, this must be any terminal node or above except table node,
  - For record and field operations, this corresponds to the table node.
- `{encodedPrimaryKey}` corresponds to the encoded representation of the primary key.

  See also *`RESTEncodingHelper`* API

- `{pathInRecord}` corresponds to the path starting from the table node.

Parameters

Here are the parameters applicable with the update operation.

Parameter	Description
`blockingConstraintsDisabled`	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. `Boolean` type, default value is `false`. See Blocking and non-blocking constraints [p 522] for more information.
`byDelta`	Specifies the behavior for setting value of nodes that are not defined in the request body. This is described in the Update modes [p 716] section. `Boolean` type, the default value is `true`.
`checkNotChangedSinceLastUpdateDate`	Timestamp in datetime format used to ensure that the record has not been modified since the last read. Also see the Optimistic locking [p 689] section. `DateTime` type value.

Message body

The request must define a message body.

The structure is the same as for:

- the dataset node (sample JSON [p 694]),
- the record (sample JSON [p 695]),
- the record fields (sample JSON [p 695]),

depending on the updated scope, by only keeping the `content` entry.

See also*Inheritance [p 690]*

HTTP codes

HTTP code	Description
`204` *(No content)*	The record, field or dataset node has been successfully updated.
`400` *(Bad request)*	The request is incorrect. This occurs when the body request structure does not comply.
`403` *(Forbidden)*	Authenticated user is not allowed to update the specified resource or the request body contains a read-only field.
`404` *(Not found)*	The selected resource is not found.
`409` *(Conflict)*	Concurrent modification, the Optimistic locking [p 689] is activated and the content has changed in the meantime, it must be reloaded before the update.
`422` *(Unprocessable entity)*	The request cannot be processed. This occurs when: • A blocking validation error occurs (only available if `blockingConstraintsDisabled` is `false`). • The record cannot be updated because the value of the primary key cannot be modified.

Delete operation

The operation uses the `DELETE` HTTP method.

Two URL formats are available:

- **Record**: delete a record specified in the URL.

  `http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}`

- **Record table**: deletes several records in the specified table, while providing a consistent answer. This mode requires a body message containing a record table. The deletions are executed sequentially, according to the order defined in the table. When an error occurs during a table operation, all deletions are cancelled and an error message is displayed with detailed information.

  `http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/{pathInDataset}`

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the table node.
- `{encodedPrimaryKey}` corresponds to the encoded representation of the primary key.

  See also *`RESTEncodingHelper`* API

In a child dataset context, this operation modifies the `inheritanceMode` property value of the record as follows:

- A record with inheritance mode set to `inherit` or `overwrite` becomes `occult`.
- A record with inheritance mode set to `occult` becomes `inherit` if the `inheritIfInOccultingMode` operation parameter is set to `true` or is undefined, otherwise there is no change.
- A record with inheritance mode set to `root` is simply deleted.

See also*Inheritance* *[p 690]*

Parameters

Here are the following parameters applicable with delete operation.

Parameter	Description
`includeOcculting`	Includes occulted records. `Boolean` type, the default value is `false`.
`inheritIfInOccultingMode`	*Deprecated since version 5.8.1* While it remains available for backward compatibility reasons, it will eventually be removed in a future version. Inherits the record if it is in occulting mode. `Boolean` type, the default value is `true`.
`checkNotChangedSinceLastUpdateDate`	Timestamp in datetime format used to ensure that the record has not been modified since the last read. Also see the Optimistic locking [p 689] section. `DateTime` type value.
`blockingConstraintsDisabled`	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. `Boolean` type, default value is `false`. See Blocking and non-blocking constraints [p 522] for more information.

Message body

The request must define a message body only when deleting several records:

- **Record table**: The message contains a table of elements related to a record, with for each element one of the following properties:
  - `details`: corresponds to the record URL, it is returned by the select operation.
  - `primaryKey`: corresponds to the primary key of the record, using the XPath expression.
  - `foreignKey`: corresponds to the value that a foreign key would have if it referred to a record.

  See also*`PrimaryKey`*API

  Example JSON [p 696].

HTTP codes

HTTP code	Description
`200` *(OK)*	The operation has been executed successfully. A report is returned in the response body.
`400`*(Bad request)*	The request is incorrect. This occurs when: • the structure of the message body does not comply with Message body [p 685]. • the message body contains a record table while the URL specifies a record.
`403` *(Forbidden)*	Authenticated user is not allowed to delete or occult the specified record.
`404` *(Not found)*	The selected record is not found. In the child dataset context, it should be necessary to use the `includeOcculting` parameter.
`409` *(Conflict)*	Concurrent modification, The Optimistic locking [p 689] is activated and the content has changed in the meantime, it must be reloaded before deleting the record. The parameter value `checkNotChangedSinceLastUpdateDate` exists but does not correspond to the actual last update date of the record.
`422` *(Unprocessable entity)*	Only available if `blockingConstraintsDisabled` is `false`, the operation fails because of a blocking validation error.

Response body

After a successful record deletion or occulting, a report is returned in the response body. It contains the number of deleted, occulted and inherited record(s).

Example JSON [p 715].

Count operation

Count operation may use one of the following methods:

- `GET` HTTP method,
- `POST` HTTP method without message body or
- `POST` HTTP method with message body but without `content` field on root.

The URL formats are:

- **Dataset node**: the `data` category returns the number of terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
  {pathInDataset}?count=true
  ```

  > **Note**
  >
  > Not applicable with the `history` category.

- **Table** depending on the operation category [p 659]:

  the `data` category returns the number of table records.

The `history` category returns the number of table history records.

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
{dataset}/{pathInDataset}?count=true
```

- **Field** depending on the operation category [p 659]:

  the `data` category counts the record fields.

  The `history` category counts the history record field.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}?count=true
  ```

  > **Note**
  >
  > The field must be either an association node, a selection node, a terminal node or above.

Where:

- `{category}` corresponds to the operation category [p 659].
- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the dataset node, that can be a group node or a table node.
- `{encodedPrimaryKey}` corresponds to the encoded representation of the primary key.

  See also *`RESTEncodingHelper`*API

- `{pathInRecord}` corresponds to the path starting from the table node.

Parameters

The following parameters are applicable to the count operation.

Parameter	Description
`count`	This parameter is used to specify whether this is a count operation or a selection operation. `Boolean` type, default value is `false`.

Table parameters

The following parameters are applicable to tables, associations and selection nodes.

Parameter	Description
`filter`	XPath predicate [p 227] expression defines the field values to which the request is applied. If empty, all records will be considered. `String` type value. **Note** The history code operation value is usable with the `ebx-operationCode` path field from the `meta` section associated with this field.
`historyMode`	Specifies the filter context applied on table. `String` type, possible values are: • `CurrentDataSpaceOnly`: history in current dataspace • `CurrentDataSpaceAndAncestors`: history in current dataspace and ancestors • `CurrentDataSpaceAndMergedChildren`: history in current dataspace and merged children • `AllDataSpaces`: history in all dataspaces The default value is `CurrentDataSpaceOnly`. **See also**`history` [p 659] **Note** This parameter is ignored with the `data` category.
`includeOcculting`	Includes the records in occulting mode. `Boolean` type, default value is `false`.
`viewPublication`	Specifies the name of the published view to be considered during the count execution. This parameter can be combined with: • the filter [p 688] parameter as the logical `and` operation. The behavior of this parameter is described in the section EBX as a Web Component [p 196].

Selector parameters

The following parameters are only applicable to fields that return an enumeration, foreign key or `osd:resource`.

Parameter	Description
`selector`	Specifies whether: • `true`: returns the number of all possible values • `false`: returns the number of possible values for the current field. `Boolean` type, default value is `false`. **Note** This parameter is ignored with the `history` category.
`selectorFilter`	Specifies the filter of the selector. `String` type value, the syntax complies with the Text search [p 114].

HTTP codes

HTTP code	Description
`200` *(OK)*	The selected resource is successfully counted.
`400` *(Bad request)*	The request is incorrect. This occurs when: • The selected field in a record or a dataset is sub-terminal. • The selected dataset field is a dataset tree. • The XPath predicate of the `filter` parameter is malformed or contains unfilterable nodes. • The table view for the `viewPublication` parameter is either hierarchical, non-existent or non-published. • The `selector` parameter is used for a non-enumerated node, or the `firstElementIndex` is negative, higher than or equal to the number of values.
`403` *(Forbidden)*	The selected resource is hidden for the authenticated user.
`404` *(Not found)*	The selected resource is not found.

Optimistic locking

To prevent an update or a delete operation on a record that was previously read but may have changed in the meantime, an optimistic locking mechanism is provided.

To enable optimistic locking, a select request must set the parameter `includeTechnicals` to `true`.

See Technical data [p 716] for more information.

The value of the `lastUpdateDate` property must be included in the following update request. If the record has been changed since the specified time, the update or delete will be cancelled.

- **Record**: update whole or partial content of the selected record.

  The property `lastUpdateDate` should be added to the request body to prevent update of a modified record.

  See the JSON [p 695] example of a record.
- **Field**: update of a single field of the selected record.

  The property value `lastUpdateDate` must be declared in the request URL by the `checkNotChangedSinceLastUpdateDate` parameter to prevent the update of a modified record.

The property value `lastUpdateDate` can also be used in the request URL `checkNotChangedSinceLastUpdateDate` parameter to prevent deletion on a modified record.

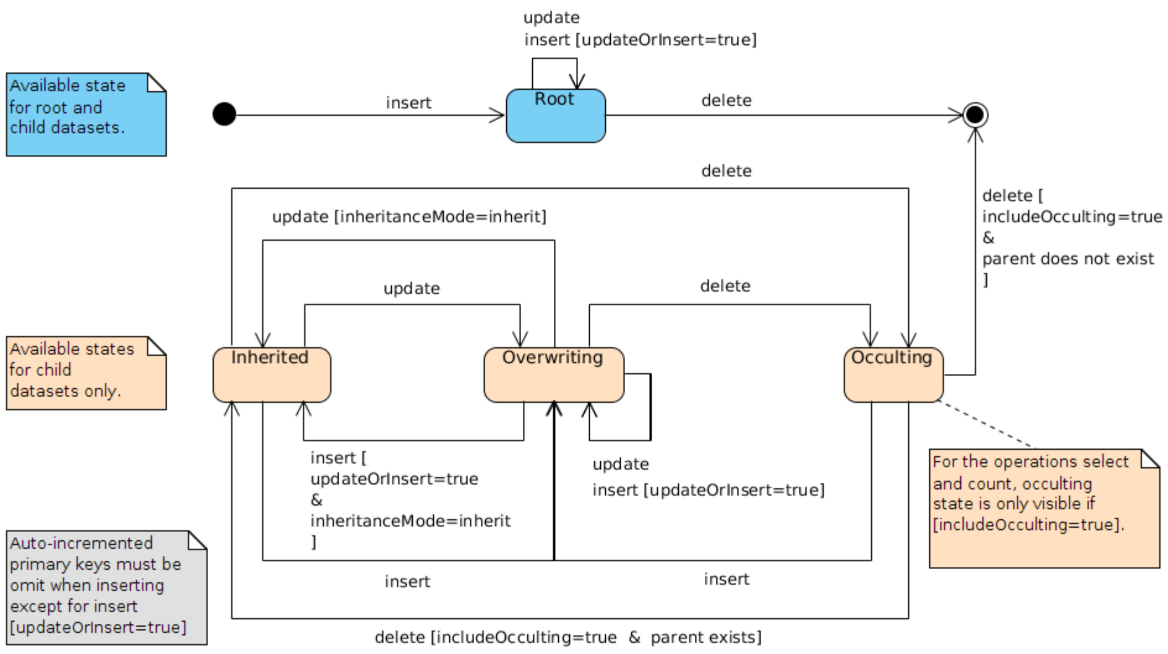
> **Note**
>
> The `checkNotChangedSinceLastUpdateDate` parameter may be used more than once but only on the same record. This implies that if the request URL returns more than one record, the request will fail.

Inheritance

EBX inheritance features are supported by built-in RESTful services using specific properties and automatic behaviors. In most cases, the inheritance state will be automatically computed by the server according to the record and field definition or content. Every action that modifies a record or a field may have an indirect impact on those states. In order to fully handle the inheritance life cycle, direct modifications of the state are allowed under certain conditions. Forbidden or incoherent explicit alteration attempts are ignored.

See also*Inheritance and value resolution [p 270]*

Record inheritance life cycle in built-in RESTful services


Available state for root and child datasets.
update
insert [updateOrInsert=true]
insert
Root
delete
delete
delete [includeOccultling=true & parent does not exist]
update [inheritanceMode=inherit]
update
delete
Available states for child datasets only.
Inherited
Overwriting
Occulting
insert [updateOrInsert=true & inheritanceMode=inherit]
update
insert [updateOrInsert=true]
For the operations select and count, occulting state is only visible if [includeOcculting=true].
Auto-incremented primary keys must be omit when inserting except for insert [updateOrInsert=true]
insert
insert
delete [includeOcculting=true & parent exists]

Inheritance properties

The following table describes properties related to the EBX inheritance features.

Property	Location	Description
`inheritance`	record or table metadata	Specifies if dataset inheritance is activated for the table. The value is computed from the data model and cannot be modified through built-in RESTful services. **See also** *`inheritance` property in metadata.* [p 705]
`inheritedField`	field metadata	Specifies the field's value source. The source data are directly taken from the data model and cannot be modified through built-in RESTful services. **See also** *`inheritedField` property in metadata.* [p 706]
`inheritanceMode`	record in child dataset	Specifies the record's inheritance state. To set a record's inheritance from `overwrite` to `inherit`, its `inheritanceMode` value must be explicitly provided in the request. In this specific case, the `content` property will be ignored if present. `occult` and `root` explicit values are always ignored. An `overwrite` explicit value without a `content` property is ignored. **Note** Inherited record's fields are necessarily `inherit`. **Note** Root records in child dataset will always be `root`. Possible values are: `root`, `inherit`, `overwrite`, `occult`. For more information, see Record lookup mechanism [p 272].
	field in `overwrite` record	Specifies the field's inheritance state. To set a field's inheritance to `inherit`, its `inheritanceMode` value must be explicitly provided in the request. The `content` property will be ignored in this case. `overwrite` explicit value without a `content` property is ignored. **Note** `inheritanceMode` at field level does not appear for `root`, `inherit` and `occult` records. **Note** `inheritedFieldMode` and `inheritanceMode` properties cannot be both set on the same field. Possible values are: `inherit`, `overwrite`. For more information, see Inheritance and value resolution [p 270].
`inheritedFieldMode`	inherited field	Specifies the inherited field's inheritance state. To set a field's inheritance to `inherit`, its `inheritedFieldMode` value must be

Property	Location	Description
		explicitly provided in the request. The `content` property will be ignored in this case. `overwrite` explicit values without a `content` property are ignored. **Note** `inheritedFieldMode` and `inheritanceMode` properties cannot be both set on the same field. **Note** `inheritedFieldMode` has priority over the `inheritanceMode` property. Possible values are: `inherit`, `overwrite`. For more information, see Value lookup mechanism [p 273].

99.7 Limitations

General limitations

- Indexes, in the request URL `{pathInDataset}` or `{pathInRecord}`, are not supported.
- Nested aggregated lists are not supported.
- Dataset nodes and field operations applied to nodes that are sub-terminal are not supported.

  See Access properties [p 537] for more information about terminal nodes.

Read operations

- Within the `selector`, the pagination context is limited to the `nextPage` property.
- Within the `viewPublication` parameter, the hierarchical view is not supported.
- The `sortOnLabel` parameter ignores programmatic labels.
- The system information response's properties cannot be browsed through the REST URL with the hierarchical representation.

  See System information operation [p 665] for more information.

Write operations

- Association fields cannot be updated, therefore, the list of associated records cannot be modified directly.
- Control policy `onUserSubmit-checkModifiedValues` of the user interface is not supported. To retrieve validation errors, invoke the select operation on the resource by including the `includeValidation` parameter.

  See Blocking and non-blocking constraints [p 522] for more information.

Directory operations

- Changing or resetting a password for a user is not supported.

CHAPTER 100
JSON format

This chapter contains the following topics:

100.1 Introduction

The JSON (JavaScript Object Notation) is the data-interchange format used by TIBCO EBX RESTful operations [p 657].

This format is lightweight, self-describing and can be used to design UIs or to integrate EBX in a company's information system.

- The data context is exhaustive, except for association fields and selection nodes that are not directly returned. However, these fields are included in the response with a URL link named `details` included by default. It can be indirectly used to get the fields content.
- The volume of data is limited by a pagination mechanism activated by default, it can be configured or disabled.

URL formatted links allow retrieving:

- Tables, records, dataset non-terminal nodes, foreign keys, resource fields (property `details`).
- Possible values for foreign keys or enumerations (`selector` parameter).

See also*Activation and configuration [p 650]*

Note

JSON data are always encoded in UTF-8.

100.2 **Global structure**

JSON Request body

Request body is represented by a JSON `Object` whose content varies according to the operation and its category.

Auth category

The request body holds several properties directly placed in the root JSON `Object`.

- **Token creation**

  Specifies the `login` and `password` to use for an authentication token creation attempt.

```
{
  "login": "...",                  // JSON String
  "password": "..."                // JSON String
}
```

  Specifies the `specific` attribute, to activate the user authentication against the HTTP request, for an authentication token creation attempt.

```
{
  "specific": true                // JSON Boolean
}
```

  See also*Create token operation* *[p 666]*

- **Password change**

  Specifies the `login`, `password` and `passwordNew` to use for the password change.

```
{
  "login": "...",                  // JSON String
  "password": "...",               // JSON String
  "passwordNew": "..."             // JSON String
}
```

  See also*Change password operation* *[p 668]*

Data category

The request body contains at least a `content` property where master data values will be defined.

- **Dataset node**

  Specifies the target values of terminal nodes under the specified node. This request is used on the dataset node update operation.

```
{
   "content": {
      "nodeName1": {
         "content": true
      },
      "nodeName2": {
         "content": 2
      },
      "nodeName3": {
         "content": "Hello"
      }
   }
}
```

See also*Update operation* *[p 682]*

- **Record**

  Specifies the target record content by setting the value for each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is used on table record insert or on record update.

  See also*Inheritance* *[p 690]*

  Some technical data can be added beside the `content` property such as `lastUpdateDate`.

  See also*Optimistic locking* *[p 689]*

```
{
   ...
   "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": "Frdric",
         "inheritedFieldMode": "overwrite"
      },
      "birthDate": {
         "content": "1810-03-01"
      },
      "deathDate": {
         "content": "1849-10-17"
      },
      "jobs": {
         "content": [
            {
               "content": "CM"
            },
            {
               "content": "PI"
            }
         ]
      },
      "infos": {
         "content": [
            {
               "content": "https://en.wikipedia.org/wiki/Chopin"
            }
         ]
      }
   }
}
```

  See also

  Insert operation *[p 677]*

  Update operation *[p 682]*

- **Record fields**

Specifies the target values of fields under the record terminal node by setting the value of each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is only used for table record updates.

See also*Inheritance* [p 690]

```
{
   "content": [
      {
         "content": "CM"
      },
      {
         "content": "PI"
      }
   ]
}
```

See also*Update operation* [p 682]

- **Record table**

  Defines the content of one or more records by indicating the value of each field. For missing fields, the behavior depends on the parameter of the request `byDelta`. This structure is used upon insert or update of records in the table.

```
{
   "rows": [
      {
         "content": {
            "gender": {
               "content": "M"
            },
            "lastName": {
               "content": "Saint-Sans"
            },
            "firstName": {
               "content": "Camille"
            },
            "birthDate": {
               "content": "1835-10-09"
            },
            ...
         }
      },
      {
         "content": {
            "gender": {
               "content": "M"
            },
            "lastName": {
               "content": "Debussy"
            },
            "firstName": {
               "content": "Claude"
            },
            "birthDate": {
               "content": "1862-10-22"
            },
            ...
         }
      }
   ]
}
```

See also

Insert operation [p 677]

Update operation [p 682]

- **Record table to be deleted**

Defines one or more records. This structure is used upon deleting several records from the same table.

```
{
   "rows": [
      {
         "details": "http://.../root/table/1"
      },
      {
         "details": "http://.../root/table/2"
      },
      {
         "primaryKey": "./oid=3"
      },
      {
         "foreignKey": "4"
      },
      ...
   ]
}
```

See also*Delete operation* *[p 684]*

- **Field**

  Specifies the target field content. This request is used on the field update.

  The request has the same structure as defined in node value [p 709] by only keeping the `content` entry. Other entries are simply ignored.

  See also*Update operation* *[p 682]*

- **Open or close user interface**

  Specifies whether the user interface is open or close and the unavailability message.

```
{
    "content": {
        "toolStatus": {
            "content": true // or false
        },
        "toolStatusCloseMessage": {
            "content": "Access is temporarily forbidden for maintenance."
        }
    }
}
```

  See also*User interface operations* *[p 665]*

Only writable fields can be mentioned in the request, this excludes the following cases:

- Association node,
- Selection node,
- Value function,
- JavaBean field that does not have a setter,
- Unwritable permission on node for authenticated user.

JSON Response body

The response body is represented by a JSON `Object` whose content depends on the operation and its category.

Admin category

The selection operation for this category only provides the values corresponding to the request under a `content` property.

- **System information**

  Contains EBX instance's system information. The representation of these data can be flat or hierarchical.

  Flat representation:

```
{
 "content": {
  "bootInfoEBX": {
   "label": "EBX® configuration",
   "content": {
    "product.version": {
     "label": "EBX® product version",
     "content": "5.8.1 [...] Enterprise Edition"
    },
    "product.configuration.file": {
     "label": "EBX® main configuration file",
     "content": "System property [ebx.properties=./ebx.properties]"
    }
    // others keys
   }
  },
  "repositoryInfo": {
   "label": "Repository information",
   "content": {
    "repository.identity": {
     "label": "Repository identity",
     "content": "00905A5753FD"
    },
    "repository.label": {
     "label": "Repository label",
     "content": "My repository"
    }
    // others keys
   }
  },
  "bootInfoVM": {
   "label": "System information",
   "content": {
    "java.home": {
     "label": "Java installation directory",
     "content": "C:\\JTools\\jdk1.8.0\\jre"
    },
    "java.vendor": {
     "label": "Java vendor",
     "content": "Oracle Corporation"
    }
    // others keys
   }
  }
 }
}
```

  Hierarchical representation:

```
{
 "content": {
  "bootInfoEBX": {
      "label": "EBX® configuration",
   "content": {
    "product": {
     "content": {
      "version": {
       "label": "EBX® product version",
       "content": "5.8.1 [...] Enterprise Edition"
      },
      "configuration": {
       "content": {
        "file": {
         "label": "EBX® main configuration file",
         "content": "System property [ebx.properties=./ebx.properties]"
        }
       }
      }
     }
    },
    "vm": {
     "content": {
      "startTime": {
       "label": "VM start time",
       "content": "2017/09/11-10:04:17-0729 CEST"
      },
      "identifier": {
       "label": "VM identifier",
       "content": "1"
      }
     }
    }
    // other hierarchical keys
   }
  }
 }
}
```

See also*System information operation* *[p 665]*

Auth category

The response body contains several properties directly placed in its root JSON `object`.

- **Token creation**

  Contains the token value and its type.

```
{
  "accessToken": "...",            // JSON String
  "tokenType": "..."               // JSON String
}
```

See also*Create token operation* *[p 666]*

Data category

The selection operation contains two different parts.

The first one named `meta` contains the exhaustive structure of the response.

The other, regrouping `content`, `rows`, `pagination`...etc, contains the values corresponding to the request.

- **Dataset tree**

  Contains the hierarchy of `table` and non-terminal `group` nodes.

```
{
   "meta": {
      "fields": [
         {
            "name": "rootName",
            "label": "Localized label",
            "description": "Localized description",
            "type": "group",
            "pathInDataset": "/rootName",
            "fields": [
               {
                  "name": "settings",
                  "label": "Settings",
                  "type": "group",
                  "pathInDataset": "/rootName/settings",
                  "fields": [
                     {
                        "name": "settingA",
                        "label": "A settings label",
                        "type": "group",
                        "pathInDataset": "/rootName/settings/settingA"
                     },
                     {
                        "name": "settingB",
                        "label": "B settings label",
                        "type": "group",
                        "pathInDataset": "/rootName/settings/settingB"
                     }
                  ]
               },
               {
                  "name": "table1",
                  "label": "Table1 localized label",
                  "type": "table",
                  "minOccurs": 0,
                  "maxOccurs": "unbounded",
                  "pathInDataset": "/rootName/table1"
               },
               {
                  "name": "table2",
                  "label": "Table2 localized label",
                  "type": "table",
                  "minOccurs": 0,
                  "maxOccurs": "unbounded",
                  "pathInDataset": "/rootName/table2"
               }
            ]
         }
      ]
   },
   "validation": [
      {
         "level": "error",
         "message": "Value must be greater than or equal to 0.",
         "details": "http://.../rootName/settings/settingA/settingA1?includeValidation=true"
      },
      {
         "level": "error",
         "message": "Field 'Settings A2' is mandatory.",
         "details": "http://.../rootName/settings/settingA/settingA2?includeValidation=true"
      }
   ],
   "content": {
      "rootName": {
         "details": "http://.../rootName",
         "content": {
            "settings": {
               "details": "http://.../rootName/settings",
               "content": {
                  "weekTimeSheet": {
                     "details": "http://.../rootName/settings/settingA"
                  },
                  "vacationRequest": {
                     "details": "http://.../rootName/settings/settingB"
                  }
               }
            },
            "table1": {
               "details": "http://.../rootName/table1"
            },
            "table2": {
               "details": "http://.../rootName/table2"
            }
         }
      }
   }
}
```

The `meta` and `validation` properties are optional.

See also

Meta-data *[p 704]*

Validation *[p 708]*

Select operation *[p 669]*

- **Dataset node**

  Contains the list of terminal nodes under the specified node.

```
{
   "meta": {
      "fields": [
         {
            "name": "nodeName1",
            "label": "Localized label of the field node 1",
            "description": "Localized description",
            "type": "boolean",
            "minOccurs": 1,
            "maxOccurs": 1,
            "pathInDataset": "/rootName/.../nodeName1"
         },
         {
            "name": "nodeName2",
            "label": "Localized label of the field node 2",
            "type": "int",
            "minOccurs": 1,
            "maxOccurs": 1,
            "pathInDataset": "/rootName/.../nodeName2"
         }
      ]
   },
   "content": {
      "nodeName1": {
         "content": true
      },
      "nodeName2": {
         "content": -5,
         "validation": [
            {
               "level": "error",
               "message": "Value must be greater than or equal to 0."
            }
         ]
      }
   }
}
```

See also*Select operation* *[p 669]*

- **Table**

  JSON `Object` containing the properties:

  - (Optional) The table meta data [p 704],
  - (Optional) The sort criteria applied,
  - (Optional) The table validation report,
  - `rows` containing a table of selected records. Each record is represented by an object, if no record is selected then the table is empty.
  - (Optional) `pagination` containing pagination [p 713] information if activated.

```
{
   "rows": [
      {
         "label": "Claude Levi-Strauss",
         "details": "http://.../root/individu/1",
         "content": {
            "id": {
               "content": 1
            },
         ...
         }
      },
      {
         "label": "Sigmoud Freud",
         "details": "http://.../root/individu/5",
         "content": {
            "id": {
               "content": 2
            },
         ...
         }
      },
   ...
      {
         "label": "Alfred Dreyfus",
         "details": "http://.../root/individu/10",
         "content": {
            "id": {
               "content": 30
            },
   ...
         }
      }
   ],
   "sortCriteria": [
      {
         "path": "/name",
         "order": "lasc"
      },
      ...
   ],
   "pagination": {
      "firstPage": null,
      "previousPage": null,
      "nextPage": "http://.../root/individu?pageRecordFilter=./id=9&pageSize=9&pageAction=next",
      "lastPage": "http://.../root/individu?pageSize=9&pageAction=last"
   }
}
```

See also*Select operation* *[p 669]*

- **Record**

  JSON `object` containing:

  - The label,
  - (Optional) The record URL,
  - (Optional) The technical data [p 716],
  - (Optional) The table metadata [p 704],
  - (Optional) The record validation report,
  - (Optional) The inheritance mode of the record is: `root`, `inherit`, `overwrite` or `occult`. This value is available for a child dataset,

    See also

    Record lookup mechanism *[p 272]*

    Inheritance *[p 690]*

  - The record content.

```
{
   "label": "Name1",
   "details": "http://.../rootName/table1/pk1",
   "creationDate": "2015-02-02T19:00:53.142",
   "creationUser": "admin",
   "lastUpdateDate": "2015-09-01T17:22:24.684",
   "lastUpdateUser": "admin",
   "inheritanceMode": "root",
   "meta": {
      "name": "table1",
      "label": "Table1 localized label",
      "type": "table",
      "minOccurs": 0,
      "maxOccurs": "unbounded",
      "primaryKeys": [
         "/pk"
      ],
      "inheritance": "true",
      "fields": [
         {
            "name": "pk",
            "label": "Identifier",
            "type": "string",
            "minOccurs": 1,
            "maxOccurs": 1,
            "pathInRecord": "pk",
            "filterable": true,
            "sortable": true
         },
         {
            "name": "name",
            "label": "Name",
            "type": "string",
            "minOccurs": 1,
            "maxOccurs": 1,
            "pathInRecord": "name",
            "filterable": true,
            "sortable": true
         },
         {
            "name": "name-fr",
            "label": "Nom",
            "type": "string",
            "minOccurs": 1,
            "maxOccurs": 1,
            "inheritedField": {
                "sourceNode": "./name"
            },
            "pathInRecord": "name-fr",
            "filterable": true,
            "sortable": true
         },
         {
            "name": "parent",
            "label": "Parent",
            "description": "Localized description.",
            "type": "foreignKey",
            "minOccurs": 1,
            "maxOccurs": 1,
            "foreignKey": {
               "tablePath": "/rootName/table1",
               "details": "http://.../rootName/table1"
            },
            "enumeration": "foreignKey",
            "pathInRecord": "parent",
            "filterable": true,
            "sortable": true
         }
      ]
   },
   "content": {
      "pk": {
         "content": "pk1"
      },
      "name": {
         "content": "Name1"
      },
      "name-fr": {
         "content": "Name1",
         "inheritedFieldMode": "inherit"
      },
      "parent": {
         "content": null,
         "validation":
         [
            {
               "level": "error",
               "message": "Field 'Parent' is mandatory."
            }
         ]
      }
   },
   "validation": {
      ...
   }
}
```

See also*Select operation* *[p 669]*

- **Fields**

  For association or selection nodes, contains the target table with associated records if, and only if, the `includeDetails` parameter is set to `true`.

  For other kinds of nodes, contains the current node value [p 709], by adding `meta` entry if enabled.

  See also*Select operation* *[p 669]*

- **Retrieve the user interface state**

  Contains the user interface status and the unavailability message.

```
{
    "content": {
        "toolStatus": {
            "content": true,
            "label": "Open",
            "selector": "http://.../domain/toolStatus/toolStatus?selector=true",
        },
        "toolStatusCloseMessage": {
            "content": "Access is temporarily forbidden for maintenance.",
        }
    }
}
```

See also*User interface operations* *[p 665]*

Note

Node, records and field in `meta`, `rows` and `content` may be hidden depending on their resolved permissions *(see permissions [p 275])*.

100.3 Meta-data

This section can be activated on demand with the `includeMeta` [p 672] parameter. It describes the structure and the JSON typing of the `content` section.

This section is deactivated by default for selection operations.

See also*Select operation* *[p 669]*

Structure of table

Table meta-data is represented by a JSON `object` with the following properties:

JSON property	JSON format	Description	Required
`name`	`String`	Name of the table defined in schema.	Yes
`label`	`String`	Table label. If undefined, the name of the schema node is returned.	Yes
`description`	`String`	Table description.	No
`type`	`String`	Always equal to: `table`.	Yes
`minOccurs`	`Number`	Number of minimum authorized record(s).	Yes
`maxOccurs`	`Number` or `String`	Number of maximum authorized record(s) or `unbounded`.	Yes
`history`	`Boolean`	Specifies if the table content is historized. Its value is `true` if history is activated, `false` otherwise. **See also***History* *[p 251]*	No
`primaryKeyFields`	`Array`	Array of the paths corresponding to the primary key.	Yes
`inheritance`	`Boolean`	Specifies whether the dataset inheritance is activated for the table. Its value is `true` if inheritance is activated, `false` otherwise. **See also***Inheritance and value resolution* *[p 270]*	No
`fields`	`Array`	Array of fields, that are direct children of the record node. Each field may also recursively contain sub-fields.	Yes

Structure of field

Each authorized field is represented by a JSON `object` with the following properties:

JSON property	JSON format	Description	Required
`name`	`String`	Name of the current authorized schema node.	Yes
`label`	`String`	Node label. If undefined, the name of the schema node is returned.	Yes
`description`	`String`	Node description.	No
`type`	`String`	Node type: simple type [p 711], `group`, `table`, `foreignKey`, etc.	Yes
`minOccurs`	`Number`	Number of minimum authorized occurrence(s).	Yes
`maxOccurs`	`Number` or `String`	Number of maximum authorized occurrence(s) or `unbounded`.	Yes

`inheritedField`	`Object`	Holds information related to the inherited field's value source.
		`"inheritedField": {`
		`    "sourceRecord": "/path/to/record", // (optional)`
		`    "sourceNode": "./path/to/Node"`
		`}`
		See also*Inheritance and value resolution* [p 270]
			No

`foreignKey`	`Object`	Contains information related to the target table.
		`{`
		`   "dataspace": "BAuthors",`
		`   "dataset": "Authors",`
		`   "tablePath": "/root/Authors",`
		`   "details": "http://.../BAuthors/Authors/root/`
		`Authors"`
		`}`
			No (*)

`dataspace`	`String`	Target dataspace or snapshot identifier. This property is placed under the `foreignKey` property.	No (*)
`dataset`	`String`	Target dataset identifier. This property is placed under the `foreignKey` property.	No (*)
`tablePath`	`String`	Target table path. This property is placed under the `foreignKey` property.	Yes
`details`	`String`	Target table URL. This property is placed under the `foreignKey` property and is included by default.	No
`enumeration`	`String`	Specifies if the field is an enumeration value. Possible values are: • `foreignKey`	No

JSON property	JSON format	Description	Required
		• `static` • `dynamic` • `programmatic` • `nomenclature` • `resource` **See also** *`SchemaFacetEnumeration`* API Specifies whether the field can be used for retrieving possible values by using the `selector` request parameter.	
`valueFunction`	`Boolean`	Specifies if the field is a computed value. **See also** *Computed values* [p 527]	No
`pathInDataset`	`String`	Relative field path starting from the schema node.	No (**)
`pathInRecord`	`String`	Relative field path starting from the table node.	No (*)
`filterable`	`Boolean`	Specifies whether the field can be used for filtering record using `filter` request parameter.	No (*)
`sortable`	`Boolean`	Specifies whether the field can be used in sort criteria using `sort` request parameter.	No (*)
`fields`	`Array` of `Object` elements	Contains the structure and typing of each group field.	No

(*) Only available for table, record and record field operations.

(**) Only available for dataset tree operations.

100.4 **Sort criteria information**

The sort criteria applied to the request can be returned on demand, by using the `includeSortCriteria` [p 673] parameter (deactivated by default). If it is activated, a `sortCriteria` property is directly added to the response root node.

A `sortCriteria` property contains a JSON `Array` that contains ordered sort criteria, and for each sort criterion, a JSON `Object` is added with the following properties:

JSON property	JSON format	Description	Required
`path`	`String`	Field path.	Yes
`order`	`String`	Possible values are: `asc`, `lasc`, `desc` or `ldesc`.	Yes

100.5 Validation

The validation can be activated on demand with the `includeValidation` [p 673] parameter (deactivated by default). If it is activated, `validation` properties are directly added on target nodes with one or several messages. For messages without a target node path, a `validation` property is added on the root node.

A `validation` property contains a JSON `Array` and for each message, corresponding to a validation item, a JSON `Object` with properties:

JSON property	JSON format	Description	Required
`level`	`String`	Severity of the validation item, the possible values are: `info`, `warn`, `error`.	Yes
`message`	`String`	Description of the validation item.	Yes
`details`	`String` corresponding to an absolute URL.	URL of the resource associated with the validation item. Only available on the table and dataset scopes, if associated resources exist and if it is included. **See also** *`includeDetails` parameter* [p 672]	No

100.6 Content

This section can be deactivated on demand with the `includeContent` [p 672] parameter (activated by default). It provides the content of the record values, dataset, or field of one of the content fields for an authorized user. It also has additional information, including labels, technical information, URLs...

The content is represented by a JSON `Object` with a property set for each sub-node.

Node value

JSON property	JSON format	Description	Required
`content`	Content of simple type [p 711] Content of group and list [p 713]	Contains the node value. Available for all nodes except `association` and `selection`. However, their content can be retrieved by invoking the URL provided in `details`.	`No`
`details`	`String` corresponding to an absolute URL.	By invocation, the node details are returned. Response type after invocation depending on the meta type. • `foreignKey`: target record (available on table, record and field operation). • `resource`: target resource [p 518] (available on dataset node, table, record and field operations). • `association`: target table containing associated records (available on table and record operations). • `selection`: target table containing associated records (available on table and record operations). • `group`: target dataset group node (available on dataset tree operation). • `table`: target table (available on dataset tree operation). Example: `http://.../BReference/dataset/root/table/pk/associationField`	`No`
`label`	`String`	Contains the foreign key or enumeration label in the current locale. The default label is returned if the current locale is not supported.	`No`
`inheritanceMode`	`String`	Contains the node's inheritance state, considering only dataset inheritance. `inheritedFieldMode` and `inheritanceMode` properties cannot be both defined on the same node. **See also** *`inheritedFieldMode`* *[p 709]* *Record lookup mechanism* *[p 272]*	`No`
`inheritedFieldMode`	`String`	Contains the node's field inheritance state, considering dataset and field inheritance. When both inheritances are used, field inheritance has priority over the dataset one. `inheritedFieldMode` and `inheritanceMode` properties cannot be both defined on the same node. **See also** *`inheritanceMode`* *[p 709]* *Value lookup mechanism* *[p 273]*	`No`
`selector`	`String`	Correspond to the URL for selector [p 714] operation.	`No`

JSON property	JSON format	Description	Required
	corresponding to an absolute URL.	Example: `http://.../BReference/dataset/root/table/pk/enumField?selector=true`	
`validation`	`Array`	Contains the validation report that concerns the current node context.	`No`

Content of simple type

A simple field value is stored in a JSON object and the content is the value of the `content` property.

XML Schema	JSON format	Examples	Meta type

`xs:string`
`xs:Name`
`osd:html`
`osd:email`
`osd:text`
	`String` (Unicode characters, cf. RFC4627)	"A text"
		"The escape of \"special character\" is preceded by a backslash."
		"<p>An HTML tag can thus be written without trouble</p>"
		"employee@mycompany.com"
		null
			`string`
			`name`
			`html`
			`email`
			`text`

`osd:locale`	`String` (Language tag, cf. RFC1766)	"en-US"	`locale`
`xs:string` (Foreign key)	`String` contains the value of the formatted foreign key.	`"0"` `"true|99"`	`foreignKey`
`xs:boolean`	`Boolean`	true false null	`boolean`
`xs:decimal`	`Number` or `null`	-10.5 20.001 15 -1e-13	`decimal`
`xs:date`	`String` with format: "yyyy-MM-dd"	"2015-04-13"	`date`
`xs:time`	`String` with format: • "HH:mm:ss" • "HH:mm:ss.SSS"	"11:55:00" "11:55:00.000"	`time`
`xs:dateTime`	`String` with format: • "yyyy-MM-ddTHH:mm:ss" • "yyyy-MM-ddTHH:mm:ss.SSS"	"2015-04-13T11:55:00" "2015-04-13T11:55:00.000"	`dateTime`
`xs:anyURI`	`String` (Uniform Resource Identifier, cf. RFC3986)	"https://fr.wikipedia.org/wiki/Ren_Descartes"	`anyURI`
`xs:int` `xs:integer`	`Number` or `null`	1596	`int`
`osd:resource`	`String`	"ebx-tutorial:ext-images:frontpages/Ajax for Dummies.jpg"	`resource`

XML Schema	JSON format	Examples	Meta type
	contains the resource formatted value.		
`osd:color`	`String` with format: "#[A-Fa-f0-9]{6}" contains the formatted value for the color.	"#F6E0E0"	`color`
`osd:datasetName`	`String` with format: "[a-zA-Z_][-a-zA-Z0-9_.]*" and 64 characters max. contains the formatted value of the dataset name.	"ebx-tutorial"	`dataset`
`osd:dataspaceKey`	`String` with format: "[BV][a-zA-Z0-9_:.\\-\\\|]*" and 33 characters max. contains the formatted key value of the dataspace.	"Bebx-tutorial"	`dataspace`

Content of group and list

XML Schema	JSON format	Examples	Meta type

Group
`xs:complexType`
	`Object`
	Contains a property per sub-node.
		Example for a simple-occurrence group.
		`{`
		`   "road" : {"content" : "11 rue scribe"},`
		`   "zipcode" : {"content" : "75009"},`
		`   "country" : {"content" : "France"}`
		`}`
			`group`

List
`maxOccurs > 1`
	`Array`
	Contains an array of all field occurrences represented by a JSON `Object`.
	Each object is represented as a node value [p 709].
		Example for a multi-occurrence field of the `xs:int` type.
		`[`
		`   {"content": 0},`
		`   {"content": 1},`
		`   {"content": 2},`
		`   {"content": 3}`
		`]`
		Example for a multi-occurrence group.
		`[`
		`   {`
		`      "content":`
		`      {`
		`         "road": {"content": "11 rue scribe"},`
		`         "zipcode": {"content": "75009"},`
		`         "country": {"content": "France"}`
		`      }`
		`   },`
		`   {`
		`      "content":`
		`      {`
		`         "road": {"content": "711 Atlantic Ave"},`
		`         "zipcode": {"content": "MA 02111"},`
		`         "country": {"content": "United States"}`
		`      }`
		`   }`
		`]`
			Meta of simple type [p 711],
			or
			`group`

Pagination

This feature allows returning a limited and parameterizable number of data. Pagination can be applied to data of the following types: records, association values, selection node values and selectors. A context named `pagination` is returned only if it has been activated. This context allows browsing data similarly to the UI.

Pagination is activated by default.

See also*Select operation* *[p 669]*

Detailed information related to this context can be found hereafter:

JSON property	JSON format	Description	Required
`firstPage`	`String` or `null` (*)	URL to access the first page.	Yes (**)
`previousPage`	`String` or `null` (*)	URL to access the previous page.	Yes (**)
`nextPage`	`String` or `null` (*)	URL to access the next page.	Yes
`lastPage`	`String` or `null` (*)	URL to access the last page.	Yes (**)

> **Note**
>
> (*) Only defines if data is available in this context and not in the response.

> **Note**
>
> (**) Not present on selector.

Selector

By invoking the URL represented by the property `selector` on a field that provides an enumeration, this returns a JSON `Object` containing the properties:

- `rows` containing an `Array` of JSON `Object` where each one contains two entries, such as the returned `content` that can be persisted and the corresponding `label`. The list of possible items is established depending on the current context.
- (Optional) `pagination` containing pagination [p 713] information (activated by default).

```
{
   "rows": [
      {
         "content": "F",
         "label": "feminine"
      },
      {
         "content": "M",
         "label": "masculine"
      }
   ],
   "pagination": {
      "nextPage": null
   }
}
```

See also*includeSelector* *[p 673]*

Insert operation report

When invoking the insert operation with a record table, it can optionally return a report. The report includes a JSON `Object` that contains the following properties:

- `rows` contains a JSON `ObjectArray`, where each element corresponds to the result of a request element.
- `code` contains an `int` of the JSON `Number` type, and allows to know whether the record has been inserted or updated. This property is included if, and only if, the `updateOrInsert` parameter is set to `true`.

- `foreignKey` contains a `string` of the JSON `String` type, corresponding to the content to be used as a foreign key for this record. This property is included if, and only if, the parameter `includeForeignKey` is set to `true`.
- `label` contains a `string` of the JSON `String` type, and allows to retrieve the record label. This property is included if, and only if, the parameter `includeLabel` is set to `yes`.
- `details` containing a `string` of the JSON `String` type, corresponding to the resource URL. This property is included if, and only if, the parameter `includeDetails` is set to `true`.

```
{
   "rows": [
      {
         "code": 204,
         "foreignKey": "62",
         "label": "Claude Debussy",
         "details": "http://.../root/individu/62"
      },
      {
         "code": 201,
         "foreignKey": "195",
         "label": "Camille Saint-Sans",
         "details": "http://.../root/individu/195"
      }
   ]
}
```

See also*Insert operation* *[p 677]*

Delete operation report

When invoking the delete operation, a report is returned. The report includes a JSON `Object` that contains the following properties:

- `deletedCount` containing an integer of the JSON `Number` type, corresponds to the number of deleted records.
- `occultedCount` containing an integer of the JSON `Number` type, corresponds to the number of occulted records.
- `inheritedCount` containing an integer of the JSON `Number` type, corresponds to the number of inherited records.

```
{
 "deletedCount": 1,
 "occultedCount": 0,
 "inheritedCount": 0
}
```

See also*Delete operation* *[p 684]*

Technical data

Each returned record is completed with the properties corresponding to its technical data, containing:

JSON property	JSON format	Description	Required
`creationDate`	`String`	Creation date.	Yes
`creationUser`	`String`	Creation user identifier.	Yes
`lastUpdateDate`	`String`	Last update date.	Yes
`lastUpdateUser`	`String`	Last update user identifier.	Yes

```
{
   ...
   "creationDate": "2015-12-24T19:00:53.158",
   "creationUser": "admin",
   "lastUpdateDate": "2015-12-25T00:00:00.001",
   "lastUpdateUser": "admin",
   ...
}
```

100.7 Update modes

The `byDelta` mode allows to ignore data model elements that are missing from the JSON source document. This mode is enabled (by default) through RESTful operations. The following table summarizes the behavior of insert and update operations when elements are not included in the source document.

See the RESTful data services operations update [p 682] and insert [p 677], as well as `ImportSpec.setByDelta`API in the Java API for more information.

State in the JSON source document	Behavior
The property does not exist in the source document	**If the `byDelta` mode is activated (default):** • For the `update` operation, the field value remains unchanged. • For the `insert` operation, the behavior is the same as when the `byDelta` mode is disabled. **If the `byDelta` mode is disabled through the RESTful operation parameter:** The target field is set to one of the following values: • If the element defines a default value, the target field is set to that default value. • If the element is of a type other than a string or list, the target field value is set to `null`. • If the element is an aggregated list, the target field value is set to an empty list value. • If the element is a string that differentiates `null` from an empty string, the target field value is set to `null`. If it is a string that does not differentiate the two, an empty string. • If the element (simple or complex) is hidden in the data services, the target value remains unchanged. **See also***Hiding a field in Data Services* [p 540] **Note** The user performing the import must have the required permissions to create or change the target field value. Otherwise, the operation will be aborted.
The element is present and its value is `null` (for example, `"content": null`)	The target field is always set to `null` except for lists, in which case it is not supported.

100.8 Known limitations

Field values

The value of fields `xs:date`, `xs:time` and `xs:dateTime` does not contain a time zone associated with the JSON-primitive type.

CHAPTER 101 REST Toolkit

This chapter contains the following topics:

101.1 Introduction

TIBCO EBX offers the possibility to develop custom REST services using the REST Toolkit. The REST Toolkit supports JAX-RS 2.1 (JSR-370) and JSON-B (JSR-367).

A REST service is implemented by a Java class and its operations are implemented by Java methods. The response can be generated by serializing POJO objects. The request input can be unserialized to POJOs. Various input and output formats, including JSON, are supported. For more details on supported formats, see media types [p 721].

Rest Toolkit supports the following:

- Injectable objects

  EBX provides injectable objects useful to authenticate the request's user, to access the EBX repository or to built URIs without worrying about the configuration (for example reverse-proxy [p 650] or REST forward [p 578] modes);

  JAX-RS injectable objects are also supported.

- Annotations

  EBX provides annotations to describe resources, grant anonymous access or add restrictions to a method.

  JAX-RS ans JSON-B annotations are also supported.

- logging and debugging utilities.

See also *JAX-RS: JavaTM API for RESTful Web Services 2.1*

101.2 Application definitions

An EBX module, that includes custom REST services, must provide at least one REST Toolkit application class. A REST Toolkit application class extends the EBX `RESTApplicationAbstract`API class. The minimum requirement is to define the base URL, using the `@ApplicationPath` annotation and the set of packages to scan for REST service classes.

> **Note**
>
> Only packages accessible from the web application's classloader can be scanned.

> **Note**
>
> It is possible to register REST resource classes or singletons, packaged inside or outside the web application archive, through the **ApplicationConfigurator.register(java.lang.Class)** `ApplicationConfigurator.register`API or **ApplicationConfigurator.register(java.lang.Object)** `ApplicationConfigurator.register`API methods.

> **Note**
>
> If no packages scope is defined, then every class reachable from the web application's classloader will be scanned.

The application path cannot be "`/`" and must not collide with an existing resource from the module. It is recommended to use "`/rest`" (the value of the `RESTApplicationAbstract.REST_DEFAULT_APPLICATION_PATH` constant).

EBX `Documentation`API annotation is optional. It is displayed to administrators in 'Technical configuration' > 'Modules and data models' or when logging and debugging.

```
import 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()));
   }
}
```

101.3 Service and operation definitions

A REST Toolkit service is implemented by a Java class and its operations are implemented by its methods.

Class and methods can be annotated by `@Path` to specify the access path. The `@Path` annotation value defined at the class level will prepend the ones defined on methods. Warning, only one `@Path` annotation is allowed per class or method.

Media types accepted and produced by a resource are respectively defined by the `@Consumes` and `@Produces` JAX-RS annotations. The supported media types are:

- `application/json` (MediaType.APPLICATION_JSON_TYPE)
- `application/octet-stream` (MediaType.APPLICATION_OCTET_STREAM_TYPE)
- `application/x-www-form-urlencoded` (MediaType.APPLICATION_FORM_URLENCODED_TYPE)
- `multipart/form-data` (MediaType.MULTIPART_FORM_DATA_TYPE)
- `text/css`
- `text/html` (MediaType.TEXT_HTML_TYPE)
- `text/plain` (MediaType.TEXT_PLAIN_TYPE)

Valid HTTP(S) methods are specified by JAX-RS annotations `@GET`, `@POST`, `@PUT`, etc. Only one of these annotations can be set on each Java method (this means that a Java method can support only one HTTP method).

Warning: URL parameters with a name prefixed with `ebx-` are reserved by REST Toolkit and should not be defined by custom REST services, unless explicitly authorized by the EBX documentation.

URL and sample

The REST URL to access the description service for the sample is defined below:

`http[s]://<host>[:<port>]/<path to webapp>/rest/track/v1/description`

Where:

- `<path to webapp>` corresponds to the web application's path holding the REST Toolkit application, itself serving the sample service. The path is composed by multiple, or none, URI segments followed by the web application's name.

> **Note**
>
> Please note that `/rest/track/v1/description` corresponds to the concatenation of the application's `@ApplicationPath` and service's `@Path` annotations.

The following REST Toolkit service sample provides methods to query and manage track data:

```
import java.net.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.regex.*;
import java.util.stream.*;

import 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()
   .parallelStream()
   .filter(Objects::nonNull)
   .filter(track -> singerPattern == null || singerPattern.matcher(track.singer).matches())
   .filter(track -> titlePattern == null || titlePattern.matcher(track.title).matches())
   .collect(Collectors.toList());
}

private static Pattern compilePattern(final String aPattern)
{
 if (aPattern == null || aPattern.isEmpty())
  return null;

 try
 {
  return Pattern.compile(aPattern);
 }
 catch (final PatternSyntaxException ignore)
 {
  // ignore invalid pattern
  return null;
 }
}

/**
 * Counts all tracks.
 */
@GET
@Path("/tracks:count")
@Documentation("Counts all tracks")
public int handleCountTracks()
{
 return TRACKS.size();
}

/**
 * Selects a track by id.
 */
@GET
@Path("/tracks/{id}")
@Documentation("Selects a track by id")
public TrackDTO handleSelectTrackById(@PathParam("id") Integer id)
{
 final TrackDTO track = TRACKS.get(id);
 if (track == null)
  throw new NotFoundException("Track id [" + id + "] does not found.");
 return track;
}

/**
 * Deletes a track by id.
 */
@DELETE
@Path("/tracks/{id}")
```

```
@Documentation("Deletes a track by id")
public void handleDeleteTrackById(@PathParam("id") Integer id)
{
 if (!TRACKS.containsKey(id))
  throw new NotFoundException("Track id [" + id + "] does not found.");
 TRACKS.remove(id);
}

/**
 * Inserts or updates one or several tracks.
 * <p>
 * The complex response structure corresponds to one of:
 * <ul>
 *  <li>An empty content with the <code>location<code> HTTP header defined
 *   to the access URI.</li>
 *  <li>A JSON array of {@link ResultDetailsDTO} objects.</li>
 * </ul>
 */
@POST
@Path("/tracks")
@Documentation("Inserts or updates one or several tracks")
public Response handleInsertOrUpdateTracks(List<TrackDTO> tracks)
{
 int inserted = 0;
 int updated = 0;

 final ResultDetailsDTO[] resultDetails = new ResultDetailsDTO[tracks.size()];
 int resultIndex = 0;

 final URI base = this.sessionContext.getURIInfoUtility()
   .createBuilderForRESTApplication()
   .path(this.getClass())
   .segment("tracks")
   .build();

 for (final TrackDTO track : tracks)
 {
  final String id = String.valueOf(track.id);
  final URI uri = UriBuilder.fromUri(base).segment(id).build();

  final int code;
  if (TRACKS.containsKey(track.id))
  {
   code = HttpServletResponse.SC_NO_CONTENT;
   updated++;
  }
  else
  {
   code = HttpServletResponse.SC_CREATED;
   inserted++;
  }

  TRACKS.put(track.id, track);

  resultDetails[resultIndex++] = ResultDetailsDTO.create(
   code,
   null,
   String.valueOf(track.id),
   uri);
 }

 if (inserted == 1 && updated == 0)
  return Response.created(resultDetails[0].details).build();

 return Response.ok().entity(resultDetails).build();
}

/**
 * Updates one track.
 */
@PUT
@Path("/tracks/{id}")
@Documentation("Update one track")
public void handleUpdateOneTrack(@PathParam("id") Integer id, TrackDTO aTrack)
{
 final TrackDTO track = TRACKS.get(id);
 if (track == null)
  throw new NotFoundException("Track id [" + id + "] does not found.");

 if (aTrack.id != null && !aTrack.id.equals(track.id))
  throw new BadRequestException("Selected track id [" + id
   + "] is not equals to body track id.");

 TRACKS.put(aTrack.id, aTrack);
}
```

```
}
```

This REST service uses the following Java classes, which represent a Data Transfer Objects (DTO), to serialize and deserialize data:

```
/**
 * DTO for a track.
 */
public final class TrackDTO
{
 public Integer id;
 public String singer;
 public String title;
}

import java.net.*;

/**
 * DTO for result details.
 */
@JsonbPropertyOrder({ "code", "label", "foreignKey", "details" })
public final class ResultDetailsDTO
{
 public int code;
 public String label;
 public String foreignKey;
 public URI details;

 public static ResultDetailsDTO create(
  final int aCode,
  final String aForeignKey,
  final URI aDetails)
 {
  return ResultDetailsDTO.create(aCode, null, aForeignKey, aDetails);
 }

 public static ResultDetailsDTO create(
  final int aCode,
  final String aLabel,
  final String aForeignKey,
  final URI aDetails)
 {
  final ResultDetailsDTO result = new ResultDetailsDTO();
  result.code = aCode;
  result.label = aLabel;
  result.foreignKey = aForeignKey;
  result.details = aDetails;
  return result;
 }
}
```

101.4 **Authentication and lookup mechanism**

A custom REST service developed with REST Toolkit supports the same authentication methods and lookup mechanism as the built-in REST data services. However, there is a slight difference concerning the 'Anonymous authentication Scheme' since its scope can be wider by using the `AnonymousAccessEnabled`API. See REST authentication and permissions [p 724] for more information.

See also

Authentication [p 652]

Lookup mechanism [p 653]

101.5 **REST authentication and permissions**

By default, every REST resource Java method requires users to be authenticated.

However, some methods may need to be accessible anonymously. These methods must be annotated by `AnonymousAccessEnabled`API.

Some methods may need to be restricted to given profiles. These methods may be annotated by `Authorization`API to specify an authorization rule. An authorization rule is a Java class that implements the `AuthorizationRule`API interface.

```
import 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()
 {
  ...
 }
}
```

101.6 **URI builders**

REST Toolkit provides an utility interface `URIInfoUtility`API to generate URIs. An instance of this interface is accessible through the injectable built-in object `SessionContext`API.

101.7 **Exception handling**

A REST Toolkit Java method can produce a standard HTTP error response by throwing a Java exception that extends the JAX-RS class `javax.ws.rs.WebApplicationException`. JAX-RS defines exceptions for various HTTP status codes. EBX defines `UnprocessableEntityException`API that adds support for the HTTP `422`*(Unprocessable entity)* code.

Plain Java exceptions are mapped to the HTTP status code `500` *(Internal server error)*.

```
{
  "userMessage": "...", // Mandatory localized message
  "errorCode":   "999", // EBX® error code (optional, used mainly for HTTP error 422)
  "errors": [           // Internal messages useful when debugging (optional).
                        // Usually not displayed to the user and not localized.
     "Message 1", "Message 2" }
  ]
}
```

101.8 **Monitoring**

REST Toolkit events monitoring is similar to the data services log configuration. The difference is the property key which must be `ebx.log4j.category.log.restServices`.

See also

Monitoring *[p 653]*

Configuring the EBX logs *[p 351]*

Some additional properties are available to configure the log messages. See Configuring REST toolkit services [p 356] for further information.

101.9 **Packaging and registration**

All applications and components are required to be packaged into the module's Web Application (war file).

The JAX-RS libraries, except the JAX-RS client API, are already included in `ebx.jar` and must not be packaged in the war file.

See Java EE deployment [p 317] for more information.

The registration of a REST Toolkit application is integrated into the EBX module registration process. The registration class must extend `ModuleRegistrationListener`API, declare the Servlet 3.0 annotation `@WebListener` and override the `handleContextInitialized` method.

See Module registration [p 460] for more information.

```
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);
 }
}
```