# EBX5

# Product Documentation

EBX5 Version 5.6.1 Fix M

EBX5 includes:
- software developped by the Apache Software Foundation - http://www.apache.org/
- software developped by the JDOM Project - http://www.jdom.org/
- software developped by the JSON Project - http://www.json.org/
- the Gagawa HTML Generator - http://code.google.com/p/gagawa/
- the Jericho HTML Parser - http://jericho.htmlparser.net/docs/index.html
- the H2 Database Engine - http://www.h2database.com/
- Font Awesome by Dave Gandy - http://fontawesome.io/

# Table of contents

## User Guide

### Introduction

### Data models

### Data spaces

### Data sets

### Workflow models

### Data workflows

# Reference Manual

## Integration

# User Guide

# Introduction

CHAPTER **1**

# How EBX5 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 EBX5, business and IT users can collaborate on a single, unified solution in order to design data models and manage master data content.

EBX5 is MDM software that allows modeling any type of master data and applying governance using the rich features included, such as collaborative workflows, data authoring, hierarchy management, version control, and role-based security.

An MDM project using EBX5 starts with the creation of a *data model*. This is where the definition of tables, fields, links and business rules related to the master data takes place. Examples of modeled data include product catalogs, financial hierarchies, supplier lists or simple reference tables.

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

*Workflows* are an invaluable feature for performing controlled change management or data approvals. 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 then be executed as *data workflows*. Data workflows are able to notify users of relevant events and outstanding work in a collaborative context.

*Data services* help integrate EBX5 with third-party systems (middleware), by allowing external systems to access data in the repository, or to manage data spaces and workflows through web services.
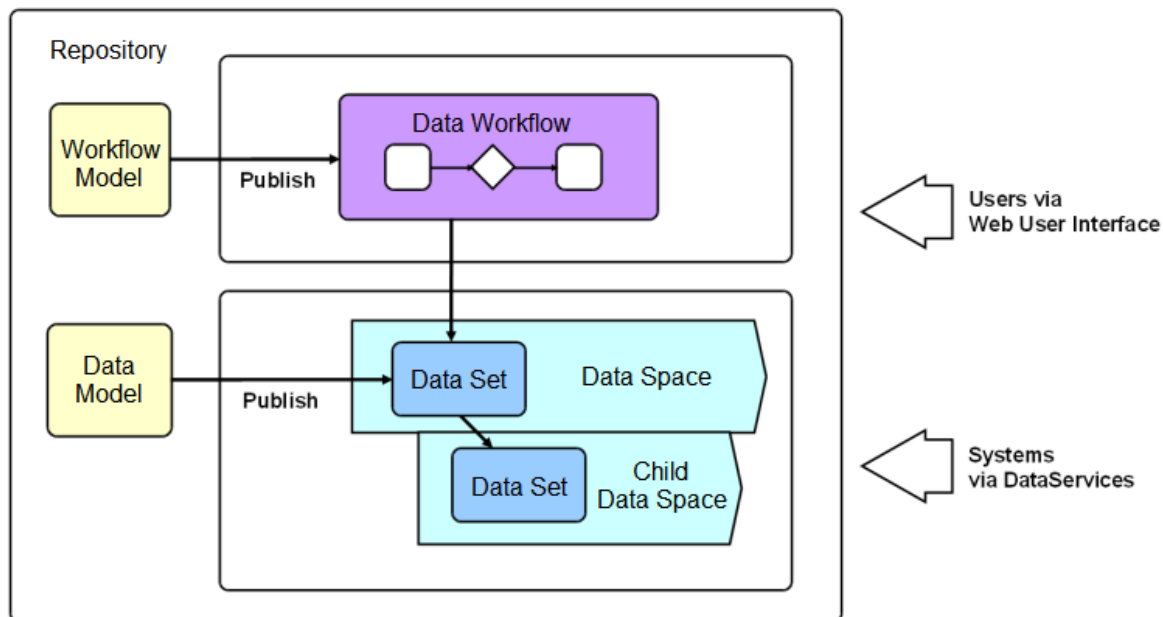


See also

Data modeling *[p 18]*

Data sets *[p 19]*

Data spaces *[p 21]*

Workflow modeling *[p 22]*

Data workflows *[p 23]*

Data services *[p 24]*

CHAPTER **2**

# Using the EBX5 user interface

This chapter contains the following topics:

## 2.1 Overview

The general layout of EBX5 workspaces is entirely customizable by a perspective administrator. If customized perspectives have been created, a drop-down menu 'Select perspective' allows switching between different perspectives.

The advanced perspective is accessible by default.

## 2.2 Advanced perspective

By default, the EBX5 advanced perspective is available to all users, but its access can be restricted to selected profiles. The view is separated into several general regions, referred to as the following in the documentation:

- **Header:** Displays the user currently logged in, the user language preference selector (when more than one available), the perspective selector (when more than one available), a link to the product documentation, and a logout button.

- **Menu bar:** The functional categories accessible to the current user.

- **Navigation pane:** Displays context-dependent navigation options. For example: selecting a table in a data set, 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, Data Spaces, Modeling, Data Workflow, Data Services,* and *Administration.*
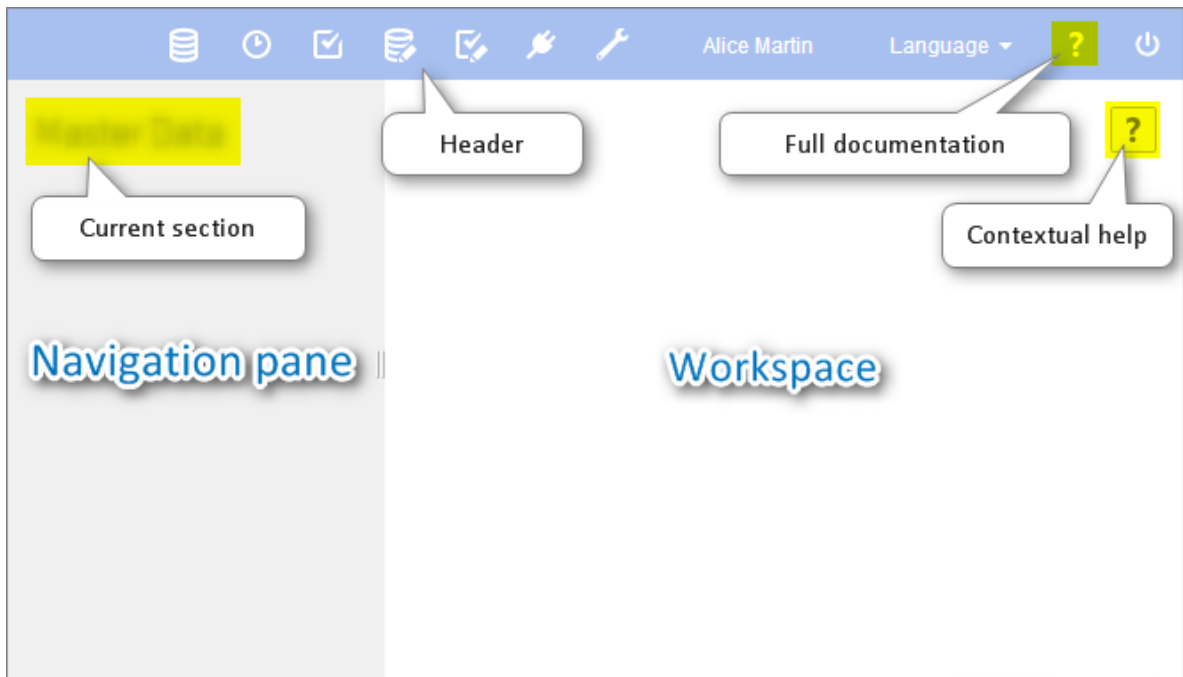


## 2.3 **Perspectives**

The EBX5 perspectives is a highly configurable view with a target audience. Perspectives offer a simplified user interface to business users and can be assigned to one or more profile. This view is separated into several general regions, referred to as the following in the documentation:

- **Header:** Displays the user currently logged in, the user language preference selector (when more than one available), the perspective selector (when more than one available), and a logout button.
- **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.

Example of a hierarchical menu:
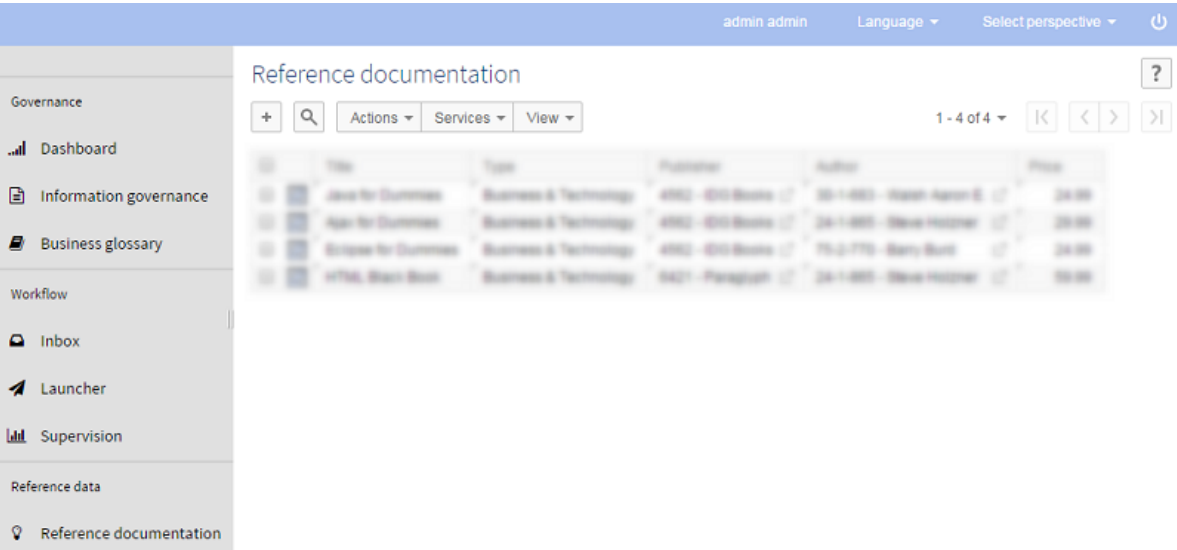


## 2.4 **User interface features**

### *Hiding the header*

It is possible to hide the header in the user interface by hovering over it, then clicking the button that appears.
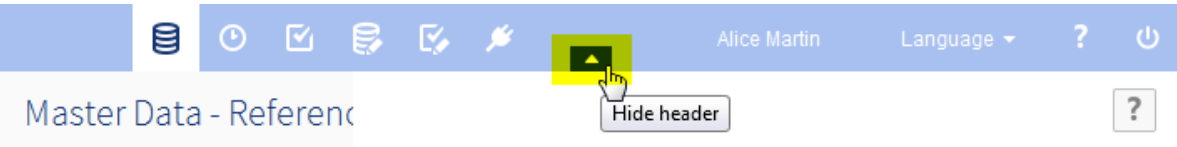
### *Resetting the navigation pane width*

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.
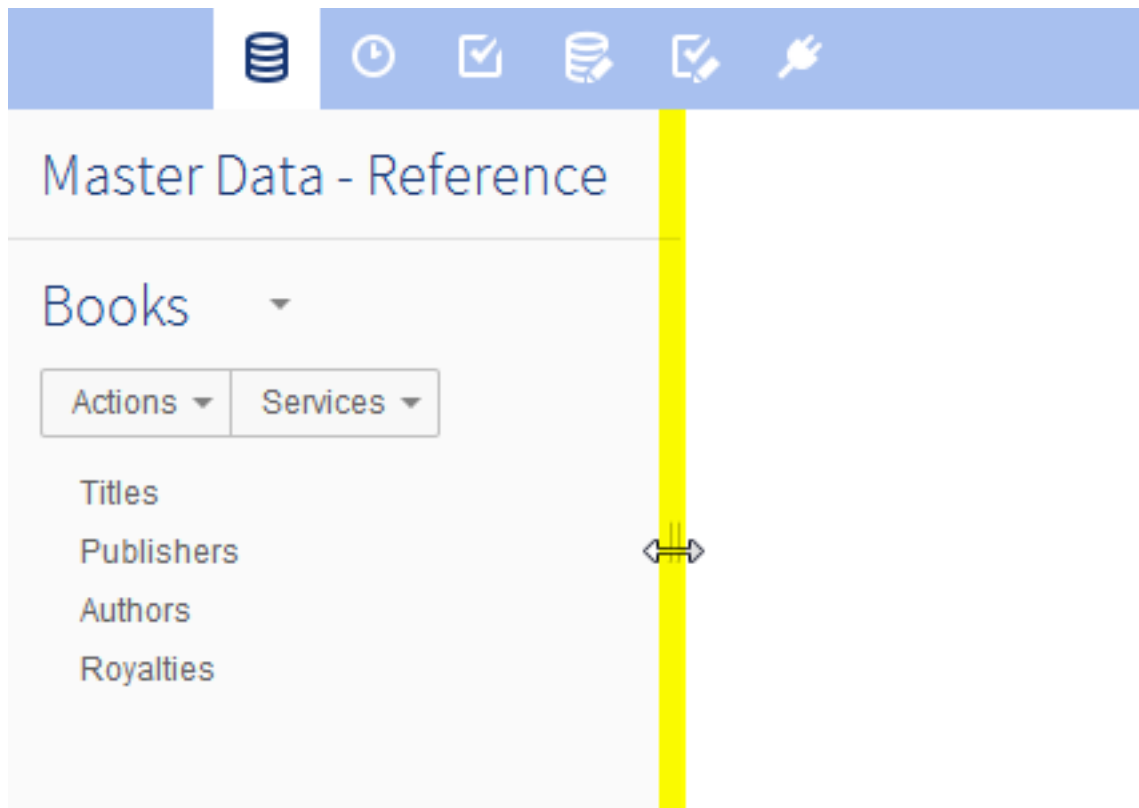


## 2.5 **Where to find EBX5 help**

In addition to the full standalone product documentation, help is accessible in various forms within the interface.

### *Contextual help*

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

When a permalink to the element is available, a link button appears in the upper right corner of the panel.
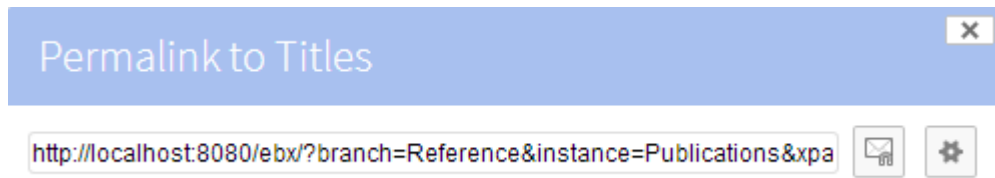
CHAPTER **3**

# Glossary

This chapter contains the following topics:

## 3.1 **Governance**

### *repository*

A back-end storage entity containing all data managed by EBX5. The repository is organized into data spaces.

See also data space [p 21].

### *profile*

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

See also user [p 17], role [p 18].

### *user*

An entity created in the repository in order for physical users or external systems to authenticate and access EBX5. Users may be assigned roles, as well as have other account information associated with them.

### 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 EBX5, 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.

## 3.2 Data modeling

*Main documentation section Data models [p 28]*

### data model

A structural definition of the data to be managed in the EBX5 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 data sets, which are instances of data models that contain the data being managed by the repository.

See also data set [p 20].

Related concept Data models [p 28].

### 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 EBX5, 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 19], group [p 19], table (in data model) [p 19], validation rule [p 19], inheritance [p 20].

Related concepts Structure elements properties [p 41], Controls on data fields [p 47].

### 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 .

### 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 18].

### 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 that is 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 19], primary key [p 18], reusable type [p 19].

### 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 19].

### 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.

### data model assistant (DMA)

The EBX5 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 28].

## 3.3 Data sets

*Main documentation section Data sets [p 78]*

### 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 that is defined in the data model. The data model drives the data types and cardinality of the fields found in records.

Records are represented by the icon .

See also table (in data set) [p 20], primary key [p 18].

### table (in data set)

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 19], primary key [p 18].

### data set

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

Data sets are represented by the icon .

See also table (in data set) [p 20], field [p 18], group [p 19], custom table view [p 20].

Related concept Data sets [p 78].

### inheritance

A mechanism by which data can be acquired by default by one entity from another entity. In EBX5, there are two types of inheritance: data set inheritance and field inheritance.

When enabled, data set inheritance allows a child data set to acquire default data values from its parent data set. 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, data set inheritance is disabled. It can be enabled during the data model definition.

Inheritance from the parent data set is represented by the icon .

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

Inherited fields are represented by the icon .

### custom table view

A customizable display configuration that may be applied for 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 set record filtering criteria.

See also hierarchical table view [p 20].

Related concept Views [p 85].

### hierarchical table view

A custom table view that offers a tree-based representation of the data in a table. Field values are displayed as nodes in the tree. A hierarchical view can be useful for showing the relationships between data. When creating a custom table view that uses the hierarchical format, dimensions can be selected to determine the structural representation of the 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 <u>custom table view</u> [p 20].

Related concept <u>Hierarchies</u> [p 86].

# 3.4 **Data management life cycle**

*Main documentation section* <u>*Data spaces*</u> *[p 60]*

### *data space*

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

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

Data spaces are represented by the icon .

See also <u>inheritance</u> [p 20], <u>repository</u> [p 17], <u>data space merge</u> [p 21].

Related concept <u>Data spaces</u> [p 60].

### *reference data space*

The root ancestor data space of all data spaces in the EBX5 repository. As every data space merge must consist of a child merging into its parent, the reference data space is never eligible to be merged into another data space.

See also <u>data space</u> [p 21], <u>data space merge</u> [p 21], <u>repository</u> [p 17].

### *data space merge*

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

Related concept <u>Merge</u> [p 68].

### *snapshot*

A static copy of a data space 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 data spaces, but it can never be modified directly.

Snapshots are represented by the icon .

Related concept <u>Snapshot</u> [p 73]

## 3.5 **History**

### *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 22], transaction history view [p 22], history profile [p 22].

### *history profile*

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

See also history profile [p 22].

### *table history view*

A view containing a trace of all modifications that are made to 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 data set level.

### *transaction history view*

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

## 3.6 **Workflow modeling**

*Main documentation section Workflow models [p 100]*

### *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 100].

### *script task*

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

Script tasks are represented by the icon  .

See also <u>workflow model</u> [p 22].

### 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 <u>workflow model</u> [p 22].

### 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 <u>Data workflows</u> [p 124]*

### workflow publication

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

### 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 <u>workflow model</u> [p 22].

Related concept <u>Data workflows</u> [p 124].

### *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 <u>user task</u> [p 23].

# 3.8 **Data services**

*Main documentation section <u>Data services</u> [p 140]*

### *data service*

EBX5 shares master data according to the <u>Service-oriented architecture</u> (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 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 EBX5 entry point.

- a SOAP connector or entry point component for messages which allows external systems interacting with the EBX5 repository. This component accepts all SOAP XML messages corresponding to the EBX5 WSDL generator.

### *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: <u>Generating a WSDL for lineage</u> [p 145].

## 3.9 **Cross-domain**

### *node*

A node is an element of a tree view or a graph. In EBX5, 'Node' can carry several meanings depending on the context of use:

- In the <u>workflow model</u> [p 22] context, a node is a workflow step or condition.
- In the <u>data model</u> [p 18] context, a node is a group, a table or a field.
- In the <u>hierarchy</u> [p 20] context, a node represents the value of a dimension.
- In an <u>adaptation tree</u> [p 20], a node is a data set.
- In a <u>data set</u> [p 19], a node is the node of the data model valuated in the context of the data set 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 EBX5 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 data sets. Once you have a publication of your data model, you and other users can create data sets based upon it to hold the data that is managed by the EBX5 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 18]
- primary key [p 18]
- foreign key [p 18]
- table (in data model) [p 19]
- group [p 19]
- reusable type [p 19]
- validation rule [p 19]

# 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 EBX5 data model assistant (DMA) facilitates the development of data models.

> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective'.
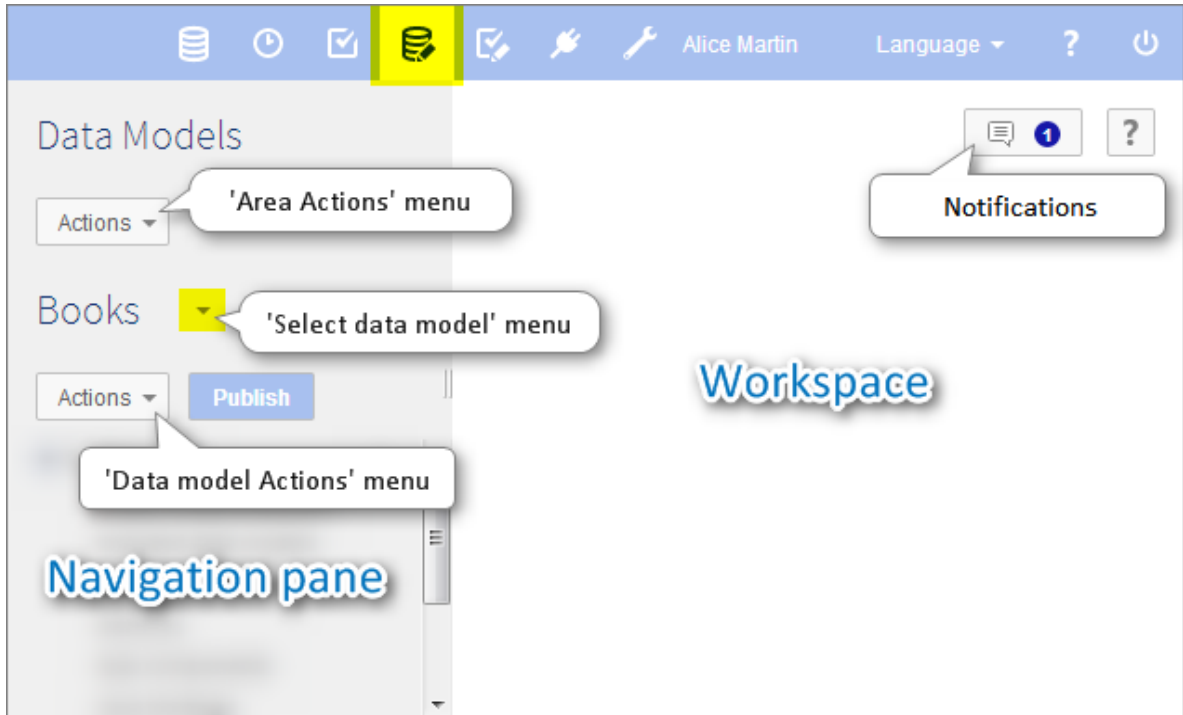
The navigation pane is organized into the following sections:

| | |
|---|---|
| **Configuration** | The technical configuration of the data model. |
| **Included data models** | Defines the data models included in the current model. The data types defined in included data models can be reused in the current model. |
| **Data structure** | The structure of the data model. Defines the relationship between the elements of the data model and provides access to the definition of each element. |
| **Simple data types** | Simple reusable types defined in the current data model. |
| **Complex data types** | Complex reusable types defined in the current data model. |
| **Included simple data types** | Simple reusable types defined in an included external data model. |
| **Included complex data types** | Complex reusable types defined in an included external data model. |

See also

## *Data model icons*

 field *[p 18]*

 primary key *[p 18]*

 foreign key *[p 18]*

 table *[p 19]*

 group *[p 19]*

Related concepts

CHAPTER **5**

# Creating a data model

This chapter contains the following topics:

1. Creating a new data model

## 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.

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 29] 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 29] 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 data set, as explained in Permissions [p 91].

## 6.3 **Data model properties**

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

| | |
|---|---|
| **Data set inheritance** | Specifies whether data set inheritance is enabled for this data model. Data set inheritance is disabled by default. See Data set inheritance [p 95] for more information. |
| **Disable auto-increment checks** | Specifies whether to disable if the check of an auto-incremented field value in associated data sets regarding to the "max value" found in the table being updated. |

## 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.

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:

1. Common actions and properties

2. Reusable types

3. Data model element creation details

4. Modifying existing elements

## 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, EBX5 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 <u>Included data models</u> [p 34] 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 37] 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 37] 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. Two main 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.

In both 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 data sets, 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 data sets, 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 18] 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 47]*

This chapter contains the following topics:

1. Basic element properties
2. Advanced element properties

# 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. |
| | This can be useful if the validation of the value depends on complex criteria or on the value of other fields. |
| | 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. |

### *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. |
| **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. |
| | 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**

### *Advanced properties for fields*

The following advanced properties are specific to fields.

### 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 data sets against the maximum value in the table being updated. |

### 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 20] in the glossary.

## *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 data set. |
| *Presentation* > **Record labeling** | Defines the fields to provide the default and localized labels for records in the table. |
| | **Attention:** Access rights defined on associated data sets 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. |
| | **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 data set. |
| **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**. |
| **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.<br><br>**Index name**: Unique name for this index.<br><br>**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 data sets. By default, all actions are allowed unless specific access rights are defined in a data set. |

## Uniqueness constraints

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

**Related concepts**  *Data validation controls on elements* *[p 47]*

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 41]*

This chapter contains the following topics:

1. Simple content validation
2. Advanced content validation

# 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. |
| | For fields of type 'Decimal', the default maximum length is 15. This is due to the fact that if the value exceeds 15 digits, loss of precision or rounding problems may occur when it is displayed in the user interface. Furthermore, if such an inaccurate value is displayed in a form and that form is submitted, the incorrect value will replace the original value. |
| **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 data sets 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. |

## 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.

| Foreign key constraint | |
| --- | --- |
| **Table** | Defines the table referenced by the foreign key. A foreign key references a table in the same data set by default. It can also reference a table in another data set in the same data space, or a data set in a different data space. |
| **Mode** | Location of the table referenced by the foreign key. 'Default': current data model. 'Other data set': different data set, in the same data space. 'Other data space': data set in a different data space. |
| **Referenced table** | XPath expression describing the location of the table. For example, /root/MyTable. |
| **Referenced data set** | Required if the table is located in another data set. The unique name of the data set containing the referenced table. |
| **Referenced data space** | Required if the table is located in another data space. The unique name of the data space containing the referenced table. |
| **Label** | Defines fields to provide the default and localized labels for records in the table. **Attention:** Access rights defined on associated data sets 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. |
| **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 an 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. |
| **Enumeration filled by another node** | Defines the possible values of this enumeration using a reference to another list or enumeration element. |

## *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 (data set 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 data set 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 41]

CHAPTER **10**

# 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 29] menu in the workspace.

This chapter contains the following topics:

1. Validating a data model
2. XML Schema Document (XSD) import and export
3. Duplicating a data model
4. Deleting a data model

## 10.1 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.

## 10.2 XML Schema Document (XSD) import and export

EBX5 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 29] 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 EBX5 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 34] for more information on declaring modules.

To perform an import select 'Import XSD' from the **data model 'Actions'** [p 29] 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.

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

## 10.3 **Duplicating a data model**

To duplicate a data model, select 'Duplicate' from the **data model 'Actions'** [p 29] menu for that data model. You must give the new data model a name that is unique in the repository.

## 10.4 **Deleting a data model**

To delete a data model, select 'Delete' from the **data model 'Actions'** [p 29] 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 data sets. 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 55] for more information on the publication process.

CHAPTER **11**

# Publishing a data model

This chapter contains the following topics:

## 11.1 About publications

Each data set based on an **embedded data model** in the EBX5 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, data sets 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 33] for more information.

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

## 11.2 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 EBX5 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, EBX5 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.

## 11.3 **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.

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 57] for more information on data model versions.

CHAPTER **12**

# Versioning a data model

This chapter contains the following topics:

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

## 12.1 About versions

You can create *versions* for data models in order to have branches 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.

## 12.2 Accessing versions

To see the existing versions of your data model, select 'Manage versions' from the **data model 'Actions'** [p 29] 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.

## 12.3 **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. |
| **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.

## 12.4 **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.

# Data spaces

CHAPTER **13**

# Introduction to data spaces

This chapter contains the following topics:

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

## 13.1 Overview

### *What is a data space?*

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 EBX5, this is made possible through the use of data spaces and snapshots.

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

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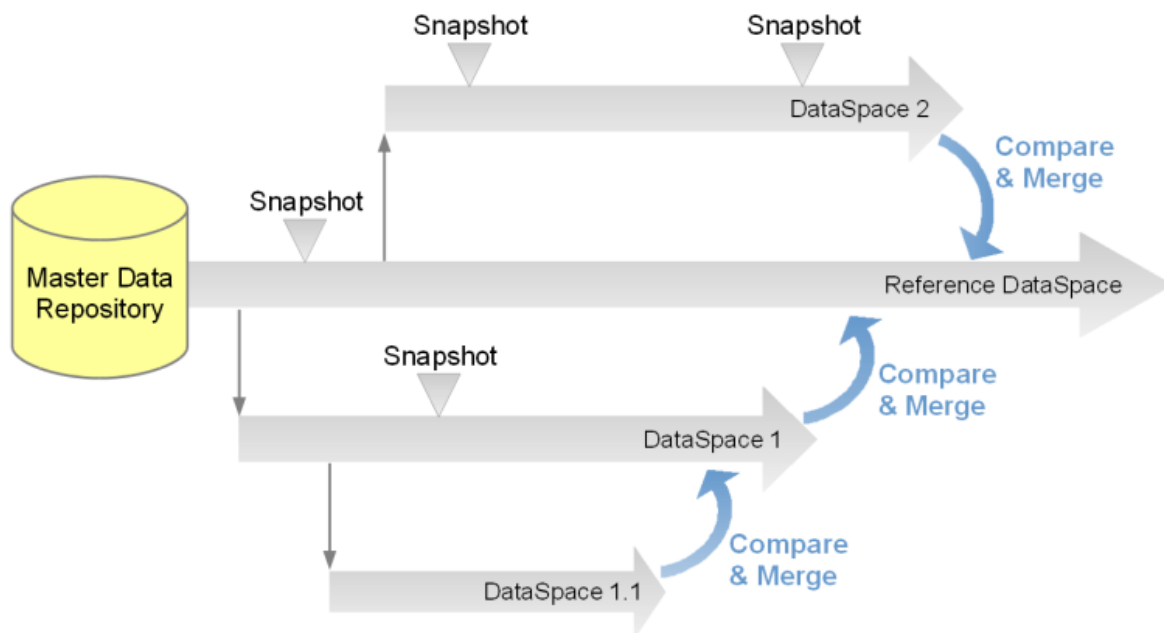


### Basic concepts related to data spaces

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

- data space [p 21]
- snapshot [p 21]
- data set [p 20]
- data space merge [p 21]
- reference data space [p 21]

## 13.2 Using the Data Spaces area user interface

Data spaces can be created, accessed and modified in the **Data Spaces** area.

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

The navigation pane displays all existing data spaces, while the workspace displays information about the selected data space and lists its snapshots.
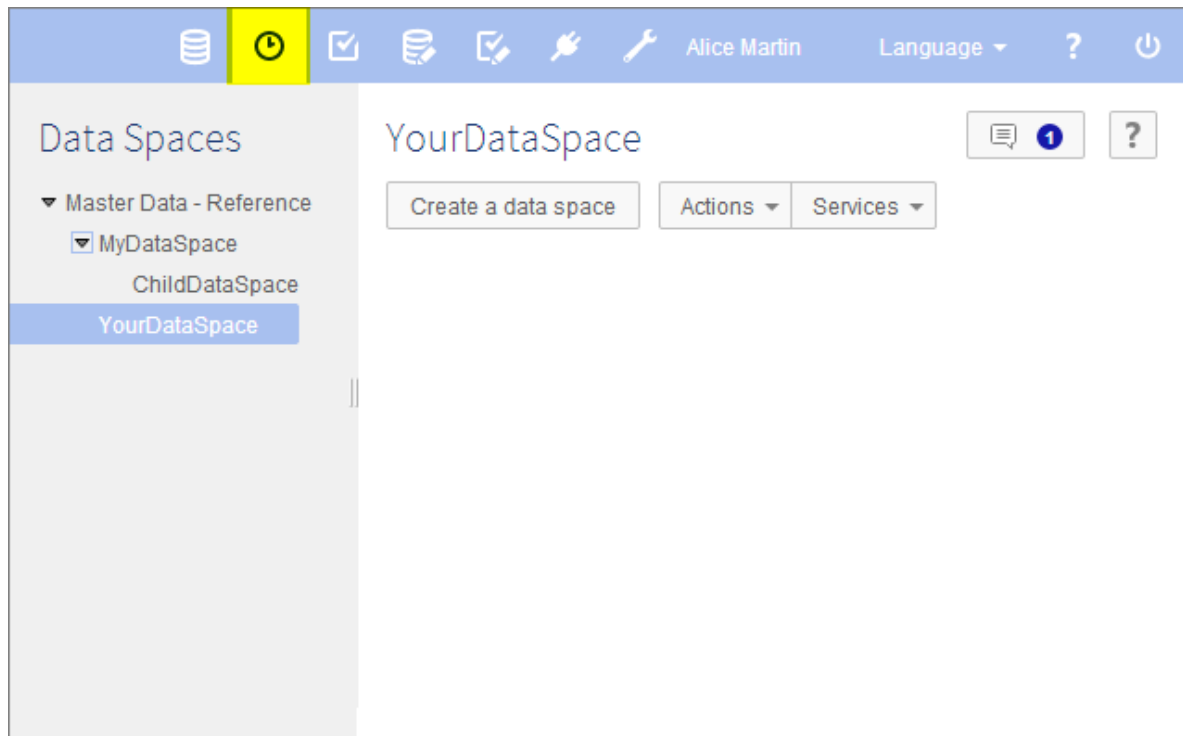


**See also**

[Creating a data space](#) *[p 63]*

[Snapshots](#) *[p 73]*

**Related concepts** [Data sets](#) *[p 78]*

CHAPTER **14**

# Creating a data space

This chapter contains the following topics:

## 14.1 Overview

By default, data spaces in EBX5 are in *semantic mode*. This mode offers full-featured data life cycle management.

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

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

The new data space will be a child data space 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 data space, which is the root of all semantic data spaces in the repository, semantic data spaces are always a child of another data space.

**See also** *Relational mode*

## 14.2 **Properties**

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

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

## 14.3 **Relational mode**

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

CHAPTER **15**

# Working with existing data spaces

This chapter contains the following topics:

1. Data space information
2. Data space permissions
3. Merging a data space
4. Comparing a data space
5. Validating a data space
6. Data space archives
7. Closing a data space

## 15.1 **Data space information**

Certain properties associated with a data space can be modified by selecting **Actions > Information** from the navigation panel in the Data Spaces area.

| | |
|---|---|
| **Documentation** | Localized labels and descriptions associated with the data space. |
| **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. <br><br> The available merge policies are: <br><br> • **Allows validation errors in result:** Child data spaces can be merged regardless of the validation result. This is the default policy. <br><br> • **Pre-validating merge:** A child data space can only be merged if the result would be valid. |
| **Current Owner** | Owner of the data space, who is, by default, allowed to modify its information and permissions. The owner does not necessarily have to be the creator of the data space. |
| **Child data space sort policy** | Defines the display order of child data spaces in data space trees. If not defined, the policy of the parent data space is applied. Default is 'by label'. |
| **Change owner** | Whether the current owner of the data space 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 data space is allowed to modify its permissions. If the value is 'Forbidden', only an administrator can modify the permissions of the data space. |

## 15.2 **Data space permissions**

### *General permissions*

| | |
|---|---|
| **Data space id** | The data space 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. |
| **Data space access** | The global access permission on the data space.<br><br>**Read-only**<br>• Can see the data space and its snapshots, as well as child data spaces, according to their permissions.<br>• Can see the contents of the data space depending on their permissions; cannot make modifications.<br><br>**Write**<br>• Can see the data space and its snapshots, as well as child data spaces, according to their permissions.<br>• Can modify the contents of the data space depending on their permissions.<br><br>**Hidden**<br>• Cannot see the data space nor its snapshots directly.<br>• From a child data space, the current data space can be seen but not selected.<br>• Cannot access the contents of the data space.<br>• Cannot perform any actions on the data space. |

### *Allowable actions*

Users can be allowed to perform the following actions:

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

## 15.3 **Merging a data space**

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

1. Both the parent and child data spaces 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 data space can be read, but they cannot be modified in any way.

   **Note:** This restriction on the parent data space means that, in addition to blocking direct modifications, other child data spaces cannot be merged until the merge in progress is finished.

2. Changes that were made in the child data space since its creation are integrated into its parent data space.

3. The child data space is closed.

4. The parent data space is unlocked.

### *Initiating a merge*

To merge a data space into its parent data space:

1. Select that data space in the navigation pane of the Data Spaces area.

2. In the workspace, select **Merge data space** from the **Actions** menu.

## *Reviewing and accepting changes*

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

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 data space state comparisons:

*   The current child data space compared to its initial snapshot.

*   The parent data space compared to the initial snapshot of the child data space.

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 data space, its initial snapshot and the parent data space, because data is likely to be modified both in the current data space and its parent.

The merge process also handles modifications to permissions on tables in the data space. 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 data set creations
*   Any changes to existing data
*   Record, data set, 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 data spaces.

Conflicts are categorized as follows:

*   A record or a data set creation conflict
*   An entity modification conflict
*   A record or data set 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 data space 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 data space 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 data space has set its child merge policy to 'Pre-validating merge', a dedicated data space is first created to hold the result of the merge. When the result is valid, this dedicated data space containing the merge result is automatically merged into the parent data space, and no further action is required.

In the case where validation errors are detected in the dedicated merge data space, you only have access to the original parent data space and the data space containing the merge result, named "[merge] *<name of child data space>*". 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 data space 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 data space.

### Setting the child merge policy of a data space

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

> **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 data space is automatically applied when merging a child data space. However, this setting is ignored during programmatic merge, which includes script tasks in data workflows.

See also  *Child merge policy* [p 69]

## *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 data spaces will remain until you unlock them in the Data Spaces area.

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

# 15.4 **Comparing a data space**

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

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

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

## 15.5 **Validating a data space**

To perform a global validation of the contents of a data space, select that data space 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 data set contained in the current data space.

## 15.6 **Data space archives**

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

### *Exporting*

To export a data space to an archive, select that data space 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.

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 data space', which exports all selected data to the archive. |
| | It may be useful to export only the differences between the data space 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. |
| **Data sets to export** | The data sets to export from this data space. For each data set, you can export its data values, permissions, and/or information. |

### *Importing*

To import content into a data space from an archive, select that data space 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 data space 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.

## 15.7 **Closing a data space**

If a data space is no longer needed, it can be closed. Once it is closed, a data space no longer appears in the **Data Spaces** area of the user interface, nor can it be accessed.

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

To close a data space, select **Actions > Close this data space**.

CHAPTER **16**

# Snapshots

This chapter contains the following topics:

## 16.1 **Overview of snapshots**

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

> **See also**   *Snapshot* *[p 21]*

## 16.2 **Creating a snapshot**

A snapshot can be created from a data space by selecting that data space in the navigation pane of the Data Spaces 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. |

## 16.3 **Viewing snapshot contents**

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

## 16.4 **Snapshot information**

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

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

## 16.5 **Comparing a snapshot**

You can compare the contents of a snapshot to those of another snapshot or data space 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 data space 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'.

## 16.6 **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 data set contained in the snapshot.

## 16.7 **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.

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

| | |
|---|---|
| **Name of the archive to create** | The name of the exported archive. |
| **Data sets to export** | The data sets to export from this snapshot. For each data set, you can choose whether to export its data values, permissions, and information. |

## 16.8 **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 data space in the Data Spaces area, nor can it be accessed.

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

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

# Data sets

CHAPTER **17**

# Introduction to data sets

This chapter contains the following topics:

1. Overview
2. Using the Data user interface

## 17.1 Overview

### *What is a data set?*

A data set 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 data sets based on its definition. If that data model is later modified and republished, all its associated data sets are automatically updated to match.

In a data set, 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 data set 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 data sets*

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

- data space [p 21]
- data set [p 20]
- record [p 19]
- field [p 18]
- primary key [p 18]
- foreign key [p 18]
- table (in data set) [p 20]
- group [p 19]

# 17.2 **Using the Data user interface**

Data sets can be created, accessed and modified in the **Data** area using the <u>Advanced perspective</u> [p 11] or from a specifically configured perspective. Only authorized users can access these interfaces.
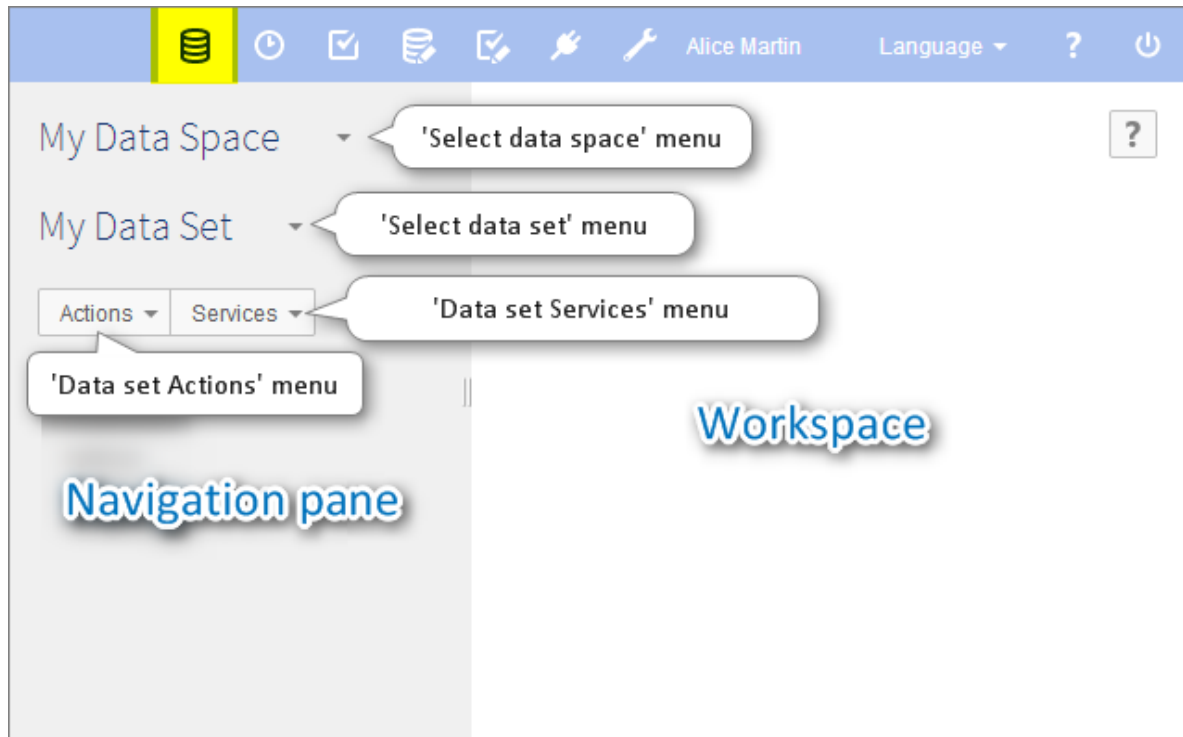


Select or create a data set using the 'Select data set' menu in the navigation pane. The data structure of the data set is then displayed in the navigation pane, while record forms and table views are displayed in the workspace.

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

Operations at the data set level are located in the **Actions** and **Services** menu in the navigation pane.

**See also**

> <u>*Creating a data set*</u> [p 81]
>
> <u>*Searching and filtering data*</u> [p 84]
>
> <u>*Working with records in the user interface*</u> [p 89]
>
> <u>*Inheritance*</u> [p 20]

**Related concepts**

> <u>*Data model*</u> [p 28]
>
> <u>*Data space*</u> [p 60]

CHAPTER **18**

# Creating a data set

This chapter contains the following topics:

1. Creating a root data set
2. Creating an inheriting child data set

## 18.1 **Creating a root data set**

To create a new root data set, that is, one that does not inherit from a parent data set, select the '**Select data set** [p 79]' ⏷ menu in the navigation pane, click the '**Create a data set**' 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 data set: 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 EBX5 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 data set, you must provide a unique name, without spaces or special characters. Optionally, you may provide localized labels for the data set, which will be displayed to users in the user interface depending on their language preferences.

> **Attention**
> Table contents are not copied when duplicating a data set.

## 18.2 **Creating an inheriting child data set**

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

To create a child data set, select the '**Select data set** [p 79]' ▼ menu in the navigation pane, then click the ➕ button next to the desired parent data set.

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

> **See also**  *Data set inheritance* [p 95]

CHAPTER **19**

# Viewing table data

This chapter contains the following topics:

1. 'View' menu
2. Sorting data
3. Searching and filtering data
4. Views
5. Views management

## 19.1 **'View' menu**

The 'View' drop-down menu allows accessing all major viewing features at a glance.

Views are managed in a dedicated sub-menu: 'Manage views' [p 88].

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 86] for more information.

## 19.2 **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, select *View > Sort criteria* 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*.

# 19.3 **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 data set level from the navigation pane.

## *Custom table searches*

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

# 19.4 **Views**

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. |
| **Other authorized profiles** | Profiles authorized to use the view. |
| **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 83]. |
| **Filter** | Defines filters for the records to be displayed in the table. |

## *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. |

**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.

**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 publication

A view can also be made available to other users of web components, workflow user tasks, or data services, by publishing it as a view publication. To publish a view, go to *View > Manage views > name of the view > Publish*. Once published, other eligible users can apply this view in their interfaces by selecting it in *View > name of the view* in their workspaces.

This view publication will be available to users belonging to the authorized profiles specified in the view.

## 19.5 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 applied rule will be the first that matches the user's profile.

> **Note**
>
> The 'Manage recommended view' feature is only available to data set 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.** |

CHAPTER **20**

# Editing data

This chapter contains the following topics:

## 20.1 **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.

## 20.2 **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 209]

[*XML Services*](#) [p 203]

CHAPTER **21**

# Working with existing data sets

This chapter contains the following topics:

## 21.1 Validating a data set

To validate a data set 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 data set 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.

## 21.2 Duplicating a data set

To duplicate an existing data set, select it from the '**Select data set** [p 79]' ▼ menu in the navigation pane, then select **Actions > Duplicate**.

## 21.3 Deactivating a data set

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

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

## 21.4 Managing data set permissions

Data set 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 17]

| | |
|---|---|
| **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. |
| **Data set actions** | Specifies the permissions for actions on the data set. |
| **Create a child data set** | Indicates whether the profile can create a child data set. Inheritance also must be activated in the data model. |
| **Duplicate the data set** | Indicates whether the profile can duplicate the data set. |
| **Delete the data set** | Indicates whether the profile can delete the data set. |
| **Activate/deactivate the data set** | Indicates whether the profile can modify the *Activated* property in the data set information. See Deactivating a data set [p 91]. |
| **Create a view** | Indicates whether the profile can create views and hierarchies in the data set. |
| **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 data set inheritance. |
| **Occult inherited record** | Indicates whether the profile can occult inherited records in the table. This permission is useful when using data set 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 data set 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 **22**

# Data set inheritance

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

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

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

**See also** *Data model configuration* [p 34]

This chapter contains the following topics:

1. Data set inheritance structure
2. Value inheritance

## 22.1 **Data set inheritance structure**

Once the root data set has been created, create a child data set from it using the ⊞ button in the data set selector in the navigation pane.

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

## 22.2 **Value inheritance**

When a child data set is created, it inherits all its field values from the parent data set. 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 data set inherits the records from the tables of its ancestor data sets. The table in the child data set 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 data set and does not exist in the parent data set. A root record is inherited by the child data sets of the current data set. |
| **Inherited** | An inherited record is one that is defined in an ancestor data set of the current data set. |
| **Overwritten** | An overwritten record is an inherited record whose values have been modified in the current data set. The overwritten values are inherited by the child data sets of the current data set. |
| **Occulted** | An occulted record is an inherited record which has been deleted in the current data set. It will still appear in the current data set as a record that is crossed out, but it will not be inherited in the child data sets of the current data set. |

When the inheritance button  is toggled on, it indicates that the record or value is inherited from the parent data set. 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 data sets of the current data set. | Standard modification of an existing record. The modified values will be inherited in the child data sets of the current data set. | Standard record deletion. The record will no longer appear in the current data set and the child data sets of the current data set. |
| **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 data set. | An overridden record can be returned to the inherited state, but its modified value will be lost.<br><br>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. |

# Workflow models

CHAPTER **23**

# Introduction to workflow models

This chapter contains the following topics:

1. Overview
2. Using the Workflow Models area user interface
3. Generic message templates
4. Limitations of workflows

## 23.1 Overview

### What is a workflow model?

Workflows in EBX5 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 22]
>
> *Data workflow (glossary)* [p 23]

### *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 22]
- user task [p 23]
- work item [p 24]
- workflow condition [p 23]
- sub-workflow invocation [p 23]
- wait task [p 23]
- data context [p 23]

## 23.2 Using the Workflow Models area user interface
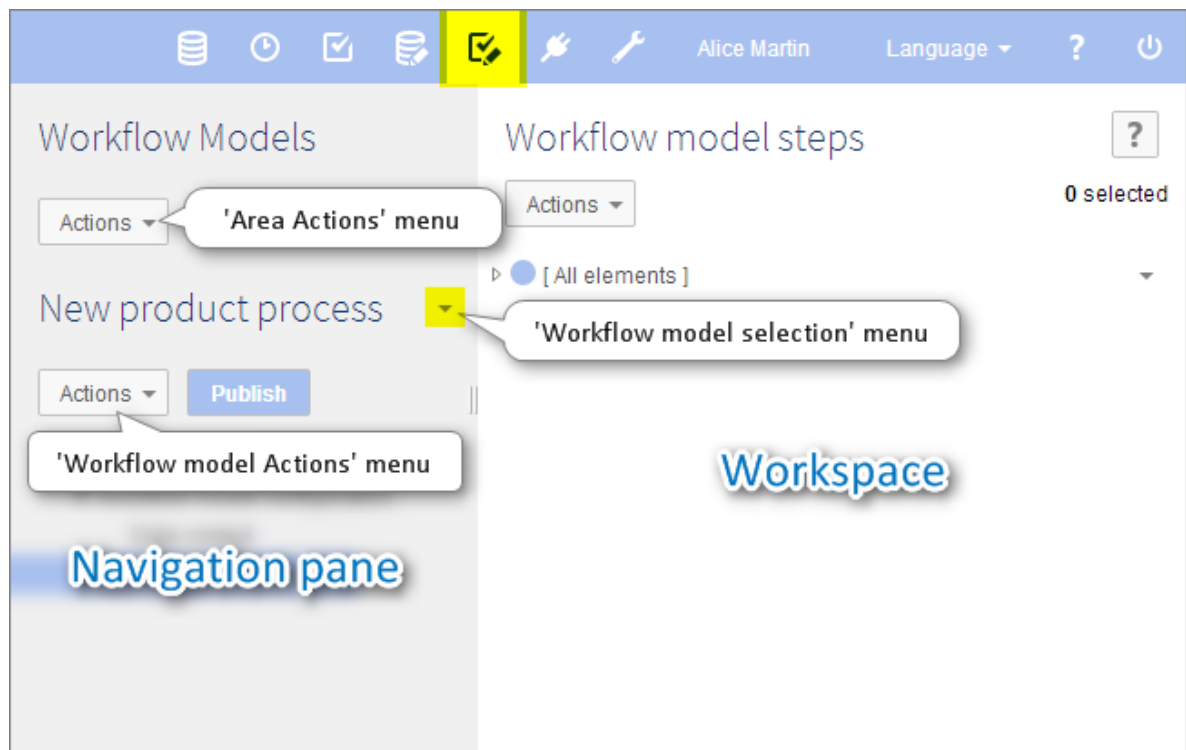


> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective'. Only authorized users can access these interfaces.

## 23.3 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.

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 preceding user task. |
| **workflow.lastDecision** | Last decisions made on the preceding 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. |
| **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. |
| **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`

## 23.4 **Limitations of workflows**

The following functionality is currently unsupported in EBX5:

- **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 124]

CHAPTER **24**

# Creating and implementing a workflow model

This chapter contains the following topics:

## 24.1 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.

## 24.2 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.

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.

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.

## 24.3 **User tasks**

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

### *Participants*

The participants of a user task are the profiles that are intended to work on the task.

By default, without service extensions, a separate work item is created and offered to each specified profile. If a profile refers to a user instead of a role, the work item is directly allocated to that user.

## *Service*

EBX5 includes the following built-in services that can be used for user tasks:

- Access a data space
- Access data (default service)
- Access the data space 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 data space
- Validate a data space, a snapshot or a data set

   **See also**  *EBX5 built-in services* [p 149]

## *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 > 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.

## 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.'

## 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'

## 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 101]

## 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.

# 24.4 **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** | EBX5 includes a number of built-in library script tasks, which can be used as-is. |
| **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. |

## *Library script tasks*

EBX5 includes the following built-in library script tasks:

- Close a data space
- Create a data space
- Create a snapshot
- Import an archive
- Merge a data space
- Send an email

> **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.

## 24.5 **Conditions**

Conditions are decision steps in workflows.

| | |
|---|---|
| **Library condition** | EBX5 includes a number of built-in library conditions, which can be used as-is. |
| **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. |

### *Library conditions*

EBX5 includes the following built-in library conditions:

- Data space modified?
- Data valid?
- Last user task accepted?
- Value null of empty?
- Value equals?

## 24.6 **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.

| 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. |

## 24.7 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.

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

CHAPTER **25**

# Configuring the workflow model

This chapter contains the following topics:

## 25.1 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 101] 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. |

## 25.2 **Workflow model properties**

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

| | |
|---|---|
| **Notification of start** | The list of profiles to which to send notifications, based on a template, when a data workflow is launched.<br><br>See Generic message templates [p 101]. |
| **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.<br><br>See Generic message templates [p 101]. |
| **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 131] for more information.<br><br>**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.<br><br>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.<br><br>If no work item has been opened, the table from which the workflow has been launched is displayed.<br><br>**Limitation:**This property will be ignored if the first step is a sub-workflow invocation. |
| --- | --- |
| **Permissions** | Permissions on actions related to the data workflows associated with the workflow model. |
| **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'. |

## 25.3 **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 135]

## 25.4 **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.

## 25.5 **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 121]*

CHAPTER **26**

# Publishing workflow models

This chapter contains the following topics:

## 26.1 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

## 26.2 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*

## 26.3 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 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 **27**

# Introduction to data workflows

This chapter contains the following topics:

1. Overview

## 27.1 **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 129]

*Launching and monitoring data workflows* [p 133]

*Administration of data workflows* [p 135]

*Permissions on associated data workflows* [p 116]

**Related concepts** *Workflow models* [p 100]

CHAPTER **28**

# Using the Data Workflows area user interface

This chapter contains the following topics:

1. Navigating within the interface
2. Navigation rules
3. Filtering items in views
4. Graphical workflow view

## 28.1 **Navigating within the interface**

Data workflow functionality is located in the **Data Workflows** area of the EBX5 user interface.
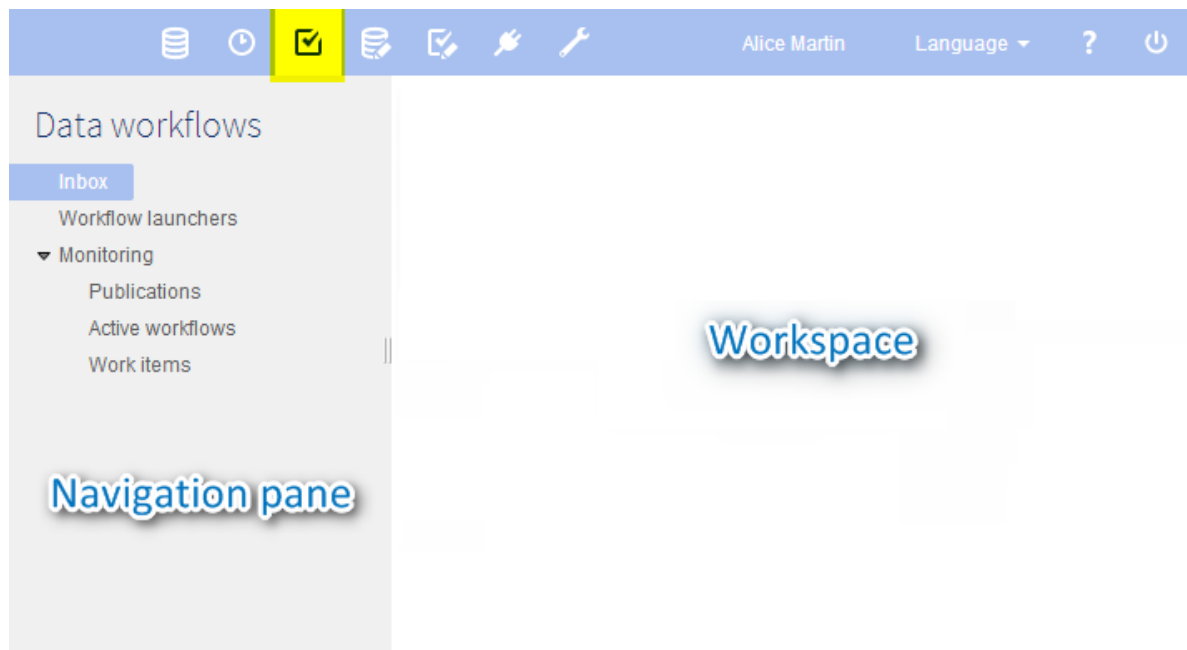


> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective' or from a specifically configured perspective. Only authorized users can access these interfaces.

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. |

## 28.2 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 106] 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 114] in workflow modeling.

## 28.3 **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.
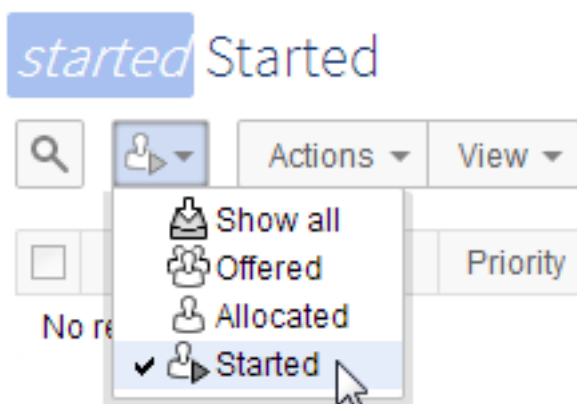


## 28.4 **Graphical workflow view**

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.

CHAPTER **29**

# Work items

This chapter contains the following topics:

## 29.1 **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 workflow model's publications will generate an individual work item for each of the participants listed in the user task.

### *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.

By default, for each individual user listed as a participant of the user task, the data workflow creates a work item in the *allocated* state. The defined user can directly begin working on the allocated work item by performing the action 'Start work item', at which time it moves to the *started* state.

By default, for each role included as a participant of the user task in the workflow model, the data workflow creates one work item in the *offered* state. Any user who is a member of that role can claim the work item using the action 'Take and start', thereby moving the work item to the *started* state.

Before a user has claimed the offered work item, a manager of the data workflow 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.

> **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.

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 work item states**
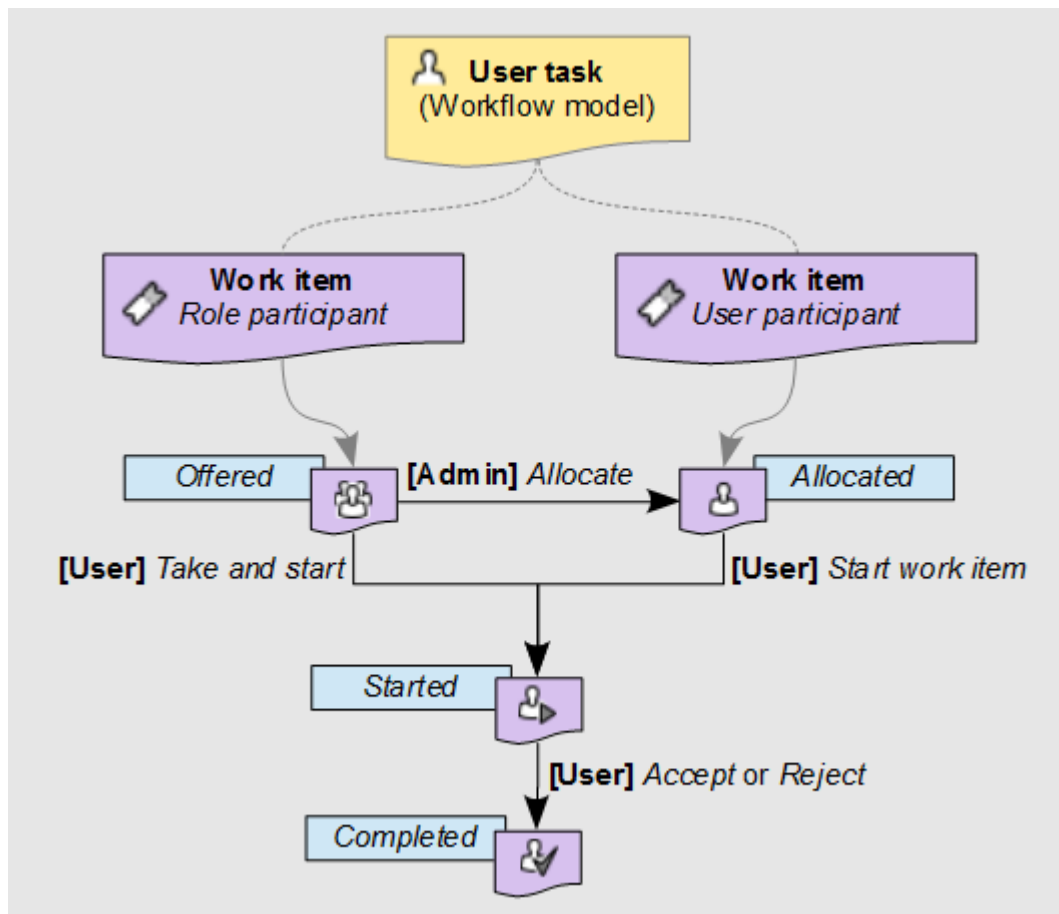


## 29.2 **Working on work items as a participant**

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.

## 29.3 **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 EBX5 repository.

**See also**  *user task (glossary)* [p 23]

**Related concepts**  *User tasks* [p 106]

CHAPTER **30**

# Launching and monitoring data workflows

This chapter contains the following topics:

## 30.1 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.

## 30.2 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.

## 30.3 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 129]

*Permissions on associated data workflows* [p 116]

**Related concepts**  *Workflow models* [p 100]

CHAPTER **31**

# Administration of data workflows

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

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

This chapter contains the following topics:

1. Overview of data workflow execution
2. Data workflow administration actions

## 31.1 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.

# 31.2 **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 <u>Disabling a workflow publication</u> .

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.

### 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.

### 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 responsability 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 and its history, 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.

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

## 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 116]*

# Data services

CHAPTER **32**

# Introduction to data services

This chapter contains the following topics:

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

## 32.1 Overview

### *What is a data service?*

A data service [p 24] is a standard Web service that can access data or interact with EBX5, thus offering some of the same functionality as is available through the user interface.

Data services can be dynamically generated based on data models from the **Data Services** area.

### *Lineage*

Lineage [p 24] 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 24]

## 32.2 **Using the Data Services area user interface**
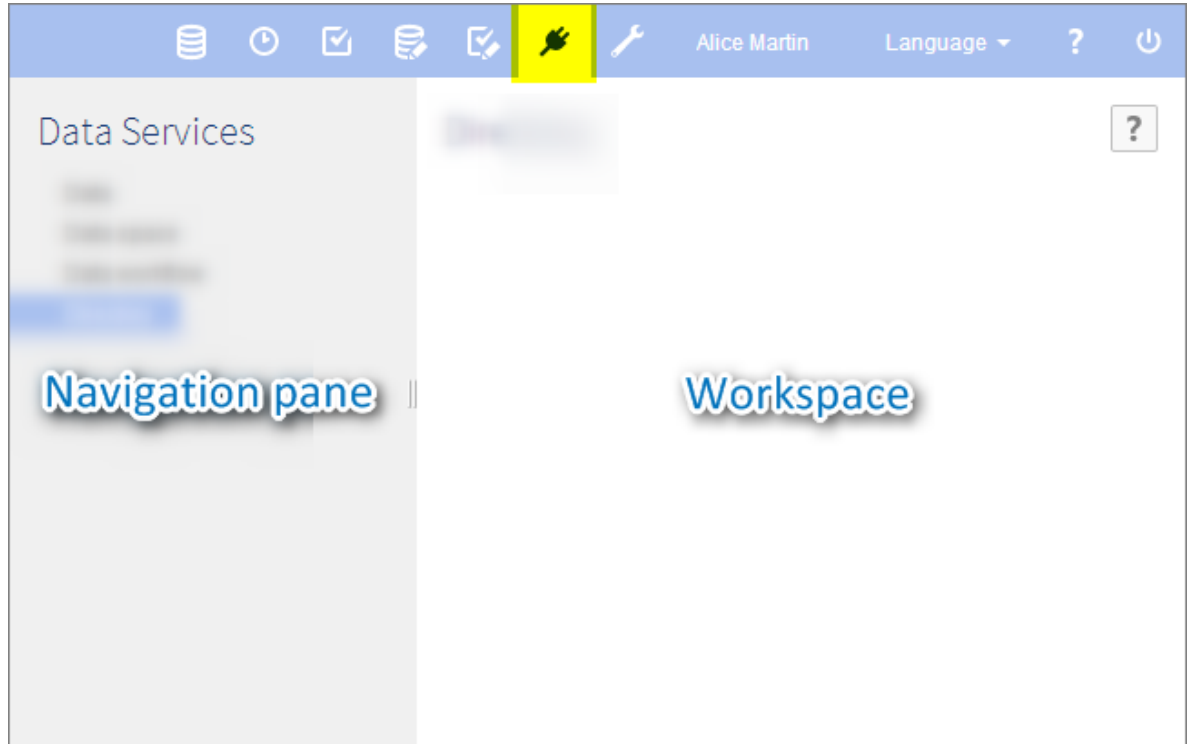


> **Note**
>
> This area is available only to authorized users in the 'Advanced perspective'.

**Related concepts**

*Data space* [p 60]

*Data set* [p 78]

*Data workflows* [p 124]

CHAPTER **33**

# Generating data service WSDLs

This chapter contains the following topics:

## 33.1 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 data set level or at the table level.

2. Identify the data space and data set 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 data sets*

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

- Get data set table(s) changes between data spaces 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 data space or snapshot
- Get credentials
- Run multiple operations on tables in the data set

## 33.2 **Generating a WSDL for data space operations**

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

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

### *Operations on data spaces*

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

- Create a data space
- Close a data space
- Create a snapshot
- Close a snapshot
- Merge a data space
- Lock a data space
- Unlock a data space
- Validate a data space or a snapshot
- Validate a data set

## 33.3 **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

# 33.4 **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 <u>WSDL for operations on data</u> [p 143].

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 data space and data set 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** <u>*Lineage*</u> *[p 140]*

# Reference Manual

# Integration

CHAPTER **34**

# Built-in UI services

EBX5 includes a number of built-in UI services. Built-in UI services can be used:

- when defining <u>workflow model tasks</u> [p 106]
- when defining <u>perspective action menu items</u> [p 12]

This reference page describes the built-in UI services and their parameters.

This chapter contains the following topics:

## 34.1 **Access a data space**

A workflow automatically considers that the data space selection service is complete.

Service name parameter: `service=@selectDataSpace`

### *Input parameters*

| Parameter | Label | Description |
|---|---|---|
| branch | Data space | The identifier of the specified data space - A data space 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 data space is required for this service. |

## 34.2 Access data (default service)

The 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 | Data space | The identifier of the specified data space - A data space 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 UI service or a trigger, for example. The default value is 'false'. Perspectives do not use this parameter. |
| instance | Data set | The value must be the reference of a data set that exists in the selected data space. |
| 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 data space 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 | Data set node (XPath) | The value must be a valid absolute location path in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax. |

## 34.3 Access the data space merge view

The merge.view service is automatically considered complete.

Service name parameter: `service=@merge.view`

### *Input parameters*

| Parameter | Label | Description |
|---|---|---|
| branch | Data space | The identifier of the specified data space - A data space 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. |

## 34.4 Compare contents

Workflows automatically consider the compare service as complete.

Service name parameter: `service=@compare`

## *Input parameters*

| Parameter | Label | Description |
|---|---|---|
| branch | Data space | The identifier of the specified data space - A data space or snapshot and a data space or snapshot to compare to are required for this service. |
| compare.branch | Data space to compare | The identifier of the data space to compare - A data space or snapshot and a data space 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 | Data set to compare | The value must be the reference of a data set that exists in the selected data space to compare. |
| compare.version | Snapshot to compare | The identifier of the snapshot to compare - A data space or snapshot and a data space 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 data set to compare. The notation must conform to a simplified XPath, in its abbreviated syntax. |
| instance | Data set | The value must be the reference of a data set that exists in the selected data space. |
| 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 data space or snapshot and a data space 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 data set. The notation must |

| Parameter | Label | Description |
|-----------|-------|-------------|
|  |  | conform to a simplified XPath, in its abbreviated syntax. |

## 34.5 **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` | Data space | The identifier of the specified data space - This field is required for this service. |
| `instance` | Data set | The value must be the reference of a data set that exists in the selected data space - 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` | Data set table (XPath) | The value must be a valid absolute location path of a table in the selected data set. 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. |

## 34.6 **Data workflows**

This service provides access to the data workflows user interfaces.

Service name parameter: `service=@workflow`

> **Note**
>
> This service is for perspective only.

## *Input parameters*

| Parameter | Label | Description |
|---|---|---|
| scope | Scope | Defines the scope of the user navigation 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 a XPath predicate surrounded by "[" and "]". |

## 34.7 **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 | Data space | The identifier of the specified data space - This field is required for this service. |
| instance | Data set | The value must be the reference of a data set that exists in the selected data space - 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. |

## 34.8 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` | Data space | The identifier of the specified data space - A data space is required for this service. |
| `instance` | Data set | The value must be the reference of a data set that exists in the selected data space - 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 data space is required for this service. |
| `xpath` | Data set table to export (XPath) | The value must be a valid absolute location path of a table in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service. |

## 34.9 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 | Data space | The identifier of the specified data space - A data space is required for this service. |
| instance | Data set | The value must be the reference of a data set that exists in the selected data space - 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 data space is required for this service. |
| xpath | Data set table to export (XPath) | The value must be a valid absolute location path of a table in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service. |

# 34.10 **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 | Data space | The identifier of the specified data space - A data space is required for this service. |
| instance | Data set | The value must be the reference of a data set that exists in the selected data space - 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 | Data set table to import (XPath) | The value must be a valid absolute location path of a table in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service. |

# 34.11 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 | Data space | The identifier of the specified data space - A data space is required for this service. |
| instance | Data set | The value must be the reference of a data set that exists in the selected data space - 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 | Data set table to import (XPath) | The value must be a valid absolute location path of a table in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service. |

## 34.12 Merge a data space

Workflows consider the merge service as complete when merger is performed and data space is closed.

Service name parameter: `service=@merge`

### *Input parameters*

| Parameter | Label | Description |
|---|---|---|
| branch | Data space | The identifier of the specified data space - A data space 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. |

## 34.13 Validate a data space, a snapshot or a data set

Workflows automatically consider the validation service as complete.

Service name parameter: `service=@validation`

## *Input parameters*

| Parameter | Label | Description |
|-----------|-------|-------------|
| branch | Data space | The identifier of the specified data space - A data space or snapshot is required for this service. |
| instance | Data set | The value must be the reference of a data set that exists in the selected data space. |
| 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 data space 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. |

CHAPTER **35**

# Introduction

This chapter contains the following topics:

1. Overview of data services
2. SOAP interactions
3. Data services security
4. Known limitations

## 35.1 Overview of data services

Data services allow external systems to interact with the data governed in the EBX5 repository using the SOAP and Web Services Description Language (WSDL) standards.

A number of WSDLs can be dynamically generated from data models, then used to perform operations such as:

- Selecting, inserting, updating, deleting, or counting records
- Getting the differences on a table between data spaces or snapshots, or between two data sets based on the same data model
- Getting the credentials of records

Other generic operations for:

- Creating, merging, or closing a data space
- Creating or closing a snapshot
- Validating a data set, data space, or a snapshot
- Starting, resuming or ending a data workflow

## 35.2 SOAP 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 in an optional SOAP header. For example:

```
<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>
```

## *Exceptions handling*

Exceptions are re-thrown to the consumer through the soap:fault element within a standard exception. 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>
```

# 35.3 **Data services security**

## *Authentication*

Authentication is mandatory. Several authentication methods are available and described below. The descriptions are ordered by priority (EBX5 applies the highest priority authentication method first).

- 'Basic Authentication Scheme' method based on the HTTP-Header `Authorization` in base 64, as described in RFC 2324.

  ```
  If the user agent wishes to send the userid "Alibaba" and password "open sesame",
  it will use the following header field: Authorization: Basic
  QWxpYmFiYTpvcGVuIHNlc2FtZQ==
  ```

- A simple authentication based on the specification Web Services Security UsernameToken Profile 1.0.

  Only the mode `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 Operations [p 177] from HTTP request.

- Standard authentication based on HTTP `Request`. User and password are extracted from request parameters. For more information, see Request parameters [p 171].

  > **Note**
  >
  > Only available for WSDL generation [p 169].

## *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>
```

> **Note**
>
> Only available for <u>Operations</u> [p 177].

### Lookup mechanism

Using HTTP, the web service connector attempts authentication in the following order:

1. Using an HTTP Request
2. Using the HTTP Header `Authorization`
3. Looking for a security header (WSS or custom)

When using JMS, the authentication process only looks for a security header (WSS or custom).

## 35.4 Known limitations

### 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 WSDL generation.

### 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 |

CHAPTER **36**

# WSDL generation

This chapter contains the following topics:

## 36.1 Supported standard

EBX5 generates a WSDL that complies with the W3C Web Services Description Language 1.1 standard.

## 36.2 Operation types

A WSDL can be generated for different types of operations:

| Operation type | WSDL description |
|---|---|
| custom | WSDL for EBX5 add-ons. |
| dataset | WSDL for data set and replication operations. |
| directory | WSDL for default EBX5 directory operations. It is also possible to filter data using the tablePaths [p 172] or operations [p 172] parameters. |
| repository | WSDL for data space or snapshot management operations. |
| tables | WSDL for operations on the tables of a specific data set. |
| userInterface | WSDL for user interface management operations (these operations can only be accessed by administrators). |
| workflow | WSDL for EBX5 workflow management operations. |

## 36.3 **Supported access methods**

EBX5 supports the following downloading methods:

- From the data services user interface
- From an HTTP request

Global access permissions can be independently defined for each method. For more information see .

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:<br><br>• No specific directory implementation is used. (The built-in Administrator role is only subject to this condition).<br>• Global access permissions are defined for the administration.<br>• 'Directory' data set permissions have writing access for the current profile. |
| repository | All profiles. |
| tables | All profiles. |
| userInterface | Built-in Administrator role or delegated administrator profiles, if all conditions are valid:<br><br>• Global access permissions are defined for the administration.<br>• 'User interface' data set permissions have writing access for the current profile. |
| workflow | All profiles. |

## 36.4 **WSDL download from the data services user interfaces**

An authorized user can download an EBX5 WSDL from the data services administration area.

> **Note**
>
> See generating data service WSDLs [p 143] in the user guide for more information.

## 36.5 **WSDL download using a HTTP request**

An application can download an EBX5 WSDL using an HTTP GET or POST request. The application has to be authenticated using a profile with appropriate rights.

## *Request details*

- HTTP(S) URL format:

  `http[s]://<host>[:<port>]/ebx-dataservices/<pathInfo>?<key - values>`

  Both `<pathInfo>` and `<key - values>` are mandatory. For more information on possible values see: Request parameters [p 171]. An error is returned for incorrect parameters.

- HTTP(S) response status codes:

| Status code | Information |
|---|---|
| 200 | The WSDL content was successfully generated and is returned by the request (optionally in an attachment [p 173]). |
| 400 | Bad request: request argument is incorrect. |
| 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 | An internal error occurred: request generates an error (a stack trace and a detailed error message are returned). |

> **Note**
>
> A detailed error message is returned for the HTTP response with status code 4xx.

- HTTP(S) parameters size restrictions:

| Request type | Information |
|---|---|
| GET | 2048 octets or more (HTTP Protocol Parameters).<br><br>*Note: Servers ought to be cautious about depending on URI lengths above 255 bytes, because some older client or proxy implementations might not properly support these lengths.* |
| POST | 2 mega octets or more (depending on the servlet/JSP container). Each key - value parameter is limited to 1024 characters. |

## *Request parameters*

A request parameter can be specified by one of the following methods:

- a path info on the URL (recommended)
- key values in a standard HTTP parameter.

For more detail, refer to the following table (some parameters do not have a path info representation):

| Parameter name | As path info | As key - values | Required | Description |
|---|---|---|---|---|
| WSDL | no | yes | yes | Used to indicate the WSDL download.<br>Empty value. |
| login | no | yes | no | A user identifier.<br>Required when the standard authentication method is used.<br>String type value. |
| password | no | yes | no | A password.<br>Required when the standard authentication method is used.<br>String type value. |
| type | yes | no | yes | An operation type [p 169].<br>Possible values are: `custom`, `dataset`, `directory`, `userInterface`, `repository`, `tables` or `workflow`.<br>String type value. |
| branch<br>version | yes | yes | (*) | A data space or a snapshot identifier.<br>(*) required for `tables` and `dataset` types, otherwise ignored.<br>String type value. |
| instance | yes | yes | (*) | A data set identifier.<br>String type value. |
| tablePaths | no | yes | no | A list of table paths.<br>Optional for `tables` or `directory` types, otherwise ignored.<br>If not defined, all tables are selected.<br>Each table path is separated by a comma character.<br>String type value. |
| operations | no | yes | no | Allows generating a WSDL for a subset of operations.<br>Optional for `tables` or `directory` operation types, otherwise ignored. If not defined, all operations for the given type are generated.<br>This parameter's value is a concatenation of one or more of the following characters:<br>• C = Count record(s)<br>• D = Delete record(s)<br>• E = Get credentials<br>• G = Get changes |

| Parameter name | As path info | As key - values | Required | Description |
|---|---|---|---|---|
| | | | | • I = Insert record(s)<br>• U = Update record(s)<br>• R = Read operations (equivalent to CEGS)<br>• S = Select record(s)<br>• W = Write operations (equivalent to DIU)<br>String type value. |
| namespaceURI | yes | yes | (**) | Unique name space URI of the custom web service.<br>(**)Is required when `type` parameter is set to `custom` types. Otherwise is ignored.<br>URI type value. |
| attachmentFilename | no | yes | (***) | The attachment file name.<br>(***) optional if `isContentInAttachment` parameter is defined and set to `true`. Otherwise is ignored.<br>String type value. |
| isContentInAttachment | no | yes | no | If value is `true`, the WSDL is downloaded as an attachment.<br>Boolean type value.<br>Default value is `false`. |

## *Request examples*

Some of the following examples are displayed in two formats: *path info* and *key - values*.

- The WSDL will contain all repository operations.

  ```
  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&login=<login>&password=<password>
  ```

- The WSDL will contain all tables operations for the data set 'dataset1' in data space 'dataspace1'.

  *Path info*

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/tables/dataspace1/dataset1?
  WSDL&login=<login>&password=<password>
  ```

  *Key - values*

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/tables?
  WSDL&login=<login>&password=<password>&
  branch=<dataspace1>&instance=<dataset1>
  ```

- The WSDL will contain all tables with only readable operations for the data set 'dataset1' in data space 'dataspace1'.

  *Path info*

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/tables/dataspace1/dataset1?
  WSDL&login=<login>&password=<password>&operations=R
  ```

  *Key - values*

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/tables?
  WSDL&login=<login>&password=<password>&
  branch=dataspace1&instance=dataset1&operations=R
  ```

- The WSDL will contain two selected tables operations for the data set 'dataset1' in data space 'dataspace1'.

  *Path info*

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/tables/dataspace1/dataset1?
  WSDL&login=<login>&password=<password>&tablePaths=/root/table1,/root/
  table2
  ```

  *Key - values*

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/tables?
  WSDL&login=<login>&password=<password>&
  branch=dataspace1&instance=dataset1&tablePaths=/root/table1,/root/table2
  ```

- The WSDL will contain custom web service operations for the dedicated URI.

  *Path info*

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/custom/urn:ebx-
  test:com.orchestranetworks.dataservices.WSDemo?
  WSDL&login=<login>&password=<password>
  ```

*Key - values*

```
http[s]://<host>[:<port>]/ebx-dataservices/custom?
WSDL&login=<login>&password=<password>&namespaceURI=urn:ebx-
test:com.orchestranetworks.dataservices.WSDemo
```

CHAPTER **37**

# Operations

This chapter contains the following topics:

## 37.1 Operations generated from a data model

For a data model used in an EBX5 repository, it is possible to dynamically generate a corresponding WSDL defining operations used to access the data sets that are an instance of this data model. 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  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.

## 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 data space to which the data set 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 data set belongs. | Either this parameter or the 'branch' parameter must be defined |
| instance | The unique name of the data set which contains the table to query. | Yes |
| predicate | XPath predicate [p 215] 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 203]. | Only required for the insert and update operations |
| viewPublication | This parameter can be combined with the predicate [p 179] parameter as a logical AND operation.<br><br>The behavior of this parameter is described in the section .<br><br>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.<br><br>This parameter cannot be used if the 'viewPublication' parameter is used. | No |
| blockingConstraintsDisabled | This property is available for all table updates data service operations.<br><br>If `true`, the validation process disables blocking constraints defined in the data model.<br><br>If this parameter is not present, the default is `false`. | No |

## *Select operation*

### Select request

```
<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 179]. | |
| version | See the description under Common parameters [p 179]. | |
| instance | See the description under Common parameters [p 179]. | |
| predicate | See the description under Common parameters [p 179].<br><br>This parameter can be combined with the viewPublication [p 179] parameter as a logical AND operation. | |
| viewPublication | See the description under Common parameters [p 179]. | |
| includesTechnicalData | The response will contain technical data if true. See also the optimistic locking [p 192] section.<br><br>Each returned record will contain additional attributes for this technical information, for instance:<br><br>`...<table`<br>`ebxd:lastTime="2010-06-28T10:10:31.046"`<br>` ebxd:lastUser="Uadmin"`<br>`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 |

## Select response

```
<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>
```

See also the optimistic locking [p 192] section.

## *Delete operation*

### 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>
 <occultIfInherit>boolean</occultIfInherit>
 <checkNotChangedSinceLastTime>dateTime</checkNotChangedSinceLastTime>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
</m:delete_{TableName}>
```

with:

| Element | Description | Required |
|---------|-------------|----------|
| branch | See the description under Common parameters [p 179]. | |
| instance | See the description under Common parameters [p 179]. | |
| predicate | See the description under Common parameters [p 179]. | |
| occultIfInherit | 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 192] section. | No |
| blockingConstraintsDisabled | See the description under Common parameters [p 179]. | |

### Delete response

```
<ns1:delete_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:delete_{TableName}Response>
```

with:

| Element | Description |
|---------|-------------|
| status | '00' indicates that the operation has been executed successfully.<br>'95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted. |

## *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 <u>Common parameters</u> [p 179]. |
| version | See the description under <u>Common parameters</u> [p 179]. |
| instance | See the description under <u>Common parameters</u> [p 179]. |
| predicate | See the description under <u>Common parameters</u> [p 179]. |

### 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>
 <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 <u>Common parameters</u> [p 179]. | |
| instance | See the description under <u>Common parameters</u> [p 179]. | |
| updateOrInsert | If `true` and the record does not currently exist, the operation creates the record. | No |
| byDelta | If `true` and an element does not currently exist in the incoming message, the target value is not changed.<br><br>If `false` and node is declared `hiddenInDataServices`, the target value is not changed.<br><br>The complete behavior is described in the sections <u>Insert and update operations</u> [p 204]. | No |
| blockingConstraintsDisabled | See the description under <u>Common parameters</u> [p 179]. | |
| data | See the description under <u>Common parameters</u> [p 179]. | |

## Update response

```
<ns1:update_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:update_{TableName}Response>
```

with:

| Element | Description |
|---|---|
| status | '00' indicates that the operation has been executed successfully.<br><br>'95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted. |

## *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>
 <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 179]. | |
| instance | See the description under Common parameters [p 179]. | |
| byDelta | If `true` and an element does not currently exist in the incoming message, the target value is not changed.<br><br>If `false` and node is declared `hiddenInDataServices`, the target value is not changed.<br><br>The complete behavior is described in the sections Insert and update operations [p 204]. | No |
| blockingConstraintsDisabled | See the description under Common parameters [p 179]. | |
| data | See the description under Common parameters [p 179]. | |

## Insert response

```
<ns1:insert_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <inserted>
  <predicate>./a='String'</predicate>
 </inserted>
</ns1:insert_{TableName}Response>
```

with:

| Element | Description |
|---------|-------------|
| status | '00' indicates that the operation has been executed successfully.<br><br>'95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted. |
| 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*

## Get changes requests

### Changes between two data sets:

```
<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>
</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 179]. | |
| version | See the description under Common parameters [p 179]. | |
| instance | See the description under Common parameters [p 179]. | |
| compareWithBranch | The identifier of the data space with which to compare. | One of either this parameter or the 'compareWithVersion [p 186]' parameter must be defined. |
| compareWithVersion | The identifier of the snapshot with which to compare. | One of either this parameter or the 'compareWithBranch [p 186]' parameter must be defined. |
| compareWithInstance | The identifier of the data set with which to compare. If it is undefined, instance [p 186] parameter is used. | No |
| resolvedMode | Defines whether or not the difference is calculated in resolved mode. Default is `true`. | No |
| pagination | Enables pagination context for the operations `getChanges` and `getChangesOnDataSet`.<br><br>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.<br><br>Get changes persisted context is built at first call according to the page size parameter in request.<br><br>For creation: Defines `pageSize` parameter.<br><br>For next: Defines `context` element with `identifier` from previous response.<br><br>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) |
| identifier (nested under context element) | Pagination context identifier. | Yes |

> **Note**
>
> If neither *compareWithBranch* nor *compareWithVersion* are specified, the comparison will be made with its parent:

- if the current data space or snapshot is a data space, the comparison is made with its initial snapshot (includes all changes made in the data space);

- if the current data space or snapshot is a snapshot, the comparison is made with its parent data space (includes all changes made in the parent data space since the current snapshot was created);

- returns an exception if the current data space is the 'Reference' data space.

## Get changes responses

### Changes between two data sets:

```
<ns1:getChangesOnDataSet_{schemaName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <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`.<br><br>Content under this element corresponding to an XML export of inserted records. | No |
| updated | Contains updated record(s). | No |
| changes (nested under context updated) | Only the group of field have been updated. | Yes |
| change (nested under context changes) | Group of fields have been updated with own `XPath predicate` attribute of the record. | Yes |
| data (nested under context changes) | Content under this element corresponding to an XML export of updated records under `changes` element. | No |
| deleted | Records have been deleted from context of request.<br><br>Content corresponding to a list of `predicate` element who contains the XPath predicate of record. | No |
| pagination | When pagination is enabled on request.<br><br>Get changes persisted context allows invoking the next page until last page or when a timeout is reached.<br><br>For next: Defines `context` element with `identifier`.<br><br>For last: Defines `context` element without `identifier`.<br><br>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 179]. | |
| version | See the description under Common parameters [p 179]. | |
| instance | See the description under Common parameters [p 179]. | |
| viewPublication | See the description under Common parameters [p 179]. | |

## 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 data set, 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 data space.

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.

```
<m:multi_ xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <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 179]. | |
| version | See the description under Common parameters [p 179]. | |
| instance | See the description under Common parameters [p 179]. | |
| blockingConstraintsDisabled | See the description under Common parameters [p 179]. | |
| 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.<br><br>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 data set (parameter instance).

- The select operation cannot use the pagination parameter.

## 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 from being performed on a record that was read earlier but may have changed in the meantime, an optimistic locking mechanism is provided.

In the select request, it is possible to ask for 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 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 be updated.

```
<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}>
```

The element `checkNotChangedSinceLastTime` can be used for one and only one record upon request. This means that if the predicate returns more than one record, the request will fail if the element `checkNotChangedSinceLastTime` is set.

## 37.2 **Operations on data sets and data spaces**

Parameters for operations on data spaces and snapshots are as follows:

| Element | Description | Required |
|---------|-------------|----------|
| branch | Identifier of the target data space on which the operation is applied. When not specified, the 'Reference' data space is used except for the merge data space operation where it is required. | One of either this parameter or the 'version' parameter must be defined. Required for the data space 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 data space child to create. If empty, it will be defined on the server side. | No |
| instance | The unique name of the data set 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. | No |
| branchToCopyPermissionFrom | Defines the identifier of the data space from which to copy the permissions. | No |
| documentation | Documentation for a dedicated language. Multiple documentation elements may be used for several languages. | No |

| Element | Description | Required |
|---------|-------------|----------|
| 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 data space*

### Validate data space request

```
<m:validate xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
</m:validate>
```

### Validate data space 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 data set*

### Validate data set 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="en-US"/>
</m:validateInstance>
```

### Validate data set 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 data space*

### Create data space 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 data space 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 data space*

### Lock data space 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 data space 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.<br>'94' indicates that the data space has been already locked by another user.<br>Otherwise, a SOAP exception is thrown. |

## *Unlocking a data space*

### Unlock data space request

```
<m:unlockBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
</m:unlockBranch>
```

### Unlock data space 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 data space*

### Merge data space 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 data space operation. Sets whether the specified data space and its associated snapshots will be deleted upon merge. | No |
| deleteHistoryOnMerge | This parameter is available for the merge data space operation. Sets whether the history associated with the specified data space will be deleted upon merge. Default value is `false`. | No |

> **Note**
>
> The merge decision step is bypassed during merges performed through data services. In such cases, the data in the child data space automatically overrides the data in the parent data space.

### Merge data space 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 data space or snapshot*

## Close data space or snapshot request

### Close data space 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 data space and close snapshot operations. Sets whether the specified snapshot, or data space and its associated snapshots, will be deleted upon closure. | No |
| deleteHistoryOnClose | This parameter is available for the close data space operation. Sets whether the history associated with the specified data space will be deleted upon closure. Default value is `false`. | No |

## Close data space or snapshot response

### Close data space 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>
```

# 37.3 **Operations on data workflows**

Parameters for operations on data workflows are as follows:

| Element | Description | Required |
|---------|-------------|----------|
| parameters | Input parameters for the process instance to be created. | No |
| parameter (nested under the `parameters` element). Multiple `parameter` elements may be used. | An input parameter for the process instance. | 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 process instance from a published process key. It is possible to start a workflow with localized documentation and specific input parameters (with name and optional value).

**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>
 <parameters>
  <parameter>
   <name>String</name>
   <value>String</value>
  </parameter>
 </parameters>
</m:workflowProcessInstanceStart>
```

with:

| Element | Description | Required |
|---------|-------------|----------|
| publishedProcessKey | Key of the published workflow to start. | Yes |
| documentation | See the description under <u>Common parameters</u> [p 193]. | No |
| parameters | See the description under <u>Common parameters</u> [p 199]. | 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 parameter 'processInstanceKey'. While it remains available for backward compatibility, it will eventually be removed in a future major version. | No |
| processInstanceKey | Key of the workflow process instance. | No |

## *Resume a workflow*

Resume a workflow process instance in a wait step from a resume identifier. It is possible to defined specific input parameters (with name and optional value).

**Sample request:**

```
<m:workflowProcessInstanceResume xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <resumeId>String</resumeId>
 <parameters>
  <parameter>
   <name>String</name>
   <value>String</value>
  </parameter>
 </parameters>
</m:workflowProcessInstanceResume>
```

with:

| Element | Description | Required |
|---------|-------------|----------|
| resumeId | Resume identifier of the waiting task. | Yes |
| parameters | See the description under <u>Common parameters</u> [p 199]. | 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 process instance has not been found. '21' indicates that the event has already been received. | Yes |
| processInstanceKey | Key of the workflow process instance. This parameter is returned if the operation has been executed successfully. | No |

## *End a workflow*

End a workflow process instance from its key representation.

**Sample request:**

```
<m:workflowProcessInstanceEnd xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <processInstanceKey>String</processInstanceKey>
</m:workflowProcessInstanceEnd>
```

with:

| Element | Description | Required |
|---------|-------------|----------|
| processInstanceKey | Key of the workflow process instance. | 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 |

CHAPTER **38**

# XML import and export

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 data set.

This chapter contains the following topics:

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

## 38.1 **Imports**

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

## *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 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.

| State in source XML document | Behavior |
|---|---|
| Element does not exist in the source document | **If 'by delta' mode is disabled (default):**<br>Target field value is set to one of the following:<br>• If the element defines a default value, the target field value is set to that default value.<br>• If the element is of a type other than a string or list, the target field value is set to `null`.<br>• If the element is an aggregated list, the target field value is set to an empty list.<br>• 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.<br>• If the element (simple or complex) is hidden in data services, the target value is not changed.<br>**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.<br>**If 'by delta' mode has been enabled through data services or the Java API:**<br>• For the `update` operation, the field value remains unchanged.<br>• 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.<br>• For non-`xs:string` type nodes, an exception is thrown in conformance with XML Schema. |
| 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.<br>In order to use the `xsi:nil="true"` attribute, you must import the namespace `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`. |

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 `x: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. The timestamp associated with the current record will be compared to this timestamp. If they are different, the update is rejected.

## 38.2 **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. |
| **Include computed values** | Specifies whether computed values will be exported. |
| **Is indented** | Specifies whether the file should be indented to improve readability. |

## 38.3 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 |

## 38.4 Known limitations

### *Association fields*

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

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

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

CHAPTER **39**

# CSV import and export

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 data set.

This chapter contains the following topics:

## 39.1 **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** | 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.<br><br>**Note:** Inheritance is always ignored, if the table data set 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.<br><br>**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.<br><br>**Note:** If this option is selected, the exported file will not be able to be re-imported. |
| **Include computed values** | Specifies whether computed values will be exported. |
| **Column header** | Specifies the whether or not to include column headers in the CSV file.<br><br>• **No header**<br>• **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.<br>• **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. |
| **List separator** | Specifies the separator to use for values lists. The default separator is line return. |

## 39.2 **Imports**

When importing a CSV file, you must specify one of the following import modes, which will determine how the import procedure handles the source records.

| | |
|---|---|
| **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. |

In order to consider the inheritance during a CSV import, the following option has to be specified.

| | |
|---|---|
| **Enable inheritance** | 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 data set of the table has no parent or if the table has no inherited field. |

## 39.3 **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 table references. 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 node'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.

### *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 |

## 39.4 **Known limitations**

### *Aggregated lists of groups*

The CSV import and export services do not support importing 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 203].

## 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 importing association values, that is multi-valued of complex type elements.

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

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

CHAPTER **40**

# Supported XPath syntax

This chapter contains the following topics:

1. Overview
2. Example expressions
3. Syntax specifications for XPath expressions

## 40.1 **Overview**

The XPath notation used in EBX5 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 on label

```
osd:label(./delivery_date)='12/30/2014' and ends-with(osd:label(../
adress),'Beijing - China')
```

# 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 braces, 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-equal-case-insensitive`, `osd:starts-with-case-insensitive`, `osd:ends-with-case-insensitive`, or `osd:contains-case-insensitive` | |
| *<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'` |
| *<date criterion>* | Quoted and formatted as 'yyyy-MM-dd' | `'2007-12-31'` |

| Expression | Format | Notes/Example |
|---|---|---|
| *<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)]`.

## 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).

## 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.

**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*

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/tableB[ fkB = '123|2008-01-21' ]`, where the string "123|2008-01-21" is a representation of the entire primary key value.

- `/root/tableB[ 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/tableB[ fkB/id >= 123 ]`, where any number operator could be used, as the targeted primary key field is of type `xs:int`.

- `/root/tableB[ 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/tableB[ fkB = "" ]` is not valid as the targetted primary key has two columns.

- `/root/tableB[ osd:is-null(fkB) ]` checks if a foreign key is `null` (notdefined).