EBX5

Product Documentation

EBX5 Version 5.8.1

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

Implementing data models

Publishing and versioning data models

Dataspaces

Datasets

Workflow models

Data workflows

Managing data workflows

Data services

Reference Manual

Integration

Data services

SOAP / WSDL

REST

File import and export services

Other

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

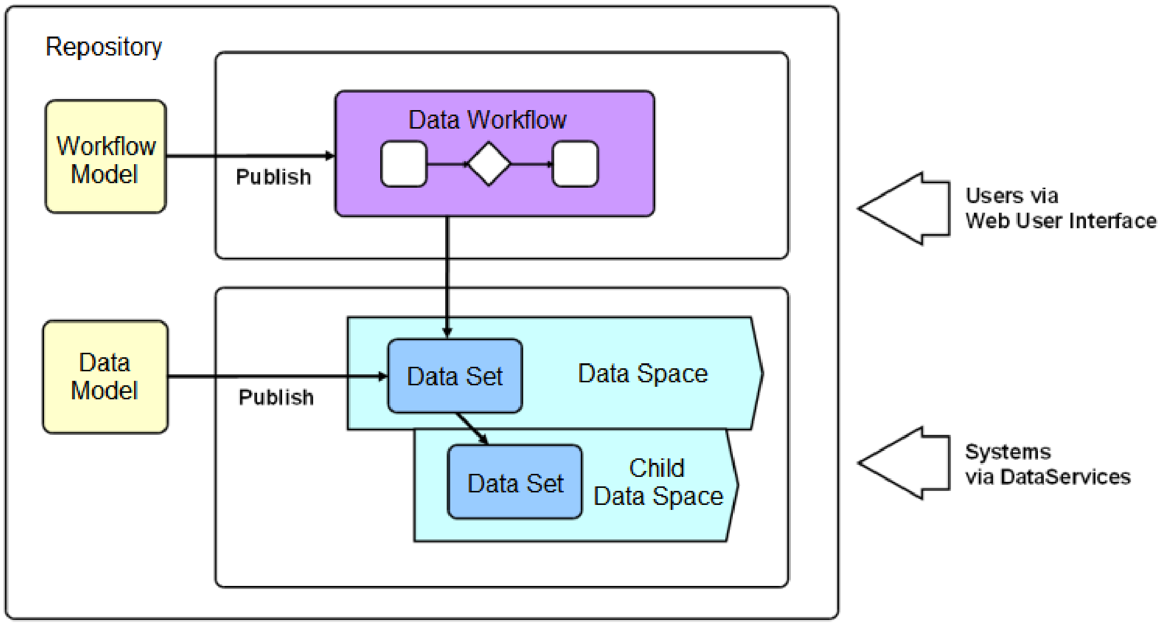
An MDM project using EBX5 starts with the creation of a *data model*. This is where tables, fields, links and business rules related to the master data are defined. Examples of modeled data include product catalogs, financial hierarchies, lists of suppliers or simple reference tables.

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

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

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

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



See also

- *Data modeling* [p 18]
- *Datasets* [p 19]
- *Dataspaces* [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 several customized perspectives have been created, the tiles icon 'Select perspective' allows the user to switch between available perspectives.

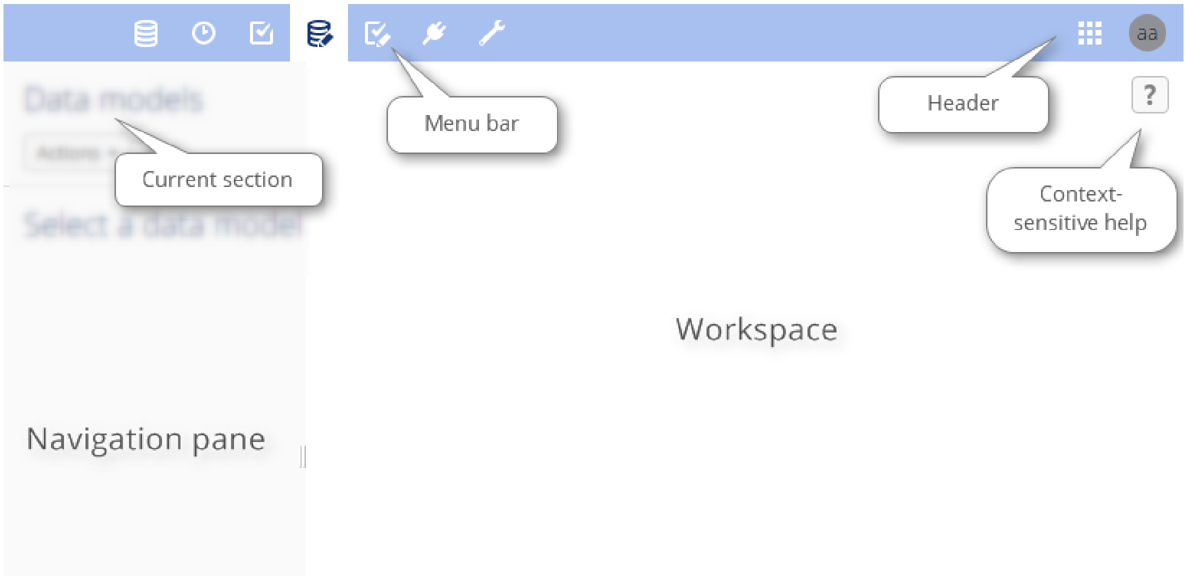
The advanced perspective is accessible by default.

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 areas, referred to as the following in the documentation:

- **Header:** Displays the avatar of the user currently logged in and the perspective selector (when more than one perspective is available). Clicking on the user's avatar gives access to the user pane.
- **Menu bar:** The functional categories accessible to the current user.
- **Navigation pane:** Displays context-dependent navigation options. For example: selecting a table in a dataset, or a work item in a workflow.
- **Workspace:** Main context-dependent work area of the interface. For example, the table selected in the navigation pane is displayed in the workspace, or the current work item is executed in the workspace.

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

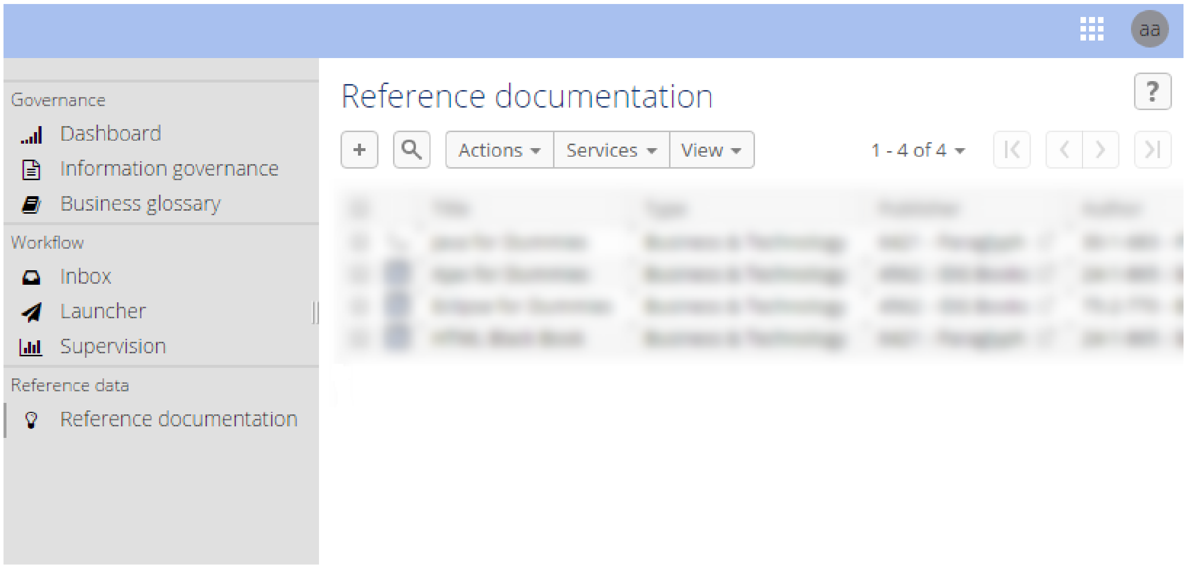


2.3 Perspectives

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

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

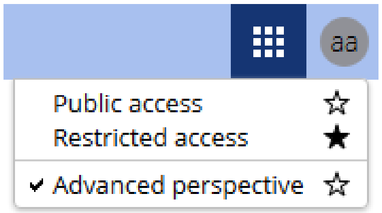
Example of a hierarchical menu:



Favorite perspectives

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

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



2.4 **User pane**

General EBX5 features are grouped in the user pane. It can be accessed by clicking on the avatar (or user's initials) in the upper right corner of any page.

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

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

Avatar

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

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

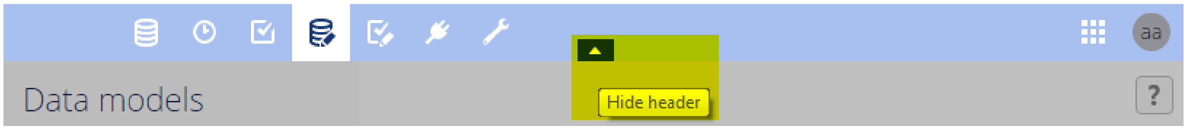
Density

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

2.5 User interface features

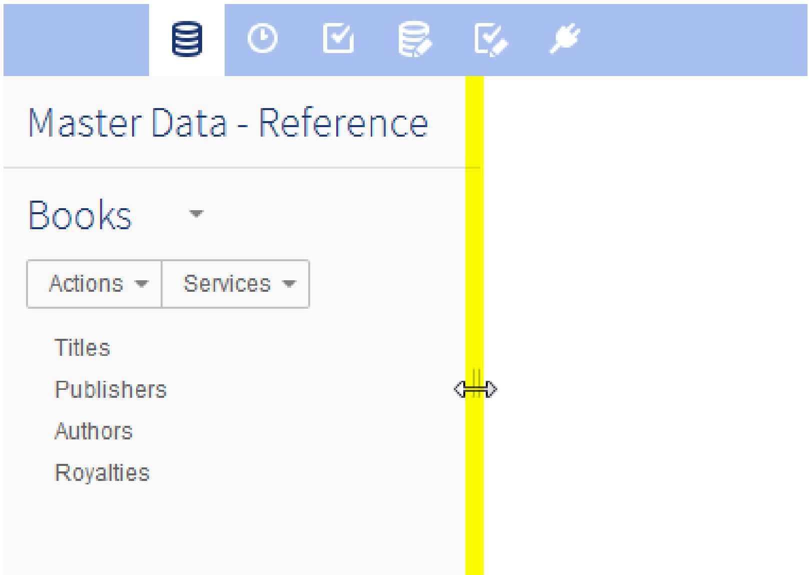
Hiding the header

It is possible to hide the header in the user interface by hovering over it, then clicking the arrow 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.



2.6 Where to find EBX5 help

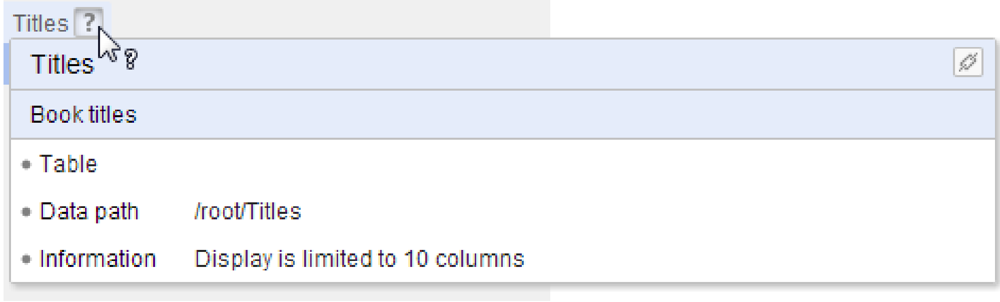
In addition to the full standalone product documentation accessible via the user pane [p 13], help is accessible in various forms within the interface.

Context-sensitive help

When browsing any workspace in EBX5, context-specific help is available by clicking on the question mark located on the upper right corner of the workspace. The corresponding chapter from the product documentation will be displayed.

Contextual help on elements

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



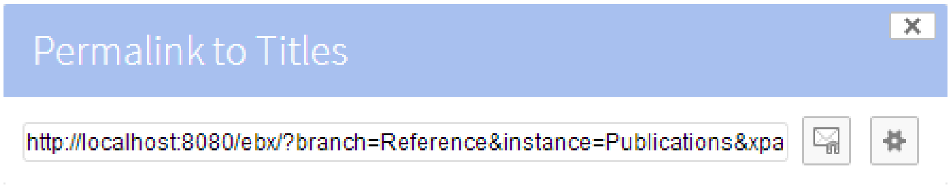
Titles ?

Titles

Book titles

- Table
- Data path /root/Titles
- Information Display is limited to 10 columns

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



Permalink to Titles

http://localhost:8080/ebx/?branch=Reference&instance=Publications&xpa

CHAPTER 3

Glossary

This chapter contains the following topics:

3.1 Governance

repository

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

See also dataspace [p 21].

profile

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

See also user [p 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 and 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 datasets, which are instances of data models that contain the data being managed by the repository.

See also dataset [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 43], Controls on data fields [p 49].

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

primary key

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

Primary keys are denoted by the icon 🔑.

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 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.

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

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 **Datasets**

Main documentation section Datasets [p 82]

record

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

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

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

table (in dataset)

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

Tables are represented by the icon ▦.

See also record [p 19], primary key [p 18].

dataset

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

Datasets are represented by the icon 🗏.

See also table (in dataset) [p 20], field [p 18], group [p 19], views [p 20].

Related concept Datasets [p 82].

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

inheritance

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

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

Inheritance from the parent dataset is represented by the icon ⤷.

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

Inherited fields are represented by the icon ⤵.

views

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

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

Related concepts Views [p 89] and Hierarchies [p 90].

recommended view

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

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

Related concept Recommended views [p 92].

favorite view

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

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

Related concept Manage views [p 93].

3.4 Data management life cycle

Main documentation section Dataspaces [p 64]

dataspace

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

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

See also inheritance [p 20], repository [p 17], dataspace merge [p 21].

Related concept Dataspaces [p 64].

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

reference dataspace

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

See also dataspace [p 21], dataspace merge [p 21], repository [p 17].

dataspace merge

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

Related concept Merge [p 72].

snapshot

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

Snapshots are represented by the icon 📷.

Related concept Snapshot [p 77]

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

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 dataspaces should have their 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 in a given table, including record creations, updates, and deletions. Each entry includes transactional information, such as a timestamp and the user performing the action, as well as the data at the conclusion of the transaction. This information can also be consulted at a record or dataset level.

transaction history view

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

3.6 Workflow modeling

Main documentation section Workflow models [p 106]

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 106].

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

script task

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

Script tasks are represented by the icon ⚙.

See also workflow model [p 23].

user task

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

User tasks are represented by the icon 👤.

See also workflow model [p 23].

workflow condition

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

Workflow conditions are represented by the icon ◇.

sub-workflow invocation

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

wait task

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

data context

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

3.7 Data workflows

Main documentation section Data workflows [p 130]

workflow publication

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

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

data workflow

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

See also workflow model [p 23].

Related concept Data workflows [p 130].

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

work list

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

work item

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

Allocated work items are represented by the icon 🏷.

See also user task [p 23].

3.8 Data services

Main documentation section Data services [p 148]

data service

EBX5 shares master data according to the Service-oriented architecture (SOA) by using XML web services. Since all data services are generated directly from models or built-in services they can be used to access part of the features available from the user interface.

Data services offer:

- a WSDL model-driven and built-in generator to build a communication interface. It can be produced through the user interface or the HTTP(S) connector for a client application. XML messages are communicated to the EBX5 entry point.
- a SOAP connector or entry point component for SOAP messages which allows external systems interacting with the EBX5 repository. This connector responds to requests coming from the WSDL produced by EBX5. This component accepts all SOAP XML messages corresponding to the EBX5 WSDL generator.

- A RESTful connector, or entry point for the select operations, allows external systems interrogating the EBX5 repository. After authenticating, it accepts the request defined in the URL and executes it according to the permissions of the authenticated user.

lineage

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

Related concept: Generating a WSDL for lineage [p 152].

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 workflow model [p 22] context, a node is a workflow step or condition.
- In the data model [p 18] context, a node is a group, a table or a field.
- In the hierarchy [p 20] context, a node represents a value of a dimension.
- In an adaptation tree [p 20], a node is a dataset.
- In a dataset [p 19], a node is the node of the data model evaluated in the context of the dataset or the record.

Data models

CHAPTER 4

Introduction to data models

This chapter contains the following topics:

4.1 Overview

What is a data model?

The first step towards managing data in 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 datasets. Once you have a publication of your data model, you and other users can create datasets based upon it to contain the data that is managed by the 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:

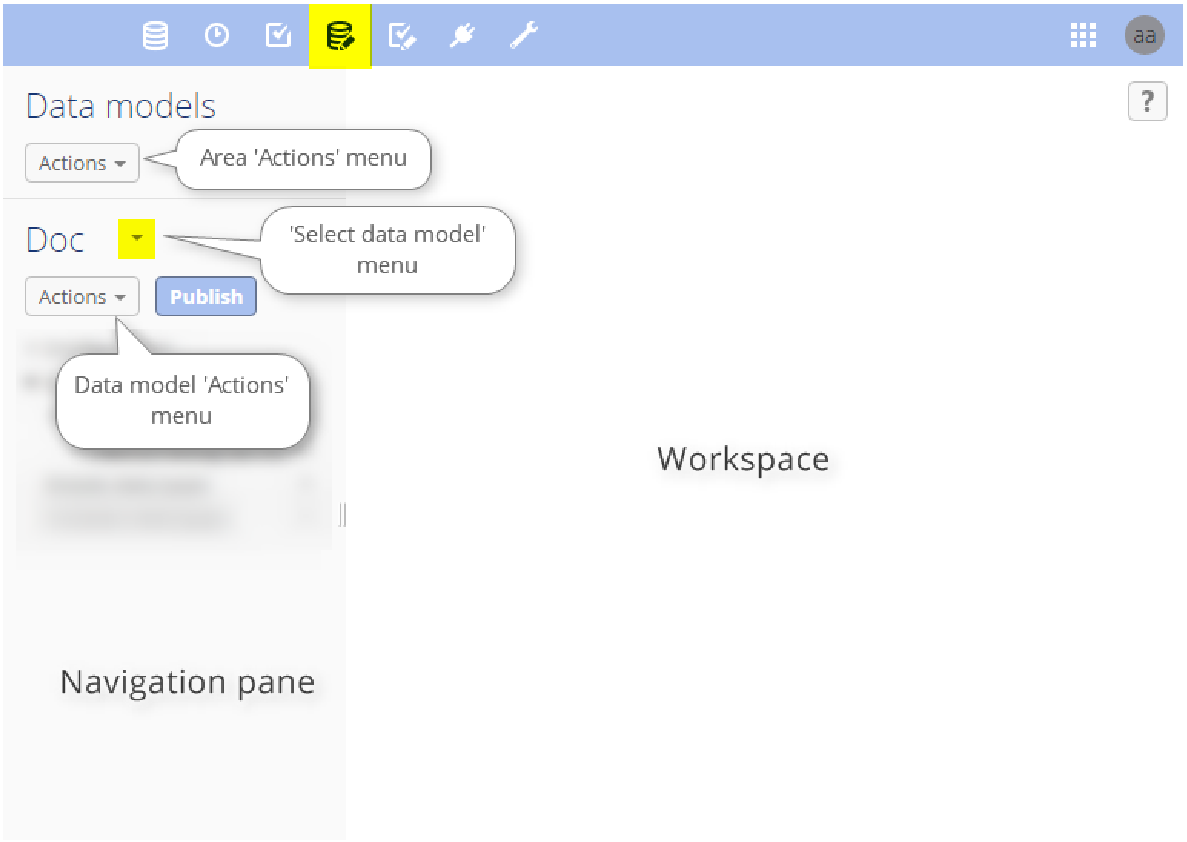
- field [p 18]
- primary key [p 18]
- foreign key [p 18]
- table (in data model) [p 19]
- group [p 19]
- reusable type [p 19]
- validation rule [p 19]

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.
Global properties	Defines the global properties of the data model.
Included data models	Defines the data models included in the current model. All types defined in included data models can be reused in the current model.
Add-ons	Specifies which add-ons are used by the data model. These add-ons will have the capacity to enrich the current data model after the publication by adding properties and constraints to the elements of the data model.
Data services	Specifies the WSDL operations' suffixes that allow to refer to a table in the data service operations using a unique name instead of its path.
Data structure	The structure of the data model. Defines the relationship between the elements of the data model and provides access to the definition of each element.
Simple data types	Simple reusable types defined in the current data model.
Complex data types	Complex reusable types defined in the current data model.
Included simple data types	Simple reusable types defined in an included external data model.
Included complex data types	Complex reusable types defined in an included external data model.

See also

Implementing the data model structure *[p 37]*

Configuring the data model *[p 35]*

Reusable types *[p 39]*

Data model element icons

field [p 18]

primary key [p 18]

foreign key [p 18]

table [p 19]

group [p 19]

Related concepts

Dataspaces [p 64]

Datasets [p 82]

CHAPTER **5**

Creating a data model

This chapter contains the following topics:

5.1 Creating a new data model

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

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 dataset, as explained in Permissions [p 97].

6.3 **Data model properties**

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

Dataset inheritance	Specifies whether dataset inheritance is enabled for this data model. Dataset inheritance is disabled by default. See Dataset inheritance [p 101] for more information.
Disable auto-increment checks	Specifies whether to disable if the check of an auto-incremented field value in associated datasets regarding to the "max value" found in the table being updated.

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:

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 Included data models [p 36] 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 39] 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 39] for more information.

Creating primary key fields

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

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

Creating or defining foreign key fields

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

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

Creating associations

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

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

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

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

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

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

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

7.4 **Modifying existing elements**

Removing a field from the primary key

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

See primary key [p 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 49]*

This chapter contains the following topics:

8.1 Basic element properties

Common basic properties

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

Information	Additional non-internationalized information associated with the element.
Minimum number of values	Minimum number of values for an element. As primary keys cannot be multi-valued, they must have this property set to '1' or 'undefined'. The minimum number of values is automatically set to '0' when defining the field as a selection node.
Maximum number of values	Maximum number of values for an element. When set to a value greater than '1', the element becomes multi-valued. As primary keys cannot be multi-valued, they must have this property set to '1' or 'undefined'. For tables, the maximum number of values is automatically set to 'unbounded' upon creation. The maximum number of values is automatically set to '0' when defining the field as a selection node.
Validation rules	This property is available for tables and fields in tables except `Password` fields, reusable types, fields in complex reusable types, and selection nodes. Used to define powerful and complex validation rules with the help of the provided XPath 1.0 criteria editor. 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.

Disable validation

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

Auto-increment

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

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

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 dataset.
***Presentation* > Record labeling**	Defines the fields to provide the default and localized labels for records in the table. **Attention:** Access rights defined on associated datasets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
***Presentation* > Default rendering for groups in forms**	Specifies the default display rendering mode of the groups contained in this table. If nothing is defined here, the default policy set in the Administration area will be used to display groups. **Enabled rendering for groups** Specifies a display rendering mode to be enabled for groups in the table in addition to the modes 'expanded' and 'collapsed', which are always available. Tabs must be enabled on the table to have the option to display groups as tabs. Similarly, links must be enabled to have the option to display groups as links. **Default rendering for groups** Specifies the default display rendering mode to use for the groups contained in this table. If a group does not specify a default mode then the default mode defined for this table will be used. Links must be enabled to define the default rendering mode as 'Link'. Select a rendering mode according to network and browser performance. Link mode is lighter as its content is not displayed on the same page, whereas the expanded and collapsed modes are heavier. **Note:** When the tabs view is enabled on a table, any groups that would otherwise have used links are automatically converted to collapsed mode. This is done to avoid the inherent display layout complexities that would arise from links and tabs coexisting in the same user interface.

***Presentation* > Specific rendering of forms**	Defines a specific rendering for customizing the record form in a dataset.
Toolbars	Defines the toolbars to use in this table. Toolbars can be edited in the *Configuration* > *Toolbars* section. **Tabular view top :** Defines the toolbar to use on the top of the default table view. **Tabular view row :** Defines the toolbar to use on each row of the default table view. **Record top :** Defines the toolbar to use in the record form. **Hierarchy top :** Defines the toolbar to use in the default hierarchy view of the table.
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. **Index name**: Unique name for this index. **Fields to index**: The fields to index, each represented by an absolute XPath notation starting under the table `root` element.
Specific filters	Defines record display filters on the table.
Actions	Specifies the actions that are allowed on the table in associated datasets. By default, all actions are allowed unless specific access rights are defined in a dataset.

Uniqueness constraints

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

Related concepts *Data validation controls on elements* *[p 49]*

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 43]*

This chapter contains the following topics:

9.1 **Simple content validation**

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

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

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 dataset by default. It can also reference a table in another dataset in the same dataspace, or a dataset in a different dataspace.
Mode	Location of the table referenced by the foreign key. 'Default': current data model. 'Other dataset': different dataset, in the same dataspace. 'Other dataspace': dataset in a different dataspace.
Referenced table	XPath expression describing the location of the table. For example, `/root/MyTable`.
Referenced dataset	Required if the table is located in another dataset. The unique name of the dataset containing the referenced table.
Referenced dataspace	Required if the table is located in another dataspace. The unique name of the dataspace containing the referenced table.
Label	Defines fields to provide the default and localized labels for records in the table. **Attention:** Access rights defined on associated datasets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
Filter	Defines a foreign key filter using an XPath expression.
Greater than [dynamic]	Defines a field to provide the minimum value allowed for this field.
Less than [dynamic]	Defines a field to provide the maximum value allowed for this field.

Fixed length [dynamic]	Defines a field to provide the exact number of characters required for this field.
Minimum length [dynamic]	Defines a field to provide the minimum number of characters allowed for this field.
Maximum length [dynamic]	Defines a field to provide the maximum number of characters allowed for this field.
Excluded values	Defines a list of values that are not allowed for this field.
Excluded segment	Defines an inclusive range of values that are not allowed for this field. **Minimum excluded value:** Lowest value not allowed for this field. **Maximum excluded value:** Highest value not allowed for this field.
Enumeration filled by another node	Defines the possible values of this enumeration using a reference to another list or enumeration element.
Dataspace set configuration	Define the dataspaces that can be referenced by a field of the type Dataspace identifier (osd:dataspaceKey). If a configuration is not set, then only opened branches can be referenced by this field by default. • Includes Specifies the dataspaces that can be referenced by this field. **Pattern:**Specifies a pattern that filters dataspaces. The pattern is checked against the name of the dataspaces. **Type:**Specifies the type of dataspaces that can be referenced by this field. If not defined, this restriction is applied to branches. **Include descendants:**Specifies if children or descendants of the dataspaces that match the specified pattern are included in the set. If not defined, this restriction is not applied to child dataspaces. If "None" then neither children nor descendants of the dataspaces that match the specified pattern are included. If "All descendants" then all descendants of the dataspaces that match the specified pattern are included. If "All descendant branches" then all descendant branches of the dataspaces that match the specified pattern are included. If "All descendant snapshots" then all descendant snapshots of the dataspaces that match the

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

- Excludes

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

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

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

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

	versions that are the direct children of the branches which are children of this version.
Dataset set configuration	Define the datasets that can be referenced by a field of the type Dataset identifier (osd:datasetName). • Includes Specifies the datasets that can be referenced by this field. **Pattern:**Specifies a pattern that filters datasets. The pattern is checked against the name of the datasets. **Include descendants:** Specifies if children or descendants of the datasets that match the specified pattern are included in the set. • Excludes Specifies the datasets that cannot be referenced by this field. Excludes are ignored if no includes are defined. **Pattern:** Specifies a pattern that filters datasets. The pattern is checked against the name of the datasets. **Include descendants:** Specifies if children or descendants of the datasets that match the specified pattern are included in the set.

Validation properties

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

Validation	Defines a localized validation message with a user-defined severity level.
Severity	Defines the severity level of the validation message. Possible values are 'Error', 'Warning', and 'Information'.
Error management policy	Specifies the behavior of the constraint when validation errors occur. It is possible to specify that the constraint must always remain valid after an operation (dataset update, record creation, update or deletion), or when a user submits a form. In this case, any input or operation that would violate the constraint will be rejected and the values will remain unchanged. If not specified, the constraint only blocks errors upon form submission by default, except for foreign key constraints in relational data models where errors are prevented for all operations by default. This option is only available upon static controls, exclude values, exclude segment and foreign key constraints. On foreign key constraints the error management policy does not concern filters. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case updates are not rejected and a validation error occurs. It is not possible to specify an error management policy on structural constraints that are defined in relational data models, when table history or replication is activated. That is, setting this property on fixed length, maximum length, maximum number of digits and decimal place constraints will raise an error at data model compilation because of the underlying RDBMS blocking constraints validation policy. This property is ineffective when importing archives. That is, all blocking constraints, excepted structural constraints, are always disabled when importing archives.
Message	Defines the message to display if the value of this field in a 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 43]*

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:

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

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

See Publishing data models [p 59] for more information on the publication process.

CHAPTER 11

Publishing a data model

This chapter contains the following topics:

11.1 About publications

Each dataset 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, datasets can be created based upon it.

> **Note**
>
> The **Publish** button is only displayed to users who have permission to publish the data model. See Data model permissions [p 35] for more information.

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

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 61] for more information on data model versions.

CHAPTER **12**

Versioning a data model

This chapter contains the following topics:

12.1 About versions

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

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.

Dataspaces

CHAPTER 13

Introduction to dataspaces

This chapter contains the following topics:

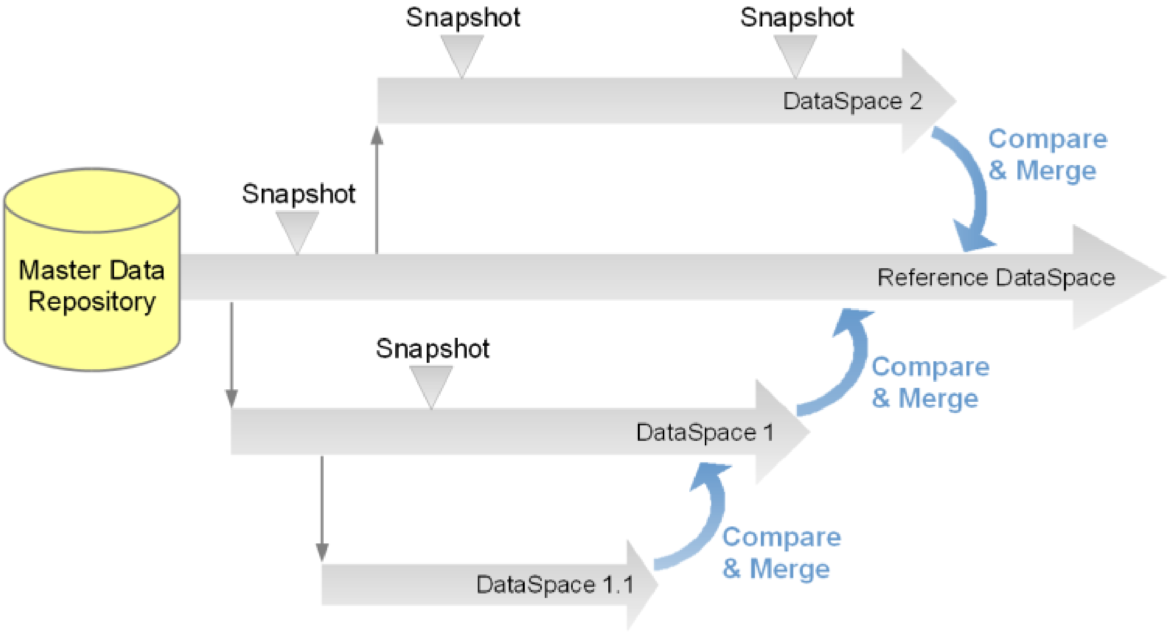
13.1 Overview

What is a dataspace?

The life cycle of data can be complex. It may be necessary to manage a current version of data while working on several concurrent updates that will be integrated in the future, including keeping a trace of various states along the way. In EBX5, this is made possible through the use of dataspaces and snapshots.

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

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



Basic concepts related to dataspaces

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

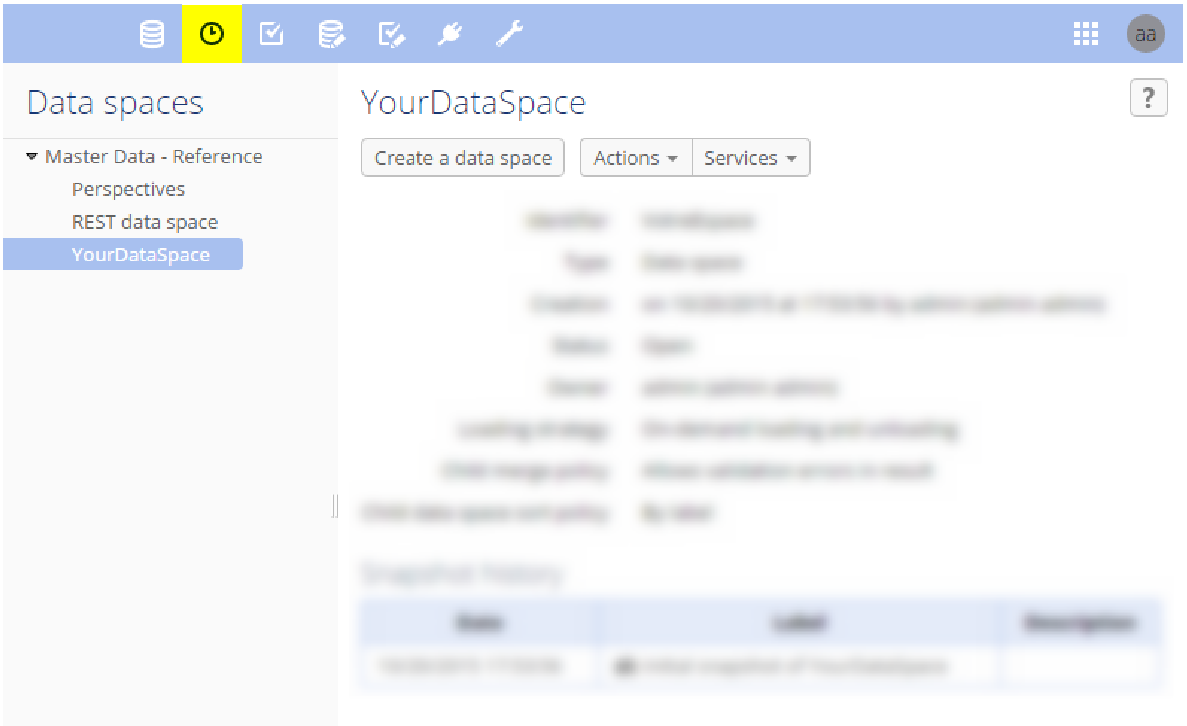
- dataspace [p 21]
- snapshot [p 22]
- dataset [p 20]
- dataspace merge [p 21]
- reference dataspace [p 21]

13.2 Using the Dataspaces area user interface

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

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

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



See also

Creating a dataspace [p 67]

Snapshots [p 77]

Related concepts *Datasets* [p 82]

CHAPTER 14

Creating a dataspace

This chapter contains the following topics:

14.1 Overview

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

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

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

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

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

See also *Relational mode* *[p 68]*

14.2 Properties

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

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

14.3 Relational mode

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

CHAPTER 15

Working with existing dataspaces

This chapter contains the following topics:

15.1 Dataspace information

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

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

15.2 **Dataspace permissions**

General permissions

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

Allowable actions

Users can be allowed to perform the following actions:

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

15.3 Merging a dataspace

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

1. Both the parent and child dataspaces are locked to all users, except the user who initiated the merge and administrator users. These locks remain for the duration of the merge operation. When locked, the contents of a 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 dataspaces cannot be merged until the merge in progress is finished.
2. Changes that were made in the child dataspace since its creation are integrated into its parent dataspace.
3. The child dataspace is closed.
4. The parent dataspace is unlocked.

Initiating a merge

To merge a dataspace into its parent dataspace:

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

Reviewing and accepting changes

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

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

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

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

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

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

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

Types of modifications

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

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

Types of conflicts

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

Conflicts are categorized as follows:

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

Finalizing a merge

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

Depending on the child merge policy that is configured on the parent 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 dataspace in your merge can set its child merge policy so that merges of its children are only finalized if the result would not contain any validation errors.

If, however, the administrator of the parent dataspace has set its child merge policy to 'Pre-validating merge', a dedicated dataspace is first created to hold the result of the merge. When the result is valid, this dedicated 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 dataspace and the dataspace 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 dataspace in its pre-merge state.
- **Continue** , which you can use to reattempt the merge after you have made the necessary corrections to the dedicated merge dataspace.

Setting the child merge policy of a dataspace

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

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

See also *Child merge policy* *[p 73]*

Abandoning a merge

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

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

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

15.4 Comparing a dataspace

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

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

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

15.5 **Validating a dataspace**

To perform a global validation of the contents of a dataspace, 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 dataset contained in the current dataspace.

15.6 **Dataspace archives**

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

Exporting

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

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

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

Importing

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

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

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

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

15.7 **Closing a dataspace**

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

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

To close a dataspace, select **Actions > Close this 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 dataspace. Snapshots exist as a record of the state and contents of a dataspace at a given point in time.

See also *Snapshot* *[p 22]*

16.2 Creating a snapshot

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

The following information is required:

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

16.3 Viewing snapshot contents

To view the contents of a snapshot, select the snapshot, then select **Actions** > **View datasets** 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 dataspace in the repository. To perform a comparison, select the snapshot, then select **Actions** > **Compare** from the workspace.

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

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

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 dataset 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.
Datasets to export	The datasets to export from this snapshot. For each dataset, you can choose whether to export its data values, permissions, and information.

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 dataspace in the Data Spaces area, nor can it be accessed.

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

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

Datasets

CHAPTER 17

Introduction to datasets

This chapter contains the following topics:

17.1 Overview

What is a dataset?

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

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

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

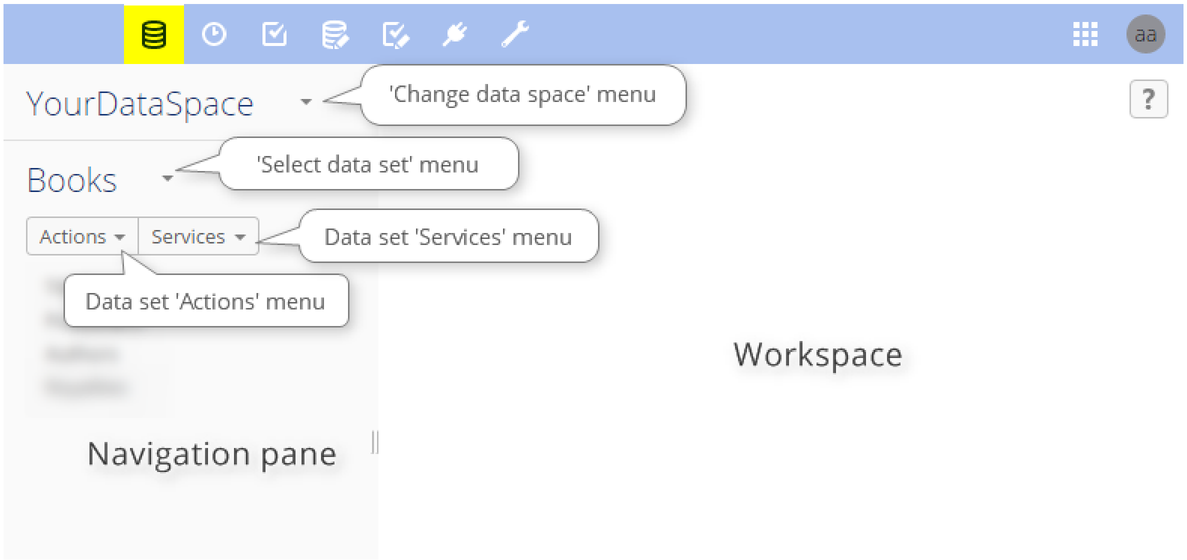
Basic concepts related to datasets

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

- dataspace [p 21]
- dataset [p 20]
- record [p 19]
- field [p 18]
- primary key [p 18]
- foreign key [p 18]
- table (in dataset) [p 20]
- group [p 19]

17.2 Using the Data user interface

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



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

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

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

See also

Creating a dataset [p 85]

Searching and filtering data [p 88]

Working with records in the user interface [p 95]

Inheritance [p 20]

Related concepts

Data model [p 28]

Dataspace [p 64]

CHAPTER **18**

Creating a dataset

This chapter contains the following topics:

18.1 Creating a root dataset

To create a new root dataset, that is, one that does not inherit from a parent dataset, select the '**Select dataset** [p 83]' ▼ menu in the navigation pane, click the '**Create a dataset**' button in the pop-up, and follow through the wizard.

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

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

- A *packaged data model* is a data model that is located within a module, which is a web application.
- An *embedded data model* is a data model that is managed entirely within the 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 dataset, you must provide a unique name, without spaces or special characters. Optionally, you may provide localized labels for the dataset, which will be displayed to users in the user interface depending on their language preferences.

Attention

Table contents are not copied when duplicating a data set.

18.2 **Creating an inheriting child dataset**

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

To create a child dataset, select the '**Select dataset** [p 83]' ▼ menu in the navigation pane, then click the + button next to the desired parent data set.

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

See also *Dataset inheritance [p 101]*

CHAPTER **19**

Viewing table data

EBX5 offers a customization mechanism for tables via the 'Views' feature. A view allows specifying which columns should be displayed as well as the display order. Views can be managed by profile thanks to the recommended views [p 92] concept.

This chapter contains the following topics:

19.1 'View' menu

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

Views are managed in a dedicated sub-menu: 'Manage views' [p 93].

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 90] 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, click on the 'Select and sort' button in the workspace.

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

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

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

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

It is possible to customize the display of tables in EBX5 according to the target user. There are two types of views: tabular [p 90] and hierarchical [p 90].

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

Two types of views can be created:

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

View description

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

Documentation	Localized label and description associated with the view.
Owner	Name of the owner of the view. This user can manage and modify it. (Only available for administrators and dataset owners)
Share with	Other profiles allowed to use this view from the 'View' menu.
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 87].
Filter	Defines filters for the records to be displayed in the table.
Pagination limit	Forces a limit to the number of visible records.
Grid edit	If enabled, users of this view can switch to grid edit, so that they can edit records directly from the tabular view.
Disable create and duplicate	If yes, users of this view cannot create nor duplicate records from the grid edit.

Hierarchical views

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

Hierarchy dimension

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

Hierarchical view configuration options

This form allows configuring the hierarchical view options.

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

Labels

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

Filter

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

Ordering field

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

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

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

Attention

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

Actions on hierarchy nodes

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

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

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

View sharing

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

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

View publication

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

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

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

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

Manage views

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

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

19.6 **Grid edit**

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

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

See also *Grid edit* *[p 90]*

Copy/paste

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

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

All simple type fields using built-in widgets are supported, except:

- foreign keys targeting non-string fields;
- non-string enumerates.

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 273]*

XML Services *[p 267]*

CHAPTER 21

Working with existing datasets

This chapter contains the following topics:

21.1 Validating a dataset

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

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

21.2 Duplicating a dataset

To duplicate an existing dataset, select it from the '**Select dataset** [p 83]' ▼ menu in the navigation pane, then select **Actions** > **Duplicate**.

21.3 Deactivating a dataset

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

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

21.4 Managing dataset permissions

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

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

See also *Profile* [p 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.
Dataset actions	Specifies the permissions for actions on the dataset.
Create a child dataset	Indicates whether the profile can create a child dataset. Inheritance also must be activated in the data model.
Duplicate the dataset	Indicates whether the profile can duplicate the dataset.
Delete the dataset	Indicates whether the profile can delete the dataset.
Activate/deactivate the dataset	Indicates whether the profile can modify the *Activated* property in the dataset information. See Deactivating a dataset [p 97].
Create a view	Indicates whether the profile can create views and hierarchies in the dataset.
Tables policy	Specifies the default permissions for all tables. Specific permissions can also be defined for a table by clicking the '+' button.
Create a new record	Indicates whether the profile can create records in the table.
Overwrite inherited record	Indicates whether the profile can override inherited records in the table. This permission is useful when using dataset inheritance.
Occult inherited record	Indicates whether the profile can occult inherited records in the table. This permission is useful when using dataset inheritance.
Delete a record	Indicates whether the profile can delete records in the table.
Values access policy	Specifies the default access permissions for all the nodes of the dataset and allows the definition of permissions for specific nodes. The default access permissions are used if no custom permissions have been defined for a node.

	The specific policy selector allows granting specific access permissions for a node. The links "ReadOnly", "ReadWrite", and "Hidden" set the corresponding access levels for the selected nodes. It is possible to remove custom access permissions using the "(default)" link.
Rights on services	This section specifies the access permissions for services. A service is not accessible to a profile if it is crossed-out.

CHAPTER **22**

Dataset inheritance

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

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

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

See also *Data model configuration* *[p 36]*

This chapter contains the following topics:

1. Dataset inheritance structure
2. Value inheritance

22.1 **Dataset inheritance structure**

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

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

22.2 **Value inheritance**

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

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

The ↳ button can be used to override a value.

Record inheritance

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

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

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

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

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

Workflow models

CHAPTER 23

Introduction to workflow models

This chapter contains the following topics:

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 23]

Data workflow (glossary) [p 24]

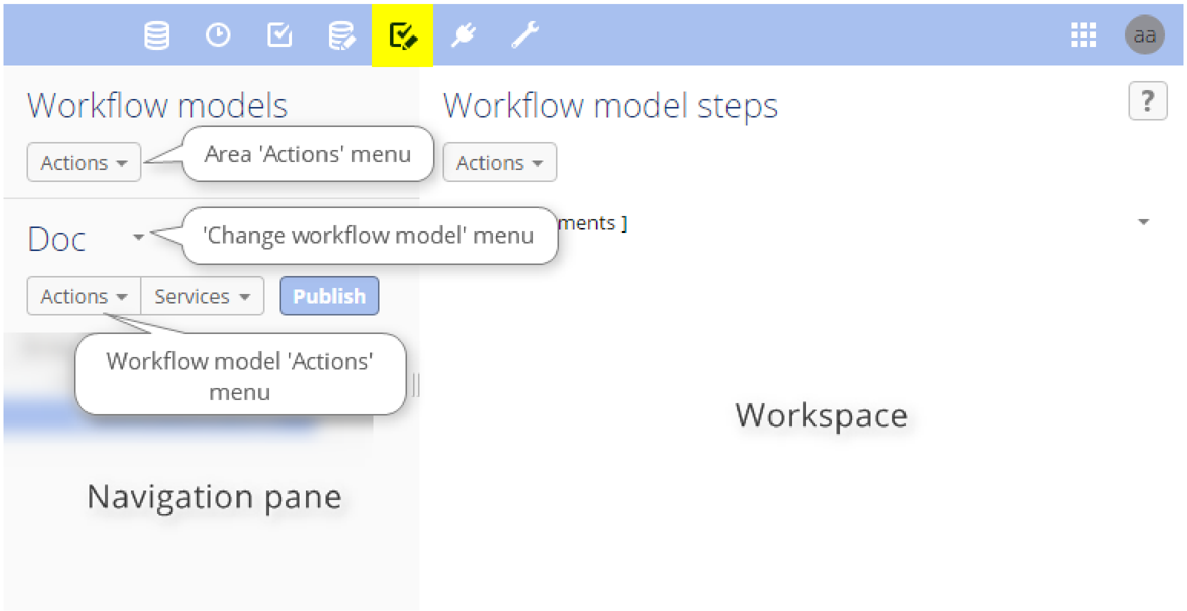
Basic concepts related to workflow models

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

- script task [p 23]
- 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.

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

When creating a new template, two fields are required:

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

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

system.time	System time of the repository.
system.date	System date of the repository.
workflow.lastComment	Last comment on the 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 130]*

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.

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

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

Hidden in graphical view

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

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

24.3 User tasks

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

Mode

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

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

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

List of profiles

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

[Default] Offered to the following profiles

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

[Legacy mode] Participants

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

Service

EBX5 includes the following built-in services:

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

See also *EBX5 built-in services* *[p 157]*

Configuration

Main options > Enable reject

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

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

Main options > Enable confirmation request

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

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

Main options > Enable comments

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

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

Main options > Comments required

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

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

Main options > Customized labels

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

[Legacy mode] Termination > Task termination criteria

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

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

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

[Legacy mode] Termination > Reject tolerance

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

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

The following task termination criteria automatically tolerate all rejections:

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

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 107]*

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:

- Create a dataspace
- Create a snapshot
- Merge a dataspace
- Import an archive
- Close a dataspace
- Set a data context variable

- 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:

- Dataspace modified?
- Data valid?
- Last user task accepted?
- Value null or empty?
- Values 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 107] 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. See Generic message templates [p 107].
Notification of completion	The list of profiles to which to send notifications, based on a template, when a data workflow is completed. The notification is only sent if the workflow has been completed under normal circumstances, that is, not due to an administration action. See Generic message templates [p 107].
Notification of error	The list of profiles that will receive notifications, based on a template, when a data workflow is in error state. See Generic message templates [p 107].
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 139] for more information. **Note:** Only users who are defined as workflow administrators will be able to manually modify the priority level of any associated data workflows.
Activate quick launch	By default, when a workflow is launched, the user is prompted to enter a documentation for the new workflow in an intermediate form. This documentation is optional. Setting the 'Activate quick launch' property to 'Yes' allows skipping this documentation step and proceeding directly to the workflow launch.
Automatically open the first step	Allows determining the navigation after a workflow is launched. By default, once a workflow is launched, the current table (workflow launchers or monitoring > publications) is automatically displayed. Enabling this property will allow the workflow user to keep working on the launched workflow. If, after the first workflow step is executed, a work item is reached, and this work item can be started by the workflow creator, then the

	work item is automatically opened (if several work items are reached, the first created is opened). This will save the user from selecting the corresponding work item from the work items inbox. If no work item has been reached, the next step progress strategy is evaluated. If no work item has been opened, the table from which the workflow has been launched is displayed. **Limitation:** This property will be ignored if the first step is a sub-workflow invocation.
Workflow trigger	Component that intercepts the main events of a workflow.
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 **Data context**

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

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

25.4 **Custom workflow execution views**

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

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

25.5 Permissions on associated data workflows

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

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

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

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

See also *Workflow administration* *[p 143]*

25.6 Workflow model snapshots

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

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

25.7 Deleting a workflow model

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

See also *Publishing workflow models* *[p 127]*

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 [p 124]*

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:

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 135]*

Launching and monitoring data workflows *[p 141]*

Administration of data workflows *[p 143]*

Permissions on associated data workflows *[p 122]*

Related concepts *Workflow models* *[p 106]*

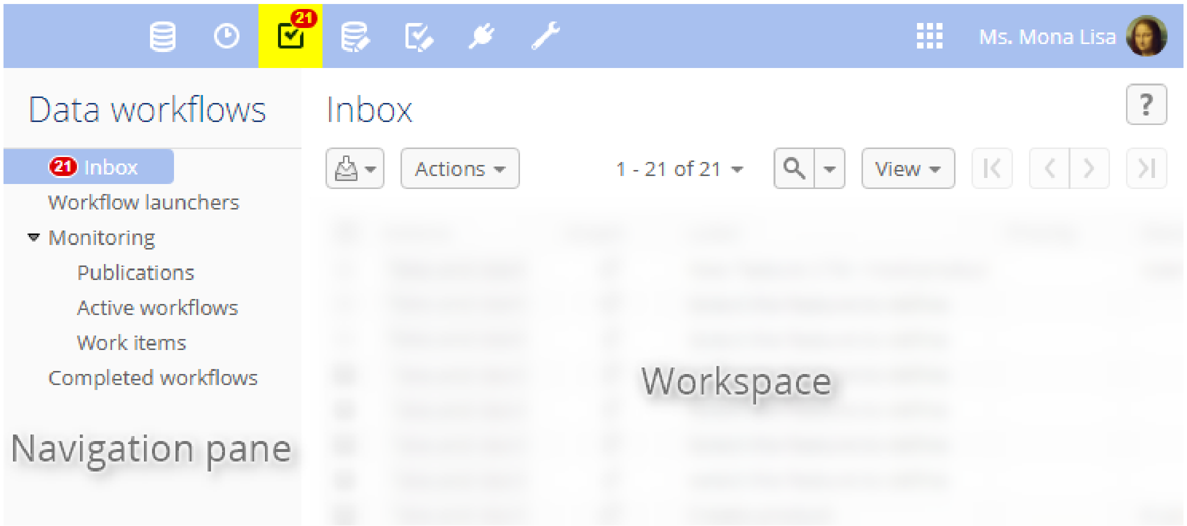
CHAPTER 28

Using the Data Workflows area user interface

This chapter contains the following topics:

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 112] 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 120] in workflow modeling.

28.3 Custom views

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

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

See the Views [p 89] for more information.

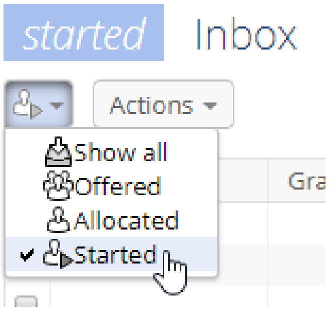
28.4 Specific columns

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

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

28.5 Filtering items in views

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



28.6 Graphical workflow view

Whether as a user with a work item to perform, or as a data workflow monitor or administrator, you can view the progress or the history of a data workflow execution by clicking the 'Preview' button that appears in the 'Data workflow' column of tables throughout the data workflows user interface.

This opens a pop-up displaying an interactive graphical view of the data workflow's execution. In this view, you can see the overall progress of the execution, as well as click on an individual step to view the details of its information.

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

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

CHAPTER 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 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.

Creation of work items

Default mode

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

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

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

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

Legacy mode

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

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

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

Variations of the work item states

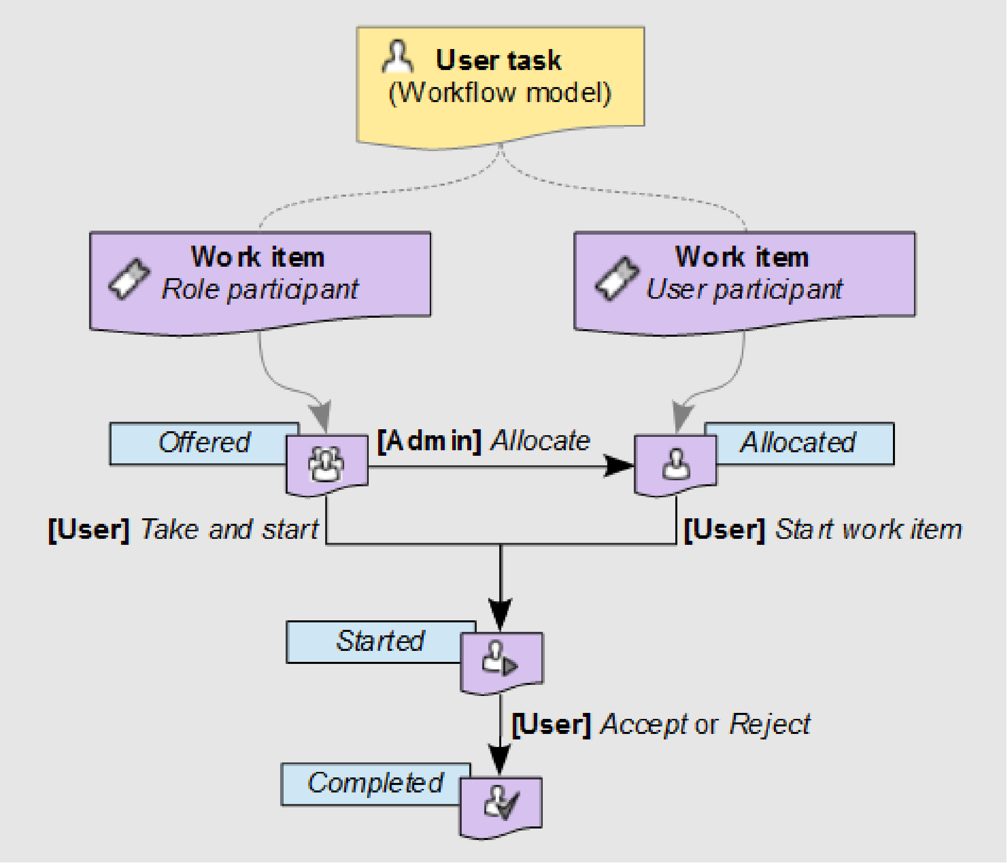
When the work item is in the *allocated* state, the defined user can directly start working on the allocated work item with the 'Take and start' action. The work item's state becomes *started*.

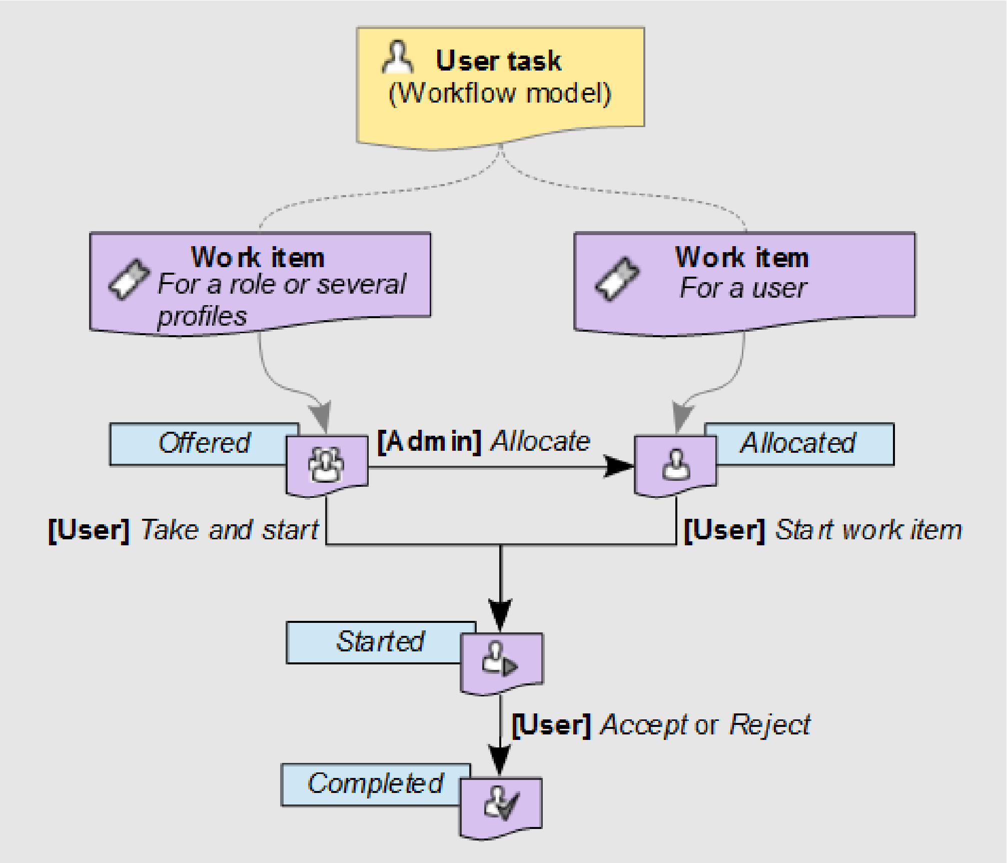
When the work item is in the *offered* state, any user or member of the roles to whom the work item is offered can take the work item with the 'Take and start' action'. The work item's state becomes *started*.

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

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

Diagram of the work item states





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 112]*

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 135]*

Permissions on associated data workflows *[p 122]*

Related concepts *Workflow models* *[p 106]*

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:

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 Disabling a workflow publication [p 144].
2. Unpublish the workflow publication by selecting **Actions** > **Unpublish** from the workflow publication's entry in the same table of publications.

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

Actions on data workflows

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

Replaying a step

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

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

Terminating and cleaning an active data workflow

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

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

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

Forcing termination of an active data workflow

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

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

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

Forcing resumption of a waiting data workflow

In order to resume a data workflow that is currently waiting for an event, select *Actions > Force resumption* from the entry of the workflow in the 'Active workflows' table. This will resume the data workflow. Before doing this action, it is the 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, you can perform a clean operation on it. To do so, select **Actions > Clean** from the entry of the workflow in the 'Completed workflows' table.

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

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

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 122]*

Data services

CHAPTER **32**

Introduction to data services

This chapter contains the following topics:

32.1 Overview

What is a data service?

A data service [p 24] is:

- a standard Web service that interacts with EBX5.

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

- a RESTful operation that allows interrogating the EBX5 repository.

  The RESTful API does not require a service interface, it is self-descriptive through the meta-data returned.

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

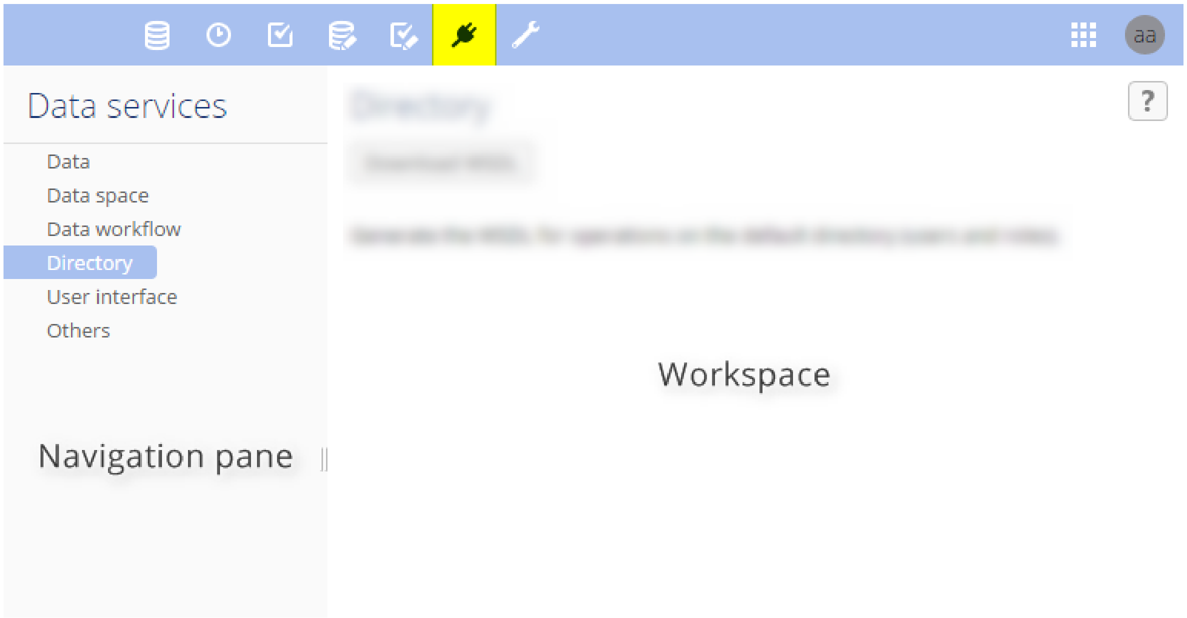
Lineage

Lineage [p 25] 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

Dataspace [p 64]

Dataset [p 82]

Data workflows [p 130]

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

Operations on datasets

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

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

Operations on tables

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

- Insert record(s)
- Select record(s)
- Update record(s)
- Delete record(s)

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

33.2 Generating a WSDL for dataspace operations

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

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

Operations on dataspaces

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

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

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 WSDL for operations on data [p 151].

Steps for generating the WSDL for lineage are as follows:

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

See also *Lineage [p 148]*

Reference Manual

Integration

CHAPTER 34

Built-in user services

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

- when defining workflow model tasks [p 112]
- when defining perspective action menu items [p 12]

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

This chapter contains the following topics:

34.1 Access data (default service)

By default, workflows automatically consider this service as complete. That is, the 'Accept' button is always available.

This is the default service used if no service is specified.

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`disableAutoComplete`	Disable Accept at start	By default, the interaction associated with this service is directly considered as complete. Therefore, the Accept button is automatically displayed at the opening of the work item. This parameter is useful to disable this behavior. If the value is 'true', the developer will be in charge of completing the interaction by using SessionInteraction in a user service or a trigger, for example. The default value is 'false'. Perspectives do not use this parameter.
`firstCallDisplay`	First call display mode	Defines the display mode that must be used when displaying a filtered table or a record upon first call. Default (value = 'auto'): the display is automatically set according to the selection. View (value = 'view'): forces the display of the tabular view or of the hierarchical view. Record (value = 'record'): if the predicate has at least one record, forces the display of the record form.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.
`viewPublication`	View	The publication name of the view to display. The view must be configured for the selected table.
`xpath`	Dataset node (XPath)	The value must be a valid absolute location path in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax.

34.2 **Create a new record**

For a workflow, the creation service is considered complete when the first successful submit is performed (record has been created). If this service is called whereas it is already complete, the created record is displayed in update or read-only mode (depending on the user rights).

Service name parameter: `service=@creation`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - This field is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Dataset table (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Output parameters

Parameter	Label	Description
`created`	Created record	Contains the XPath of the created record.

34.3 **Duplicate a record**

For a workflow, the duplicate service is considered complete when the first successful submit is performed (record has been created). If this service is called whereas it is already complete, the created record is displayed in update or read-only mode (depending on the user rights).

Service name parameter: `service=@duplicate`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - This field is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Record to duplicate (XPath)	The value must be a valid absolute location path of an existing record. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Output parameters

Parameter	Label	Description
`created`	Created record	Contains the XPath of the created record.

34.4 Export data to an XML file

The exportToXML service is considered complete when export is done and file downloaded.

Service name parameter: `service=@exportToXML`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.
`xpath`	Dataset table to export (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

34.5 Export data to a CSV file

Workflows consider the exportToCSV service as complete when export is done and file downloaded.

Service name parameter: `service=@exportToCSV`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.
`xpath`	Dataset table to export (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

34.6 **Import data from an XML file**

Workflows consider the importFromXML service as complete when import is performed.

Service name parameter: `service=@importFromXML`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Dataset table to import (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

34.7 **Import data from a CSV file**

Workflows consider the importFromCSV service as complete when import is performed.

Service name parameter: `service=@importFromCSV`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace - This field is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`xpath`	Dataset table to import (XPath)	The value must be a valid absolute location path of a table in the selected dataset. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

34.8 **Access a dataspace**

A workflow automatically considers that the dataspace selection service is complete.

Service name parameter: `service=@selectDataSpace`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace is required for this service.

34.9 **Validate a dataspace, a snapshot or a dataset**

Workflows automatically consider the validation service as complete.

Service name parameter: `service=@validation`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace or snapshot is required for this service.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace or snapshot is required for this service.

Output parameters

Parameter	Label	Description
`hasError`	Found errors	Contains 'true' if validation has produced errors.
`hasFatal`	Found fatal errors	Contains 'true' if validation has produced fatal errors.
`hasInfo`	Found informations	Contains 'true' if validation has produced informations.
`hasWarning`	Found warnings	Contains 'true' if validation has produced warnings.

34.10 Merge a dataspace

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

Service name parameter: `service=@merge`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.

Output parameters

Parameter	Label	Description
`mergeResult`	Merge success	Contains 'true' if merge succeeded, otherwise 'false'.
`mergeState`	Merge state	Contains the return code of the merge. It is strongly recommended to parse this value by using the InteractionMergeState UIHttpManagerComponentReturnCode.

34.11 **Access the dataspace merge view**

The merge.view service is automatically considered complete.

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

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace is required for this service.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.

34.12 **Compare contents**

Workflows automatically consider the compare service as complete.

Service name parameter: `service=@compare`

Input parameters

Parameter	Label	Description
`branch`	Dataspace	The identifier of the specified dataspace - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`compare.branch`	Dataspace to compare	The identifier of the dataspace to compare - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`compare.filter`	Comparison filter	To ignore inheritance and computed values fields in the comparison (disable resolved mode), the filter "persistedValuesOnly" must be specified. By default, when no filter is defined, the comparison uses resolved mode.
`compare.instance`	Dataset to compare	The value must be the reference of a dataset that exists in the selected dataspace to compare.
`compare.version`	Snapshot to compare	The identifier of the snapshot to compare - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`compare.xpath`	Table or record to compare (XPath)	The value must be a valid absolute location path of a table or a record in the selected dataset to compare. The notation must conform to a simplified XPath, in its abbreviated syntax.
`instance`	Dataset	The value must be the reference of a dataset that exists in the selected dataspace.
`scope`	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session. If left blank, the default value will be used. For perspectives, the default value is always 'node'. For workflows, the default value depends on the selected entities or service.
`trackingInfo`	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
`version`	Snapshot	The identifier of the specified snapshot - A dataspace or snapshot and a dataspace or snapshot to compare to are required for this service.
`xpath`	Table or record (XPath)	The value must be a valid absolute location path of a table or a record in the selected dataset. The notation must

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

34.13 Data workflows

This service provides access to the data workflows user interfaces.

Service name parameter: `service=@workflow`

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

Input parameters

Parameter	Label	Description
`scope`	Scope	Defines the scope of the user navigation for this service.
`viewPublication`	View publication	Defines the publication name of the view to apply for this service.
`workflowView`	Workflow view	Specifies the workflow view type. Value can be one of the following: "inbox", "launcher", "monitoringPublications", "monitoringWorkflows", "monitoringWorkItems" or "completedWorkflows".
`xpath`	Filter (XPath)	An optional filter. The syntax should conform to an XPath predicate surrounded by "[" and "]".

CHAPTER 35

Introduction

This chapter contains the following topics:

35.1 Overview of data services

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

In order to invoke SOAP operations [p 189], for an integration use case, a WSDL [p 181] must be generated from a data model. To invoke RESTful operations, for integration, mobile and web interface use cases, it is not mandatory to generate an interface, the request response is self-descriptive, the syntax is described in the chapter RESTful operations [p 217]. It will be possible to perform operations such as:

- Selecting, inserting, updating, deleting, or counting records
- Selecting or counting history records
- Selecting dataset values
- Getting the differences on a table between dataspaces or snapshots, or between two datasets based on the same data model
- Getting the credentials of records

Other generic WSDLs can be generated and allow performing operations such as:

- Creating, merging, or closing a dataspace
- Creating or closing a snapshot
- Validating a dataset, dataspace, or a snapshot

- Starting, resuming or ending a data workflow

> **Note**
>
> See SOAP and RESTful comparative [p 179].

35.2 Interactions

Input and output message encoding

All input messages must be *exclusively* in UTF-8. All output messages are in UTF-8.

Tracking information

Depending on the data services operation being called, it may be possible to specify session tracking information.

- Example for a SOAP operation, the request header contains:

```
<SOAP-ENV:Header>
 <!-- optional security header here -->
 <m:session xmlns:m="urn:ebx-schemas:dataservices_1.0">
  <trackingInformation>String</trackingInformation>
 </m:session>
</SOAP-ENV:Header>
```

- Example for a RESTful operation, the JSON request contains:

```
{
 "procedureContext": // JSON Object (optional)
 {
    "trackingInformation": "String" // JSON String (optional)
 }, ...
}
```

Session parameters

Depending on the data services operation being called, it is possible to specify session input parameters. They are defined in the request body.

- Example for a SOAP operation, the optional request header contains:

```
<SOAP-ENV:Header>
 <!-- optional security header here -->
 <m:session xmlns:m="urn:ebx-schemas:dataservices_1.0">
  <!-- optional trackingInformation header here -->
  <inputParameters>
   <parameter>
    <name>String</name>
    <value>String</value>
   </parameter>
   <!-- for some other parameters, copy complex
        element 'parameter' -->
  </inputParameters>
 </m:session>
</SOAP-ENV:Header>
```

- Example for a RESTful operation, the JSON request contains:

```
{
 "procedureContext": // JSON Object (optional)
 {
    "trackingInformation": "String", // JSON String (optional)
    "inputParameters": // JSON Array (optional)
    [
     // JSON Object for each parameter
     {
      "name" : "String" // JSON String (required)
      "value" : "String" // JSON String (optional)
      },
      ...
     ]
 }, ...
}
```

Exception handling

In case of unexpected server error upon execution of:

- A SOAP operation, a SOAP exception response is returned to the caller via the `soap:Fault` element. For example:

```
<soapenv:Fault>
 <faultcode>soapenv:java.lang.IllegalArgumentException</faultcode>
 <faultstring />
 <faultactor>admin</faultactor>
 <detail>
  <m:StandardException xmlns:m="urn:ebx-schemas:dataservices_1.0">
   <code>java.lang.IllegalArgumentException</code>
   <label/>
   <description>java.lang.IllegalArgumentException:
    Parent home not found at
    com.orchestranetworks.XX.YY.ZZ.AA.BB(AA.java:44) at
    com.orchestranetworks.XX.YY.ZZ.CC.DD(CC.java:40) ...
   </description>
  </m:StandardException>
 </detail>
</soapenv:Fault>
```

- A RESTful operation, a JSON exception response is returned to the caller. For example:

```
{
  "code": 999, // JSON Number, HTTP status code
  "errors": [
   {
    "severity": "...",     // JSON String, severity (optional)
    "rowIndex": 999,       // JSON Number, request row index (optional)
    "message": "...",      // JSON String, message
    "details": "...",      // JSON String, URL (optional)
    "pathInRecord": "...", // JSON String, Path in record (optional)
    "pathInDataset": "..." // JSON String, Path in dataset (optional)
   }
  ]
}
```

  The response contains an HTTP status code and a table of errors. The severity of each error is specified by a character, with one of the possible values (`F`: fatal, `E`: error, `W`: warning, `I`: information).

  See also *HTTP codes [p 221]*

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).

- 'Token Authentication Scheme' method is based on the HTTP-Header `Authorization`, as described in RFC 2324.

  Tracking information [p 172] and Session parameters [p 172] must be defined during the token creation, otherwise a `400 (Bad request)` error is returned.

  For more information on this authentication scheme, see Token authentication operations [p 222].

- 'Basic Authentication Scheme' method is based on the HTTP-Header `Authorization` in base 64 encoding, 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==
  ```

- 'Standard Authentication Scheme' is based on the HTTP `Request`. User and password are extracted from request parameters. For more information on request parameters, see Request parameters [p 184] section.

- The 'SOAP Security Header Authentication Scheme' method is based on the Web Services Security UsernameToken Profile 1.0 specification.

  By default, the type `PasswordText` is supported. This is done with the following SOAP-Header defined in the WSDL:

  ```
  <SOAP-ENV:Header>
   <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
    <wsse:UsernameToken>
     <wsse:Username>String</wsse:Username>
     <wsse:Password Type="wsse:PasswordText">String</wsse:Password>
    </wsse:UsernameToken>
   </wsse:Security>
  </SOAP-ENV:Header>
  ```

  > **Note**
  >
  > Only available for SOAP operations [p 189].

- The 'SOAP Specific Header Authentication Scheme'.

  For more information, see Overriding the SOAP security header [p 174].

- The 'REST Forward Authentication Scheme' method is based on the HTTP `Request` and reuses the current authenticated session.

  Tracking information [p 172] or Session parameters [p 172] are retrieved from the session and must not be defined on the request, otherwise a `400 (Bad request)` error is returned.

  > **Note**
  >
  > Only available for the RESTful operations [p 217].

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 SOAP operations [p 189].

Lookup mechanism

Because EBX5 offers several authentication methods, a lookup mechanism based on conditions was set to know which method should be applied for a given request. The method application conditions are evaluated according to the authentication scheme priority. If the conditions are not satisfied, the server evaluates the next method. The following table presents the available authentication methods for each supported protocol and their application conditions. They are ordered from the highest priority to the lowest.

Operation / Protocol	Authentication methods and application conditions
SOAP / JMS	SOAP Security Header [p 174] • The SOAP request is received over the JMS protocol. • The SOAP header content must contains a `Security` element. SOAP Specific Header [p 174] • The SOAP request is received over the JMS protocol. • The SOAP header content must not contain a `Security` element.
SOAP / HTTP	Basic [p 174] • The HTTP request must hold an `Authorization` header. • `Authorization` header value must start with the word `Basic`. • No `login` is provided in the URL parameters. Standard [p 174] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters. SOAP Security Header [p 174] • The SOAP header content must contain a `Security` element. • The HTTP request must not hold an `Authorization` header. • No `login` is provided in the URL parameters. SOAP Specific Header [p 174] • The SOAP header content must not contain a `Security` element. • The HTTP request must not hold an `Authorization` header. • No `login` is provided in the URL parameters.
WSDL / HTTP	Basic [p 174] • The HTTP request must not hold an `Authorization` header. • `Authorization` header value must start with the word `Basic`. • No `login` is provided in the URL parameters. Standard [p 174] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters.
REST / HTTP	Token [p 173] • The HTTP request must hold an `Authorization` header. • `Authorization` header value must start with the word `EBX`. • No `login` is provided in the URL parameters. Basic [p 174] • The HTTP request must hold an `Authorization` header. • `Authorization` header value must start with the word `Basic`. • No `login` is provided in the URL parameters. Standard [p 174] • The HTTP request must not hold an `Authorization` header.

Operation / Protocol	**Authentication methods and application conditions**
	- A `login` and a `password` are provided in the URL parameters. Rest forward [p 174] - The HTTP request must not contain an `Authorization` header. - No `login` is provided in the URL parameters. - An `onwbpID` URL parameter and a session cookie must be provided in the HTTP request.

In case of multiple authentication methods present in the same request, EBX5 will return an HTTP code `401 Unauthorized`.

35.4 SOAP and RESTful comparative

Operations	**SOAP**	**REST**
Data		
Select or count records (with filter and/or view publication)	X	X
Selector for possible enumeration values (with filter)		X
Insert, update or delete records	X	X
Select or count history records (with filter and/or view publication)		X
Select node values from dataset	X	X
Update node value from dataset		X
Get table or dataset changes between dataspaces or snapshots	X	
Get credentials for records	X	
Dataspaces		
Create, close, merge a dataspace	X	
Create, close a snapshot	X	
Validate a dataspace or a snapshot	X	
Validate a dataset	X	
Locking a dataspace	X	
Workflow		
Start, resume or end a workflow	X	

35.5 Limitations

Date, time & dateTime format

Data services only support the following date and time formats:

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

SOAP naming convention

Due to the naming convention of the data service operations, each table defined within a data model must have a unique name for the WSDL generation.

CHAPTER 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 dataset and replication operations.
`directory`	WSDL for default EBX5 directory operations. It is also possible to filter data using the `tablePaths` [p 185] or `operations` [p 185] parameters.
`repository`	WSDL for dataspace or snapshot management operations.
`tables`	WSDL for operations on the tables of a specific dataset.
`userInterface`	Deprecated since version 5.8.1. This operation type has been replaced by `administration`. While the user interface management operations are still available for backward compatibility reasons, it is recommended to no longer use this type. WSDL for user interface management operations (these operations can only be accessed by administrators).
`administration`	WSDL for administration operations like: • user interface management 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

A WSDL can only be downloaded by authorized profiles:

Operation type	Access right permissions
`custom`	All profiles, if at least one web service is registered.
`dataset`	All profiles.
`directory`	All profiles, if the following conditions are valid: • No specific directory implementation is used. (The built-in Administrator role is only subject to this condition). • Global access permissions are defined for the administration. • 'Directory' dataset permissions have writing access for the current profile.
`repository`	All profiles.
`tables`	All profiles.
`userInterface`	Deprecated since version 5.8.1. This operation type has been replaced by `administration`. While the user interface management operations are still available for backward compatibility reasons, it is recommended to no longer use this type. Built-in administrator role or delegated administrator profiles, if all conditions are valid: • Global access permissions are defined for the administration. • 'User interface' dataset permissions have writing access for the current profile.
`administration`	Built-in administrator role or delegated administrator profiles, if all conditions are valid: • Global access permissions are defined for the administration. • 'Administration' dataset permissions have write access for the current profile.
`workflow`	All profiles.

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 a WSDL for dataspace operations [p 152] 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 184]. An HTTP code is always returned, errors are indicated by an error code above 400.

- HTTP(S) response status codes:

Status code	Information
`200` *(OK)*	The WSDL content was successfully generated and is returned by the request (optionally in an attachment [p 186]).
`400` *(Bad request)*	The request is incorrect. This occurs when: • A request element is incorrect. • The unicity check on table names contains at least one error.
`401` *(Unauthorized)*	Request requires an authenticated user.
`403` *(Forbidden)*	Request is not allowed for the authenticated user.
`405` *(Method not allowed)*	Request is not allowed in this configuration.
`500` *(Internal error)*	Request generates an error (a stack trace and a detailed error message are returned).

> **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). *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 parameter is limited to a value containing 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. Empty value.
`login`	no	yes	no	A user identifier. Required when the standard authentication method is used. String type value.
`password`	no	yes	no	A password. Required when the standard authentication method is used. String type value.
`type`	yes	no	yes	An operation type [p 182]. Possible values are: `custom`, `dataset`, `directory`, `administration`, `userInterface`, `repository`, `tables` or `workflow`. String type value.
`branch` `version`	yes	yes	(*)	A dataspace or a snapshot identifier. (*) required for `tables` and `dataset` types, otherwise ignored. String type value.
`instance`	yes	yes	(*)	A dataset identifier. String type value.
`tablePaths`	no	yes	no	A list of table paths. Optional for `tables` or `directory` types, otherwise ignored. If not defined, all tables are selected. Each table path is separated by a comma character. String type value.
`operations`	no	yes	no	Allows generating a WSDL for a subset of operations. Optional for `tables` or `directory` operation types, otherwise ignored. If not defined, all operations for the given type are generated. This parameter's value is a concatenation of one or more of the following characters: • C = Count record(s) • D = Delete record(s) • E = Get credentials • G = Get changes

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

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 dataset `'dataset1'` in dataspace `'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 dataset `'dataset1'` in dataspace `'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 dataspace `'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

SOAP 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, that defines its operations. When using this WSDL, it will be possible to read and/or write in the EBX5 repository. For example, for a table located at the path `/root/XX/exampleTable`, the generated requests would follow the structure of its underlying data model and include the name of the table `<m:{operation}_exampleTable xmlns:m="urn:ebx-schemas:dataservices_1.0">`.

> **Attention**
>
> Since the WSDL and the SOAP operations tightly depend on the data model structure, it is important to redistribute the up-to-date WSDL after any data model change.

Content policy

Access to the content of records, the presence or absence of XML elements, depend on the 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 dataspace to which the dataset belongs.	Either this parameter or the 'version' parameter must be defined. Required for the 'insert', 'update' and 'delete' operations.
version	The identifier of the snapshot to which the dataset belongs.	Either this parameter or the 'branch' parameter must be defined
instance	The unique name of the dataset which contains the table to query.	Yes
predicate	XPath predicate [p 279] 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 267].	Only required for the insert and update operations
viewPublication	This parameter can be combined with the predicate [p 191] parameter as a logical `AND` operation. The behavior of this parameter is described in the section . It cannot be used if the 'viewId' parameter is used, and cannot be used on hierarchical views.	No
viewId	*Deprecated since version 5.2.3*. This parameter has been replaced by the parameter 'viewPublication'. While it remains available for backward compatibility, it will eventually be removed in a future version. This parameter cannot be used if the 'viewPublication' parameter is used.	No
blockingConstraintsDisabled	This property is available for all table updates data service operations. If `true`, the validation process disables blocking constraints defined in the data model. If this parameter is not present, the default is `false`.	No
details	The `details` element specifies the following option: The optional attribute `locale` (default 'en-US') defines the language of the blockingConstraintsDisabled [p 191] parameter in which the validation messages must be returned.	No

Select operations

Select request on table

```
<m:select_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <viewPublication>String</viewPublication>
 <exportCredentials>boolean</exportCredentials>
 <pagination>
  <previousPageLastRecordPredicate>String</previousPageLastRecordPredicate>
  <pageSize>Integer</pageSize>
 </pagination>
</m:select_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 191].	
version	See the description under Common parameters [p 191].	
instance	See the description under Common parameters [p 191].	
predicate	See the description under Common parameters [p 191]. This parameter can be combined with the viewPublication [p 191] parameter as a logical `AND` operation.	
viewPublication	See the description under Common parameters [p 191].	

includesTechnicalData	The response will contain technical data if `true`. See also the optimistic locking [p 206] section.
	Each returned record will contain additional attributes for this technical information, for instance:
	`...<table`
	`ebxd:lastTime="2010-06-28T10:10:31.046"`
	` ebxd:lastUser="Uadmin"`
	`ebxd:uuid="9E7D0530-828C-11DF-B733-0012D01B6E76">... .`
		No

exportCredentials	If `true` the select will also return the credentials for each record.	No
pagination	Enables pagination, see child elements below.	No
pageSize (nested under the `pagination` element)	When pagination is enabled, defines the number of records to retrieve.	When pagination is enabled, yes
previousPageLastRecordPredicate (nested under the `pagination` element)	When pagination is enabled, XPath predicate that defines the record after which the page must fetched, this value is provided by the previous response, as the element `lastRecordPredicate`. If the passed record is not found, the first page will be returned.	No

Select response on table

```
<ns1:select_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <data>
  <XX>
   <TableName>
    <a>key1</a>
    <b>valueb</b>
    <c>1</c>
    <d>1</d>
   </TableName>
  </XX>
 </data>
 <credentials>
  <XX>
   <TableName predicate="./a='key1'">
    <a>W</a>
    <b>W</b>
    <c>W</c>
    <d>W</d>
   </TableName>
  </XX>
 </credentials>
 <lastRecordPredicate>./a='key1'</lastRecordPredicate>
</ns1:select_{TableName}Response>
```

with:

Element	Description
data	Content of records that are displayed following the table path.
credentials	Contains the access right for each node of each record.
lastRecordPredicate	Only returned if the pagination is enabled, this defines the last records in order to be used on the next request in the element `previousPageLastRecordPredicated`.

See also the optimistic locking [p 206] section.

Select request on dataset

This operation returns dataset content without table.

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 191].	
version	See the description under Common parameters [p 191].	
instance	See the description under Common parameters [p 191].	

Select response on dataset

```
<ns1:selectInstanceResponse xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
```

```
 <data>
  <settings>
   <XX>
    <a>key1</a>
    <b>valueb</b>
    <c>1</c>
    <d>true</d>
   </XX>
  </settings>
 </data>
</ns1:selectInstanceResponse>
```

with:

Element	Description
data	Dataset content without table.

Delete operation

Delete request

```
<m:delete_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <includeOcculting>boolean</includeOcculting>
 <inheritIfInOccultingMode>boolean</inheritIfInOccultingMode>
 <checkNotChangedSinceLastTime>dateTime</checkNotChangedSinceLastTime>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
</m:delete_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 191].	
instance	See the description under Common parameters [p 191].	
predicate	See the description under Common parameters [p 191].	
includeOcculting	Includes the records in occulting mode. Default value is `false`.	No
inheritIfInOccultingMode	Inherits the record from its parent if it is in occulting mode. Default value is `false`.	No
occultIfInherit	*Deprecated since version 5.7.0* Occults the record if it is in inherit mode. Default value is `false`.	No
checkNotChangedSinceLastTime	Timestamp used to ensure that the record has not been modified since the last read. Also see the optimistic locking [p 206] section.	No
blockingConstraintsDisabled	See the description under Common parameters [p 191].	
details	See the description under Common parameters [p 191].	

Delete response

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

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. '95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted.
blockingConstraintMessage	This element is present if the status is equal to '95' with a localized message. The locale of the message is retrieved from the request parameter or from the user session.

Count operation

Count request

```
<m:count_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
</m:count_{TableName}>
```

with:

Element	Description
branch	See the description under Common parameters [p 191].
version	See the description under Common parameters [p 191].
instance	See the description under Common parameters [p 191].
predicate	See the description under Common parameters [p 191].

Count response

```
<ns1:count_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <count>Integer</count>
</ns1:count_{TableName}Response>
```

with:

Element	Description
count	The number of records that correspond to the predicate in the request.

Update operation

Update request

```
<m:update_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <updateOrInsert>boolean</updateOrInsert>
 <byDelta>boolean</byDelta>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
 <data>
  <XX>
   <TableName>
    <a>String</a>
    <b>String</b>
    <c>String</c>
    <d>String</d>
    ...
   </TableName>
  </XX>
 </data>
</m:update_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 191].	
instance	See the description under Common parameters [p 191].	
updateOrInsert	If `true` and the record does not currently exist, the operation creates the record. `boolean` type, the default value is `false`.	No
byDelta	If `true` and an element does not currently exist in the incoming message, the target value is not changed. If `false` and node is declared `hiddenInDataServices`, the target value is not changed. The complete behavior is described in the sections Insert and update operations [p 268].	No
blockingConstraintsDisabled	See the description under Common parameters [p 191].	
details	See the description under Common parameters [p 191].	
data	See the description under Common parameters [p 191].	

Update response

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

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. '95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted.
blockingConstraintMessage	This element is present if the status is equal to '95' with a localized message. The locale of the message is retrieved from the request parameter or from the user session.

Insert operation

Insert request

```
<m:insert_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <byDelta>boolean</byDelta>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
 <data>
  <XX>
   <TableName>
    <a>String</a>
    <b>String</b>
    <c>String</c>
    <d>String</d>
    ...
   </TableName>
  </XX>
 </data>
</m:insert_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 191].	
instance	See the description under Common parameters [p 191].	
byDelta	If `true` and an element does not currently exist in the incoming message, the target value is not changed. If `false` and node is declared `hiddenInDataServices`, the target value is not changed. The complete behavior is described in the sections Insert and update operations [p 268].	No
blockingConstraintsDisabled	See the description under Common parameters [p 191].	
details	See the description under Common parameters [p 191].	
data	See the description under Common parameters [p 191].	

Insert response

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

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. '95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted.
blockingConstraintMessage	This element is present if the status is equal to '95' with a localized message. The locale of the message is retrieved from the request parameter or from the user session.
predicate	A predicate matching the primary key of the inserted record. When several records are inserted, the predicates follow the declaration order of the records in the input message.

Get changes operations

Returns changes according to the Content policy [p 189].

Get changes requests

Changes between two datasets:

```
<m:getChangesOnDataSet_{schemaName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <compareWithBranch>String</compareWithBranch>
 <compareWithVersion>String</compareWithVersion>
 <compareWithInstance>String</compareWithInstance>
 <resolvedMode>boolean</resolvedMode>
 <includeInstanceUpdates>boolean</includeInstanceUpdates>
</m:getChangesOnDataSet_{schemaName}>
```

Changes between two tables:

```
<m:getChanges_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <compareWithBranch>String</compareWithBranch>
 <compareWithVersion>String</compareWithVersion>
 <resolvedMode>boolean</resolvedMode>
</m:getChanges_{TableName}>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 191].	
version	See the description under Common parameters [p 191].	
instance	See the description under Common parameters [p 191].	
compareWithBranch	The identifier of the dataspace with which to compare.	One of either this parameter or the 'compareWithVersion [p 200]' parameter must be defined.
compareWithVersion	The identifier of the snapshot with which to compare.	One of either this parameter or the 'compareWithBranch [p 200]' parameter must be defined.
compareWithInstance	The identifier of the dataset with which to compare. If it is undefined, instance [p 200] parameter is used.	No
resolvedMode	Defines whether or not the difference is calculated in resolved mode. Default is `true`.	No
includeInstanceUpdates	Defines if the content updates of the dataset are included. Default is `false`.	No
pagination	Enables pagination context for the operations `getChanges` and `getChangesOnDataSet`. Allows client to define pagination context size. Each page contains a collection of inserted, updated and/or deleted records of tables according to the maximum size. Get changes persisted context is built at first call according to the page size parameter in request. For creation: Defines `pageSize` parameter. For next: Defines `context` element with `identifier` from previous response. Enables pagination, see child elements below.	No
pageSize (nested under `pagination` element)	Defines maximum number of records in each page. Minimal size is `50`.	No (Only for creation)
context (nested under `pagination` element)	Defines content of pagination context.	No (Only for next)
identifier (nested under `context` element)	Pagination context identifier.	Yes

Note

> If none of the *compareWithBranch* or *compareWithVersion* parameters are specified, the comparison will be made with their parent:
>
> - if the current dataspace or snapshot is a dataspace, the comparison is made with its initial snapshot (includes all changes made in the dataspace);
> - if the current dataspace or snapshot is a snapshot, the comparison is made with its parent dataspace (includes all changes made in the parent dataspace since the current snapshot was created);
> - returns an exception if the current dataspace is the 'Reference' dataspace.

Get changes responses

Changes between two datasets:

```
<ns1:getChangesOnDataSet_{schemaName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <updated>
  <changes>
   <path>... Path of changed terminal value ...</path>
   <path>...</path>
  </changes>
  <data>
   ... see the whole content of data set values (without table) ...
  </data>
 </updated>
 <getChanges_{TableName1}>
  ... see the getChanges between tables response example ...
 </getChanges_{TableName1}>
 <getChanges_{TableName2}>
  ... see the getChanges between tables response example ...
 </getChanges_{TableName2}>
 ...
</ns1:getChangesOnDataSet_{schemaName}Response>
```

Changes between two tables:

```
<ns1:getChanges_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <inserted>
  <XX>
   <TableName>
    <a>AVALUE3</a>
    <b>BVALUE3</b>
    <c>CVALUE3</c>
    <d>DVALUE3</d>
   </TableName>
  </XX>
 </inserted>
 <updated>
  <changes>
   <change predicate="./a='AVALUE2'">
    <path>/b</path>
    <path>/c</path>
   </change>
  </changes>
  <data>
   <XX>
    <TableName>
     <a>AVALUE2</a>
     <b>BVALUE2.1</b>
     <c>CVALUE2.1</c>
     <d>DVALUE2</d>
    </TableName>
   </XX>
  </data>
 </updated>
 <deleted>
  <predicate>./a='AVALUE1'</predicate>
 </deleted>
</ns1:getChanges_{TableName}Response>
```

with:

Element	Description	Required
inserted	Contains inserted record(s) from choice `compareWithBranch` or `compareWithVersion`. Content under this element corresponding to an XML export of inserted records.	No
updated	Contains updated record(s) or dataset content.	No
changes (nested under `updated` element)	Only the group of field have been updated.	Yes
change (nested under `changes` element)	Group of fields have been updated with own `XPath predicate` attribute of the record.	Yes
path (nested under `change` element)	Path in the record.	Yes
path (nested under `changes` element)	Path in the dataset.	Yes
data (nested under `updated` element)	Content under this element corresponding to an XML export of dataset or updated records.	No
deleted	Records have been deleted from context of request. Content corresponding to a list of `predicate` element who contains the XPath predicate of record.	No
pagination	When pagination is enabled on request. Get changes persisted context allows invoking the next page until last page or when the context timeout is reached. Contains a next page: Defines `context` element with `identifier`. Is the last page: Defines `context` element without `identifier`. Enables pagination, see child elements below.	No
context (nested under `pagination` element)	Defines content of pagination context.	Yes (Only for next and last)
identifier (nested under `context` element)	Pagination context identifier. Not defined at last returned page.	No
pageNumber (nested under `context` element)	Current page number in pagination context.	Yes
totalPages (nested under `context` element)	Total pages in pagination context.	Yes

Get changes operation with pagination enabled

Only `pagination` element and sub elements have been described.

For creation:

Extract of request:

```
...
 <pagination>
  <!-- on first request for creation -->
  <pageSize>Integer</pageSize>
 </pagination>
...
```

Extract of response:

```
...
 <pagination>
  <!-- on next request to continue -->
  <context>
   <identifier>String</identifier>
   <pageNumber>Integer</pageNumber>
   <totalPages>Integer</totalPages>
  </context>
 </pagination>
...
```

For next:

Extract of request:

```
...
 <pagination>
  <context>
   <identifier>String</identifier>
  </context>
 </pagination>
...
```

Extract of response:

```
...
 <pagination>
  <!-- on next request to continue -->
  <context>
   <identifier>String</identifier>
   <pageNumber>Integer</pageNumber>
   <totalPages>Integer</totalPages>
  </context>
 </pagination>
...
```

For last:

Extract of request:

```
...
 <pagination>
  <context>
   <identifier>String</identifier>
  </context>
 </pagination>
...
```

Extract of response:

```
...
 <pagination>
  <context>
   <pageNumber>Integer</pageNumber>
   <totalPages>Integer</totalPages>
  </context>
 </pagination>
...
```

Get credentials operation

Get credentials request

```
<m:getCredentials_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <viewPublication>String</viewPublication>
</m:getCredentials_{TableName}>
```

with:

Element	**Description**	**Required**
branch	See the description under Common parameters [p 191].	
version	See the description under Common parameters [p 191].	
instance	See the description under Common parameters [p 191].	
predicate	See the description under Common parameters [p 191].	
viewPublication	See the description under Common parameters [p 191].	

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 dataspace.

When an error occurs during one operation in the sequence, all updates are rolled back and the client receives a `StandardException` error message with details.

```
<m:multi_ xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <blockingConstraintsDisabled>boolean</blockingConstraintsDisabled>
 <details locale="Locale"/>
 <request id="id1">
  <{operation}_{TableName}>
  ...
  </{operation}_{TableName}>
 </request>
 <request id="id2">
  <{operation}_{TableName}>
  ...
  </{operation}_{TableName}>
 </request>
</m:multi_>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 191].	
version	See the description under Common parameters [p 191].	
instance	See the description under Common parameters [p 191].	
blockingConstraintsDisabled	See the description under Common parameters [p 191].	
details	See the description under Common parameters [p 191].	
request	This element contains one operation, like a single operation without branch, version and instance parameters. This element can be repeated multiple times for additional operations. Each request can be identified by an 'id' attribute. In a response, this 'id' attribute is returned for identification purposes. Operations such as `count`, `select`, `getChanges`, `getCredentials`, `insert`, `delete` or `update`.	Yes

Note:

- Does not accept a limit on the number of `request` elements.
- The request `id` attribute must be unique in multi-operation requests.
- If all operations are read only (`count`, `select`, `getChanges`, or `getCredentials`) then the whole transaction is set as read-only for performance considerations.

Limitation:

- The `multi` operation applies to one model and one 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 on a record that was read earlier but may have changed in the meantime, an optimistic locking mechanism is provided.

A select request can include technical information by adding the element `includesTechnicalData`:

```
<m:select_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <includesTechnicalData>boolean</includesTechnicalData>
</m:select_{TableName}>
```

The value of the `lastTime` attribute can then be used in the following update request. If the record has been changed since the specified time, the update will be cancelled. The attribute `lastTime` has to be added on the record to prevent the update of a modified record.

```
<m:update_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
    <version>String</version>
 <instance>String</instance>
 <updateOrInsert>true</updateOrInsert>
 <data>
  <XX>
   <TableName ebxd:lastTime="2010-06-28T10:10:31.046">
    <a>String</a>
    <b>String</b>
    <c>String</c>
    <d>String</d>
    ...
   </TableName>
  </XX>
 </data>
</m:update_{TableName}>
```

The value of the `lastTime` attribute can also be used to prevent deletion on a modified record:

```
<m:delete_{TableName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <predicate>String</predicate>
 <checkNotChangedSinceLastTime>2010-06-28T10:10:31.046</checkNotChangedSinceLastTime>
```

```
</m:delete_{TableName}>
```

> **Note**
>
> The element `checkNotChangedSinceLastTime` may be used more than once but only for the same record. This implies that if the `predicate` element returns more than one record, the request will fail.

37.2 **Operations on datasets and dataspaces**

Parameters for operations on dataspaces and snapshots are as follows:

Element	*Description*	*Required*
branch	Identifier of the target dataspace on which the operation is applied. When not specified, the 'Reference' dataspace is used except for the merge dataspace operation where it is required.	One of either this parameter or the 'version' parameter must be defined. Required for the dataspace merge, locking, unlocking and replication refresh operations.
version	Identifier of the target snapshot on which the operation is applied.	One of either this parameter or the 'branch' parameter must be defined
versionName	Identifier of the snapshot to create. If empty, it will be defined on the server side.	No
childBranchName	Identifier of the dataspace child to create. If empty, it will be defined on the server side.	No
instance	The unique name of the dataset on which the operation is applied.	Required for the replication refresh operation.
ensureActivation	Defines if validation must also check whether this instance is activated.	Yes
details	Defines if validation returns details. The optional attribute `severityThreshold` defines the lowest severity level of message to return. The possible values are, in descending order of severity, 'fatal', 'error', 'warning', or 'info'. For example, setting the value to 'error' will return error and fatal validation messages. If this attribute is not defined, all levels of messages are returned by default. The optional attribute `locale` (default 'en-US') defines the language in which the validation messages are to be returned.	No. If not specified, no details are returned.
owner	Defines the owner.	No
branchToCopyPermissionFrom	Defines the identifier of the dataspace 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 dataspace

Validate dataspace request

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

Validate dataspace response

```
<ns1:validate_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <validationReport>
  <instanceName>String</instanceName>
  <fatals>boolean</fatals>
  <errors>boolean</errors>
  <infos>boolean</infos>
  <warnings>boolean</warnings>
 </validationReport>
</ns1:validate_Response>
```

Validate a dataset

Validate dataset request

```
<m:validateInstance xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <ensureActivation>boolean</ensureActivation>
 <details severityThreshold="fatal|error|warning|info" locale="Locale"/>
</m:validateInstance>
```

Validate dataset response

```
<ns1:validateInstance_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <validationReport>
  <instanceName>String</instanceName>
  <fatals>boolean</fatals>
  <errors>boolean</errors>
  <infos>boolean</infos>
  <warnings>boolean</warnings>
  <details>
   <reportItem>
    <severity>{fatal|error|warning|info}</severity>
    <message>
     <internalId />
     <text>String</text>
    </message>
    <subject>
     <table>Path</table>
     <predicate>String</predicate>
     <path>Path</path>
    </subject>
   </reportItem>
  </details>
 </validationReport>
</ns1:validateInstance_Response>
```

Create a dataspace

Create dataspace request

```
<m:createBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <owner>String</owner>
 <branchToCopyPermissionFrom>String</branchToCopyPermissionFrom>
 <documentation>
  <locale>Locale</locale>
  <label>String</label>
  <description>String</description>
 </documentation>
 <childBranchName>String</childBranchName>
</m:createBranch>
```

Create dataspace response

```
<ns1:createBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <childBranchName>String</childBranchName>
</ns1:createBranch_Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully.

Create a snapshot

Create snapshot request

```
<m:createVersion xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <versionName>String</versionName>
 <owner>String</owner>
 <documentation>
  <locale>Locale</locale>
  <label>String</label>
  <description>String</description>
 </documentation>
</m:createVersion>
```

Create snapshot response

```
<ns1:createVersion_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
 <versionName>String</versionName>
</ns1:createVersion_Response>
```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully.

Locking a dataspace

Lock dataspace request

```
<m:lockBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <durationToWaitForLock>Integer</durationToWaitForLock>
 <message>
  <locale>Locale</locale>
  <label>String</label>
 </message>
</m:lockBranch>
```

with:

Element	*Description*	*Required*
durationToWaitForLock	This parameter defines the maximum duration (in seconds) that the operation waits for a lock before aborting.	No, does not wait by default
message	User message of the lock. Multiple `message` elements may be used.	No
locale (nested under the `message` element)	Locale of the user message.	Only required when the `message` element is used
label (nested under the `message` element)	The user message.	No

Lock dataspace response

```
<ns1:lockBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:lockBranch_Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully. '94' indicates that the dataspace has been already locked by another user. Otherwise, a SOAP exception is thrown.

Unlocking a dataspace

Unlock dataspace request

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

Unlock dataspace response

```
<ns1:unlockBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:unlockBranch_Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully. Otherwise, a SOAP exception is thrown.

Merge a dataspace

Merge dataspace request

```
<m:mergeBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <deleteDataOnMerge>boolean</deleteDataOnMerge>
 <deleteHistoryOnMerge>boolean</deleteHistoryOnMerge>
</m:mergeBranch>
```

with:

Element	*Description*	*Required*
deleteDataOnMerge	This parameter is available for the merge dataspace operation. Sets whether the specified dataspace and its associated snapshots will be deleted upon merge.	No
deleteHistoryOnMerge	This parameter is available for the merge dataspace operation. Sets whether the history associated with the specified dataspace will be deleted upon merge. Default value is `false`.	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 dataspace.

Merge dataspace response

```
<ns1:mergeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:mergeBranch_Response>
```

with:

Element	**Description**
status	'00' indicates that the operation has been executed successfully.

Close a dataspace or snapshot

Close dataspace or snapshot request

Close dataspace request:

```
<m:closeBranch xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>String</branch>
 <deleteDataOnClose>boolean</deleteDataOnClose>
 <deleteHistoryOnClose>boolean</deleteHistoryOnClose>
</m:closeBranch>
```

Close snapshot request:

```
<m:closeVersion xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <version>String</version>
 <deleteDataOnClose>boolean</deleteDataOnClose>
</m:closeVersion>
```

with:

Element	*Description*	*Required*
deleteDataOnClose	This parameter is available for the close dataspace and close snapshot operations. Sets whether the specified snapshot, or data space and its associated snapshots, will be deleted upon closure.	No
deleteHistoryOnClose	This parameter is available for the close dataspace operation. Sets whether the history associated with the specified dataspace will be deleted upon closure. Default value is `false`.	No

Close dataspace or snapshot response

Close dataspace response:

```
<ns1:closeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:closeBranch_Response>
```

Close snapshot request:

```
<ns1:closeVersion_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</ns1:closeVersion_Response>
```

37.3 Operations on data workflows

Parameters for data workflows operations are retrieved from the SOAP header in the session.

Deprecated since version 5.7.0 to define parameters in the SOAP message body.

See session parameters [p 172] for more information.

Element	Description	Required
parameters	*Deprecated since version 5.7.0* While it remains available for backward compatibility, it will eventually be removed in a future major version. **Note** The `parameters` element is ignored if at least one session parameter has been defined.	No
parameter (nested under the `parameters` element). Multiple `parameter` elements may be used.	An input parameter for the workflow.	No
name (nested under the `parameter` element)	Name of the parameter.	Yes
value (nested under the `parameter` element)	Value of the parameter.	No

Start a workflow

Start a workflow from a workflow launcher. It is possible to start a workflow with localized documentation and specific input parameters (with name and optional value).

> **Note**
>
> The workflow creator is initialized from the session and the workflow priority is retrieved from the last published version.

Sample request:

```
<m:workflowProcessInstanceStart xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <publishedProcessKey>String</publishedProcessKey>
 <documentation>
  <locale>Locale</locale>
  <label>String</label>
  <description>String</description>
 </documentation>
</m:workflowProcessInstanceStart>
```

with:

Element	Description	Required
publishedProcessKey	Identifier of the workflow launcher.	Yes
documentation	See the description under Common parameters [p 208].	No
parameters	*Deprecated since version 5.7.0* See the description under Common parameters [p 214].	No

Sample response:

```
<m:workflowProcessInstanceStart_Response xmlns:m="urn:ebx-schemas:dataservices_1.0">
```

```
 <processInstanceKey>String</processInstanceKey>
</m:workflowProcessInstanceStart_Response>
```

with:

Element	Description	Required
processInstanceId	*Deprecated since version 5.6.1* This parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No
processInstanceKey	Workflow identifier.	No

Resume a workflow

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

Sample request:

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

with:

Element	Description	Required
resumeId	Resume identifier of the waiting task.	Yes
parameters	*Deprecated since version 5.7.0* See the description under Common parameters [p 214].	No

Sample response:

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

with:

Element	Description	Required
status	'00' indicates that the operation has been executed successfully. '20' indicates that the workflow has not been found. '21' indicates that the event has already been received.	Yes
processInstanceKey	Identifier of the workflow. This parameter is returned if the operation has been executed successfully.	No

End a workflow

End a workflow from its identifier.

Sample request:

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

```
 <processInstanceKey>String</processInstanceKey>
</m:workflowProcessInstanceEnd>
```

with:

Element	Description	Required
processInstanceKey	Identifier of the workflow.	Either this parameter or 'publishedProcessKey' and 'processInstanceId' parameters must be defined.
publishedProcessKey	*Deprecated since version 5.6.1* Due to a limitation this parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No
processInstanceId	*Deprecated since version 5.6.1* Due to a limitation this parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No

Sample response:

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

with:

Element	Description	Required
status	'00' indicates that the operation has been executed successfully.	Yes

CHAPTER 38

RESTful operations

This chapter contains the following topics:

38.1 Introduction

The architecture used is called ROA (Resource-Oriented Architecture), it can be an alternative to SOA (Service-Oriented Architecture). The chosen resources are readable and/or writable by third-party systems, according to the request content.

> **Note**
>
> All operations are stateless.

38.2 Request

This chapter describes the elements to use in order to build a conform REST request, such as: the HTTP method, the URL format, the header fields and the message body.

- Interactions [p 172] for details on tracking information and session parameters.
- Data services security [p 173] for details on authentication methods and global permissions.

HTTP method

Considered HTTP methods for REST operations, are:

- `GET`: used to select master data defined in the URL (the URL size limit depends on the application server or on the browser, that must be lower than or equal to 2KB).
- `POST`: used to insert one or more records in a table or to select the master data defined in the URL (the size limit is 2MB or more depending on the application server. Each parameter is limited to a value containing 1024 characters).

- PUT: used to update the master data defined in the URL.
- DELETE: used to delete either the record defined in the URL or multiple records defined with the table URL and the record table in the message body.

URL

REST URL contains:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/<restCategory>/<categoryVersion>/
<specificPath>?<urlParameters>
```

Where:

- `<restCategory>` corresponds to the operation category [p 218].
- `<categoryVersion>` corresponds to the category version: current value is `v1`.
- `<specificPath>` corresponds to a specific path inside the category.
- `<urlParameters>` corresponds to common [p 219] or dedicated operation parameters passed by the URL.

Operation category

It specializes the operation, it is added in path of URL in `<restCategory>` and it take one of the following values:

auth	Manage token authentication method. For more information, see: Token authentication operations [p 222] and Token Authentication Scheme [p 173].
data	Lists dataset content, requests a table, a record or a field record content, including modified operations on dataset node, table, record and record field. For more information, see: Data operations [p 223].
history	Lists history dataset content, requests an history table, an history of a record or an history record. For more information, see: Data operations [p 223].

Header fields

These header field definitions are used by EBX5.

`Accept`	Used to specify content types by order of preference to be used in the response, the first supported one will be chosen and specified in the response header `Content-Type`. Currently, the only supported one is `application/json`. If none is supported, the result depends on the `ebx.dataservices.rest.request.checkAccept` property: • If `true`, an HTTP error response is returned with code `406`. • If `false`, the response is returned with the default content type, that is `application/json`.
`Accept-Language`	Used for specifying the preferred locale for the response. The supported locales are defined in the schema model. If none of the preferred locale are supported, the default locale for the current model is used.
`Authorization`	Used for 'Basic Authentication Scheme' and 'Token Authentication Scheme' methods, otherwise the request is rejected. **See also** *Authentication* *[p 173]*

See RFC2616 for more information about HTTP Header Field Definitions.

Common parameters

These optional parameters are available for all data service operations.

Parameter	Description
`indent`	Specifies if the response should be indented, to be easier to read for a human. `Boolean` type, default value is `false`.

Message body

It contains the request data using the JSON format, see JSON Request body [p 246].

> **Note**
>
> Requests may define a message body only when using `POST` or `PUT` HTTP methods.

38.3 **Response**

This chapter describes the responses returned by RESTful services.

- See Exception handling [p 173] for details on standard error handling (where the HTTP code is greater than or equal to `300`).

Header fields

These header field definitions are used by EBX5.

`Content-Type`	Indicates the response format type.
`Content-Language`	Indicates the locale used in the response for labels and descriptions.
`WWW-Authenticate`	This header field is added to the HTTP response on authentication failure only when 'Basic Authentication Scheme' method is enabled. It is set to `Basic` when authentication aborts with 401 *(Unauthorized)* HTTP code. **See also** *Authentication* *[p 173]*
`Location`	If a new record has been successfully inserted, the query URL for this record is returned by this field.

HTTP codes

HTTP code	Description
`200` *(OK)*	The request has been successfully handled.
`201` *(Created)*	A new record has been created, in this case, the header field `Location` is returned with its resource URL.
`204` *(No content)*	The request has been successfully handled but no response body is returned.
`400` *(Bad request)*	the requet's URL or body is not well-formed or contains invalid content.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	Permission was denied to read or modify the specified resource for the authenticated user. This error is also returned when the user: • is not allowed to modify a field mentioned in the request message body. • is not allowed to access the REST connector.
`404` *(Not found)*	The resource specified in the URL cannot be found.
`406` *(Not acceptable)*	Content type defined in the request's `Accept` parameter is not supported. This error can be returned only if the EBX5 property `ebx.rest.request.checkAccept` is set to `true`.
`409` *(Conflict)*	A concurrent modification has occurred. **See also** *Optimistic locking* [p 241]
`422` *(Unprocessable entity)*	The new resource's content cannot be accepted for semantic reasons.
`500` *(Internal error)*	Unexpected error thrown by the application. Error details can usually be found in EBX5 logs.

Message body

The response body content's format depends on the HTTP code value:

- HTTP codes from `200` included to `300` excluded: the content format depends on the associated request (JSON [p 249] samples).

  With the exception of code `204` *(No content)*.

- HTTP codes greater than or equal to `300`: the content describes the error. See JSON [p 173] for details on the format.

38.4 Token authentication operations

These operations allow to create or revoke an authentication token. Authentication tokens have a timeout period. If a token is not used to access the EBX5 server within this period, it will automatically be revoked. This timeout period is refreshed on each access to EBX5 server.

Create token operation

This operation requires using the `POST` HTTP method with the message body containing the user's credentials and, optionally, session parameters [p 172].

URL format is:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/auth/v1/token:create
```

> **Note**
>
> No other authentication information is required in the HTTP headers.

Message body

The message body must be defined in the request. It necessarily contains a `login` and a `password` value (they are both `String`).

See the JSON [p 246] example of a token creation request.

HTTP codes

HTTP code	Description
`200` *(OK)*	The token was successfully created.
`400` *(Bad request)*	The request is not correct, it contains one of the following errors: • the configuration is not activated, • the syntax is not correct, • the HTTP method is not `POST`, • the operation is not supported.
`401` *(Unauthorized)*	Authentication has failed, for one of the following reasons: • The `login` and/or the `password` is/are incorrect. • Authentication data for other methods are defined.

Response body

If HTTP code is `200 (OK)`, the body holds the token value and its type.

See the JSON [p 249] example of a token creation response.

The token can later be used to authenticate a user by setting the HTTP-Header `Authorization` accordingly.

See also *'Token authentication Scheme' method [p 173]*

Revoke token operation

This operation requires using the `POST` HTTP method. No message body is needed.

URL format is:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/auth/v1/token:revoke
```

Header fields

`Authorization`	This field is required, `tokenType` and `accessToken` fields must have the values returned from the "token create" operation. `> Authorization: <tokenType> <accessToken>`

HTTP codes

HTTP code	**Description**
`204` *(No content)*	The token has been revoked successfully.
`400` *(Bad request)*	The request is not correct, it contains one of the following errors: • the configuration is not activated, • the syntax is incorrect, • the HTTP method is not `POST`, • the operation is not supported.
`401` *(Unauthorized)*	Authentication has failed.

38.5 **Data operations**

Operations from the `data` operation category concern the datasets, the dataset fields, tables, records or record fields.

Operations from the `history` category and concern historized content from datasets, tables, records or the record fields.

Select operation

Select operation may use one of the following methods:

- `GET` HTTP method,
- `POST` HTTP method without message body or
- `POST` HTTP method with message body and optionally session parameters [p 172].

URL formats are:

- **Dataset tree**, depending on operation category [p 218]:

  The `data` category returns the hierarchy of the selected dataset, this includes group and table nodes.

The `history` category returns the hierarchy of the selected history dataset, this includes the pruned groups for history table nodes only.

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/<restCategory>/v1/<dataspace>/
<dataset>
```

> **Note**
>
> Terminal nodes and sub-nodes are not included.

- **Dataset node**: the `data` category returns the terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/<dataspace>/<dataset>/
  <pathInDataSet>
  ```

  > **Note**
  >
  > Not applicable with the `history` category.

- **Table**, depending on operation category [p 218]:

  the `data` category returns the table content and/or metadata, current page records and URLs for pagination.

  The `history` category returns the history table content and/or metadata, current page records and URLs for pagination.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/<restCategory>/v1/<dataspace>/
  <dataset>/<pathInDataSet>
  ```

- **Record**, depending on operation category [p 218]:

  the `data` category returns the record content and/or metadata.

  The `history` category returns history record content and/or metadata.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/<restCategory>/v1/<dataspace>/
  <dataset>/<pathInDataSet>/<encodedPrimaryKey>
  ```

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/<restCategory>/v1/<dataspace>/
  <dataset>/<pathInDataSet>?primaryKey=<xpathExpression>
  ```

  > **Note**
  >
  > The record access by the primary key (`primaryKey` parameter) is limited to its root node. It is recommended to use the encoded primary key, available in the `details` field in order to override this limitation. Similarly, for a history record, use the encoded primary key, available in the `historyDetails` field.

- **Field**, depending on operation category [p 218]:

  the `data` category returns the field record content where structure depends on its type.

  The `history` category returns the field history record content where structure depends on its type.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/<restCategory>/v1/<dataspace>/
  <dataset>/<pathInDataSet>/<encodedPrimaryKey>/<pathInRecord>
  ```

  > **Note**
  >
  > The field must be either an association node, a selection node, a terminal node or above.

Where:

- `<restCategory>` corresponds to the operation category [p 218].
- `<dataspace>` corresponds to the dataspace or snapshot identifier.
- `<dataset>` corresponds to the dataset identifier.
- `<pathInDataSet>` corresponds to the path of the dataset node, that can be a group node or a table node.
- `<encodedPrimaryKey>` corresponds to the encoded representation of the primary key.
- `primaryKey=<xpathExpression>` corresponds to the record primary key, using the XPath expression.
- `<pathInRecord>` corresponds to the path starting from the table node.

Parameters

The following parameters are applicable to the select operation.

Parameter	Description
`includeContent`	Includes the `content` field with the content corresponding to the selection. `Boolean` type, default value is `true`.
`includeDetails`	Includes the `details` field in the metadata and the response, for each indirect reachable resource. The returned value corresponds to its URL resource. Type `Boolean`, default value is `true`. **See also** *includeMeta* [p 226]
`includeHistory`	Includes those fields for a historized content: • The `history` property in the metadata. The returned value corresponds to a boolean value. • The `historyDetails` property in the content and for each indirectly reachable resource. This point is coupled to the includeDetails [p 226] parameter. The returned value corresponds to its URL resource. `Boolean` type, default value is `false`. **Note** The `includeHistory` parameter is ignored in the `history` category, the default value is `true`. **See also** *includeMeta* [p 226] *includeDetails* [p 226]
`includeLabel`	Includes the `label` field associated with each simple type `content`. Possible values are: • `yes`: the `label` is included for the foreign key, enumeration, record and selector values. • `all`: the `label` field is included, as for the `yes` value and also for the Content of simple type [p 260]. • `no`: the `label` field is not included (integration use case). `String` type, default value is `yes`. **Note** The `label` field is not included if it is equal to the `content` field.
`includeMeta`	Includes the `meta` field corresponding to the description of the structure returned in the `content` field. `Boolean` type, default value is `false`. **See also** *includeHistory* [p 226] *includeDetails* [p 226]

Parameter	Description
`includeSelector`	Includes the `selector` field in the response, for each indirect reachable resource. The returned value corresponds to its URL resource. Type `Boolean`, default value is `true`. **See also** *selector* [p 230]
`includeSortCriteria`	Includes the `sortCriteria` field corresponding to the list of sort criteria applied. The sort criteria parameters are added by using: • sort [p 229] • sortOnLabel [p 229] • viewPublication [p 230] `Boolean` type, default value is `false`. Example JSON [p 256]
`includeTechnicals`	Includes the internal technical data. `Boolean` type, default value is `false`. **See also** *Technical data* [p 265] **Note** This parameter is ignored with the `history` category.
`includeValidation`	Includes the validation report corresponding to the selection. `Boolean` type, default value is `false`. **Note** This parameter is ignored with the `history` category. **See also** *includeDetails* [p 226]

Table parameters

The following parameters are applicable to tables, associations and selection nodes.

Parameter	Description
`filter`	XPath predicate [p 279] expression defines the field values to which the request is applied. If empty, all records will be retrieved. `String` type value. **Note** The history code operation value is usable with `ebx-operationCode` path field from the `meta` section associated with this field.
`historyMode`	Specifies the filter context applied on table. `String` type, possible values are: - `CurrentDataSpaceOnly`: history in current dataspace - `CurrentDataSpaceAndAncestors`: history in current dataspace and ancestors - `CurrentDataSpaceAndMergedChildren`: history in current dataspace and merged children - `AllDataSpaces`: history in all dataspaces The default value is `CurrentDataSpaceOnly`. **See also** *`history`* [p 218] **Note** This parameter is ignored with the `data` category.
`includeOcculting`	Includes the records in occulting mode. `Boolean` type, default value is `false`.
`primaryKey`	Search a record by a primary key, using the XPath expression. The XPath predicate [p 279] expression should only contain field(s) of the primary key and all of them. Fields are separated by the operator `and`. A field is represented by one of the following possibilities according to its simple type: - For the `date`, `time` or `dateTime` types: use the `date-equal(path, value)` - For other types: indicate the `path`, the `=` operator and the `value`. Example with a composed primary key: `./pk1i=1 and date-equal(./pk2d,'2015-11-13')` The response will only contain the corresponding record, otherwise an error is returned. Consequently, the other table parameters are ignored (as filter [p 228], viewPublication [p 230], sort [p 229], etc.) `String` type value.
`pageFirstRecordFilter`	Specifies the first record XPath predicate [p 279] expression filter of the page. `String` type value.
`pageAction`	Specifies the pagination action to perform starting from page defined by `pageFirstRecordFilter`.

Parameter	Description
	`String` type, possible values are: - `first` - `previous` - `next` - `last`
`pageSize`	Specifies the number of records per page. `Integer` type, default value is based on the user preferences. `String` type, the `unbounded` value can be defined to return all records. Only for this case, no pagination context is returned.
`sort`	Specifies that the operation result will be sorted according to the specified criteria. The criteria are composed of one or more criteria, the result will be sorted by priority from the left. A criterion is composed of the field path and, optionally, the sorting order (ascending or descending, on value or on label). This parameter can be combined with: 1. the sortOnLabel [p 229] parameter as a new criteria added after the `sort`. 2. the viewPublication [p 230] parameter as a new criteria added after the `sort`. The value structure is as follows: `<path1>:<order>;...;<pathN>:<order>` Where: - `<path1>` corresponds to the field path in priority `1`. - `<order>` corresponds to the sorting order, with one of the following values: - `asc`: ascending order on value (default), - `desc`: descending order on value, - `lasc`: ascending order on label, - `ldesc`: descending order on label. `String` type, the default value orders according to the primary key fields (ascending order on value). **Note** The history code operation value is usable with the `ebx-operationCode` path field from the `meta` section associated with this field.
`sortOnLabel`	Specifies that the operation result will be sorted according to the record label. This parameter can be combined with: 1. the sort [p 229] parameter as a new criteria added before the `sortOnLabel`. 2. the viewPublication [p 230] parameter as a new criteria added after the `sortOnLabel`. The value structure is as follows: `<order>` Where: - `<order>` corresponds to the sorting order, with one of the following values: - `lasc`: ascending order on label, - `ldesc`: descending order on label. `String` type value.

Parameter	Description
`viewPublication`	Specifies the name of the published view. This parameter can be combined with: 1. the filter [p 228] parameter as the logical `and` operation. 2. the sort [p 229] parameter as a new criteria added before the `viewPublication`. 3. the sortOnLabel [p 229] parameter as new criteria added before the `viewPublication`. The behavior of this parameter is described in the section . `String` type value.

Selector parameters

The following parameters are only applicable to fields that return an enumeration, foreign key or `osd:resource` (Example JSON [p 263]). By default, a pagination mechanism is enabled.

Parameter	Description
`selector`	Specifies whether: • `true`: returns all possible values (includes their labels) • `false`: returns the current values for the current field. `Boolean` type, default value is `false`. **Note** This parameter is ignored with the `history` category.
`firstElementIndex`	Specifies the index of the first element returned by the selector. Must be an integer higher than or equal to `0`. `Integer` type, default value is `0`.
`pageSize`	Specifies the number of elements per page. `Integer` type, default value is based on the user preferences. `String` type, the `unbounded` value can be defined to return all values. Only for this case, no pagination context is returned.
`selectorFilter`	Specifies the filter of the selector. `String` type value, the syntax complies with the Text search [p 88].

HTTP codes

HTTP code	Description
`200` *(OK)*	The selected resource is successfully retrieved.
`400` *(Bad request)*	The request is incorrect. This occurs when: • The selected field in a record or a dataset is sub-terminal, • The XPath predicate of the `filter` parameter is malformed or contains unfilterable nodes. • The XPath predicate of the `primaryKey` parameter is malformed or is not a record primary key. • The sort criteria of the `sort` parameter have an invalid syntax or contain unsortable nodes. • `pageAction` parameter value is not included in allowed values, or `pageFirstRecordFilter` is malformed or non-existent when selecting next or previous page. • `pageSize` parameter value is below 0, above 100, or is a string different from `unbounded`. • The table view for the `viewPublication` parameter is either hierarchical, non-existent or non-published. • The `selector` parameter is used for a non-enumerated node, or the `firstElementIndex` is negative, higher than or equal to the number of values.
`403` *(Forbidden)*	The selected resource is hidden for the authenticated user.
`404` *(Not found)*	The selected resource is not found.

Response body

After a successful dataset, table, record or field selection, the result is returned in the response body. The content depends on the provided parameters and selected data.

Example: JSON [p 249].

Insert operation

Insert operation uses the `POST` HTTP method. A body message is required to specify data. This operation supports the insertion of one or more records in a single transaction. Moreover, it is also possible to update record(s) through parameterization.

- **Record**: insert a new record or modify an existing one in the selected table.
- **Record table**: insert or modify one or more records in the selected table, while securing a consistent answer. Operations are executed sequentially, in the order defined on the client side. When an error occurs during a table operation, all updates are cancelled and the client receives an error message with detailed information.

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/<dataspace>/<dataset>/
<pathInDataSet>
```

Where:

- `<dataspace>` corresponds to the dataspace or snapshot identifier.

- `<dataset>` corresponds to the dataset identifier.
- `<pathInDataSet>` corresponds to the path of the table node.

Parameters

The following parameters are applicable with the insert operation.

Parameter	Description
`includeDetails`	Includes the `details` field in the answer for access to the details of the data. The returned value corresponds to its URL resources. Type `Boolean`, the default value is `false`. Note Only applicable on the **record table**.
`includeForeignKey`	Includes the `foreignKey` field in the answer for each record. The returned value corresponds to the value of a foreign key field that was referencing this record. `Boolean` type, the default value is `false`. Note Only applicable on the **record table**.
`includeLabel`	Includes the `label` field in the answer for each record. Possible values are: • `yes`: the `label` field is included. • `no`: the `label` field is not included (use case: integration). `String` type, the default value is `no`. Note Only applicable on the **record table**.
`updateOrInsert`	Specifies the behavior when the record to insert already exists: • If `true`: the existing record is updated with new data. For a request on a **record table**, the `code` field is added to the report in order to specify if this is an insert `201` or an update `204`. • If `false` (default value): a client error is returned and the operation is aborted. `Boolean` type value.
`blockingConstraintsDisabled`	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. `Boolean` type, default value is `false`.

Message body

The request must define a message body. The format depends on the inserted object type:

- **Record**: similar to the select operation of a record but without the record's header (example JSON [p 246]).
- **Record table**: Similar to the select operation on a table but without the pagination information (example JSON [p 247]).

See also *Inheritance [p 242]*

HTTP codes

HTTP code	Description
`200`*(OK)*	If the request relates to a **record table**. The insert request was applied successfully, an optional report is returned in the response body.
`201` *(Created)*	If the request relates to a **record**. A new record has been created, in this case, the header field `Location` is returned with its resource URL.
`204` *(No content)*	If the request relates to a **record**. Only available if `updateOrInsert` is `true`, an existing record has been successfully updated, in this case, the header field `Location` is returned with its resource URL.
`400` *(Bad request)*	The request is incorrect. This occurs when the body message structure does not comply with what was mentioned in Message body [p 232].
`403` *(Forbidden)*	Authenticated user is not allowed to create a record or the request body contains a read-only field.
`404` *(Not found)*	The selected resource is not found.
`409` *(Conflict)*	Concurrent modification, only available if `updateOrInsert` is `true`, the Optimistic locking [p 241] is activated and the content has changed in the meantime, it must be reloaded before update.
`422` *(Unprocessable entity)*	The request cannot be processed. This occurs when: • A blocking validation error occurs (only available if `blockingConstraintsDisabled` is `false`). • The record cannot be inserted because a record with the same primary key already exists (only available if `updateOrInsert` is `false`). • The record cannot be inserted because the definition of the primary key is either non-existent or incomplete. • The record cannot be updated because the value of the primary key cannot be modified.

Response body

The response body format depends on the inserted object type:

- **Record**: is empty if the operation was executed successfully. The header field `Location` is returned with its URL resource.

- **Record table**: (optional) contains a table of element(s), corresponding to the insert operation report (example JSON [p 263]). This report is automatically included in the response body, if at least one of the following options is set:
  - `includeForeignKey`
  - `includeLabel`
  - `includeDetails`

See also *Inheritance [p 242]*

Update operation

This operation allows the modification of a single dataset or record. The `PUT` HTTP method must be used. Available URL formats are:

- **Dataset node**: modifies the values of terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/<dataspace>/<dataset>/
  <pathInDataSet>
  ```

- **Record**: modifies the content of selected record.

  > **Note**
  >
  > Also available for `POST` HTTP methods. In this case, the URL must correspond to the table by setting the parameter `updateOrInsert` to `true`.

- **Field**: modifies the field content.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/<dataspace>/<dataset>/
  <pathInDataSet>/<encodedPrimaryKey>/<pathInRecord>
  ```

  > **Note**
  >
  > The field must be either a terminal node or above.

Where:

- `<dataspace>` corresponds to the dataspace or snapshot identifier.
- `<dataset>` corresponds to the dataset identifier.
- `<pathInDataSet>` corresponds to the path of the dataset node:
  - For dataset node operations, this must be any terminal node or above except table node,
  - For record and field operations, this corresponds to the table node.
- `<encodedPrimaryKey>` corresponds to the encoded representation of the primary key.
- `<pathInRecord>` corresponds to the path starting from the table node.

Parameters

Here are the parameters applicable with the update operation.

Parameter	Description
`blockingConstraintsDisabled`	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. `Boolean` type, default value is `false`.
`byDelta`	Specifies the behavior for setting value of nodes that are not defined in the request body. This is described in the Update modes [p 265] section. `Boolean` type, the default value is `true`.
`checkNotChangedSinceLastUpdateDate`	Timestamp in datetime format used to ensure that the record has not been modified since the last read. Also see the Optimistic locking [p 241] section. `DateTime` type value.

Message body

The request must define a message body.

The structure is the same as for:

- the dataset node (sample JSON [p 246]),
- the record (sample JSON [p 246]),
- the record fields (sample JSON [p 247]),

depending on the updated scope, by only keeping the `content` entry.

See also *Inheritance [p 242]*

HTTP codes

HTTP code	Description
`204` *(No content)*	The record, field or dataset node has been successfully updated.
`400` *(Bad request)*	The request is incorrect. This occurs when the body request structure does not comply.
`403` *(Forbidden)*	Authenticated user is not allowed to update the specified resource or the request body contains a read-only field.
`404` *(Not found)*	The selected resource is not found.
`409` *(Conflict)*	Concurrent modification, the Optimistic locking [p 241] is activated and the content has changed in the meantime, it must be reloaded before the update.
`422` *(Unprocessable entity)*	The request cannot be processed. This occurs when: • A blocking validation error occurs (only available if `blockingConstraintsDisabled` is `false`). • The record cannot be updated because the value of the primary key cannot be modified.

Delete operation

The operation uses the `DELETE` HTTP method.

Two URL formats are available:

- **Record**: delete a record specified in the URL.

  `http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/<dataspace>/<dataset>/<pathInDataSet>/<encodedPrimaryKey>`

- **Record table**: deletes several records in the specified table, while providing a consistent answer. This mode requires a body message containing a record table. The deletions are executed sequentially, according to the order defined in the table. When an error occurs during a table operation, all deletions are cancelled and an error message is displayed with detailed information.

  `http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/<dataspace>/<dataset>/<pathInDataSet>`

Where:

- `<dataspace>` corresponds to the dataspace or snapshot identifier.
- `<dataset>` corresponds to the dataset identifier.
- `<pathInDataSet>` corresponds to the path of the table node.
- `<encodedPrimaryKey>` corresponds to the encoded representation of the primary key.

In a child dataset context, this operation modifies the `inheritanceMode` property value of the record as follows:

- A record with inheritance mode set to `inherit` or `overwrite` becomes `occult`.
- A record with inheritance mode set to `occult` becomes `inherit` if the `inheritIfInOccultingMode` operation parameter is set to `true` or is undefined, otherwise there is no change.

- A record with inheritance mode set to `root` is simply deleted.

See also *Inheritance [p 242]*

Parameters

Here are the following parameters applicable with delete operation.

Parameter	**Description**
`includeOcculting`	Includes occulted records. `Boolean` type, the default value is `false`.
`inheritIfInOccultingMode`	*Deprecated since version 5.8.1* While it remains available for backward compatibility reasons, it will eventually be removed in a future version. Inherits the record if it is in occulting mode. `Boolean` type, the default value is `true`.
`checkNotChangedSinceLastUpdateDate`	Timestamp in datetime format used to ensure that the record has not been modified since the last read. Also see the Optimistic locking [p 241] section. `DateTime` type value.
`blockingConstraintsDisabled`	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. `Boolean` type, default value is `false`.

Message body

The request must define a message body only when deleting several records:

- **Record table**: The message contains a table of elements related to a record, with for each element one of the following properties:
  - `details`: corresponds to the record URL, it is returned by the select operation.
  - `primaryKey`: corresponds to the primary key of the record, using the XPath expression.
  - `foreignKey`: corresponds to the value that a foreign key would have if it referred to a record.

  Example JSON [p 248].

HTTP codes

HTTP code	Description
`200` *(OK)*	The operation has been executed successfully. A report is returned in the response body.
`400`*(Bad request)*	The request is incorrect. This occurs when: • the structure of the message body does not comply with Message body [p 237]. • the message body contains a record table while the URL specifies a record.
`403` *(Forbidden)*	Authenticated user is not allowed to delete or occult the specified record.
`404` *(Not found)*	The selected record is not found. In the child dataset context, it should be necessary to use the `includeOcculting` parameter.
`409` *(Conflict)*	Concurrent modification, The Optimistic locking [p 241] is activated and the content has changed in the meantime, it must be reloaded before deleting the record. The parameter value `checkNotChangedSinceLastUpdateDate` exists but does not correspond to the actual last update date of the record.
`422` *(Unprocessable entity)*	Only available if `blockingConstraintsDisabled` is `false`, the operation fails because of a blocking validation error.

Response body

After a successful record deletion or occulting, a report is returned in the response body. It contains the number of deleted, occulted and inherited record(s).

Example JSON [p 264].

Count operation

Count operation may use one of the following methods:

- `GET` HTTP method,
- `POST` HTTP method without message body or
- `POST` HTTP method with message body but without `content` field on root.

The URL formats are:

- **Dataset node**: the `data` category returns the number of terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/<dataspace>/<dataset>/
  <pathInDataSet>?count=true
  ```

  > **Note**
  >
  > Not applicable with the `history` category.

- **Table** depending on the operation category [p 218]:

  the `data` category returns the number of table records.

The `history` category returns the number of table history records.

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/<restCategory>/v1/<dataspace>/
<dataset>/<pathInDataSet>?count=true
```

- **Field** depending on the operation category [p 218]:

  the `data` category counts the record fields.

  The `history` category counts the history record field.

  ```
  http[s]://<host>[:<port>]/ebx-dataservices/rest/<restCategory>/v1/<dataspace>/
  <dataset>/<pathInDataSet>/<encodedPrimaryKey>/<pathInRecord>?count=true
  ```

  > **Note**
  >
  > The field must be either an association node, a selection node, a terminal node or above.

Where:

- `<restCategory>` corresponds to the operation category [p 218].
- `<dataspace>` corresponds to the dataspace or snapshot identifier.
- `<dataset>` corresponds to the dataset identifier.
- `<pathInDataSet>` corresponds to the path of the dataset node, that can be a group node or a table node.
- `<encodedPrimaryKey>` corresponds to the encoded representation of the primary key.
- `<pathInRecord>` corresponds to the path starting from the table node.

Parameters

The following parameters are applicable to the count operation.

Parameter	**Description**
`count`	This parameter is used to specify whether this is a count operation or a selection operation. `Boolean` type, default value is `false`.

Table parameters

The following parameters are applicable to tables, associations and selection nodes.

Parameter	Description
`filter`	XPath predicate [p 279] expression defines the field values to which the request is applied. If empty, all records will be considered. `String` type value. **Note** The history code operation value is usable with the `ebx-operationCode` path field from the `meta` section associated with this field.
`historyMode`	Specifies the filter context applied on table. `String` type, possible values are: • `CurrentDataSpaceOnly`: history in current dataspace • `CurrentDataSpaceAndAncestors`: history in current dataspace and ancestors • `CurrentDataSpaceAndMergedChildren`: history in current dataspace and merged children • `AllDataSpaces`: history in all dataspaces The default value is `CurrentDataSpaceOnly`. **See also** *`history`* *[p 218]* **Note** This parameter is ignored with the `data` category.
`includeOcculting`	Includes the records in occulting mode. `Boolean` type, default value is `false`.
`viewPublication`	Specifies the name of the published view to be considered during the count execution. This parameter can be combined with: • the filter [p 240] parameter as the logical `and` operation. The behavior of this parameter is described in the section .

Selector parameters

The following parameters are only applicable to fields that return an enumeration, foreign key or `osd:resource`.

Parameter	Description
`selector`	Specifies whether: • `true`: returns the number of all possible values • `false`: returns the number of possible values for the current field. `Boolean` type, default value is `false`. Note This parameter is ignored with the `history` category.
`selectorFilter`	Specifies the filter of the selector. `String` type value, the syntax complies with the Text search [p 88].

HTTP codes

HTTP code	Description
`200` *(OK)*	The selected resource is successfully counted.
`400` *(Bad request)*	The request is incorrect. This occurs when: • The selected field in a record or a dataset is sub-terminal. • The selected dataset field is a dataset tree. • The XPath predicate of the `filter` parameter is malformed or contains unfilterable nodes. • The table view for the `viewPublication` parameter is either hierarchical, non-existent or non-published. • The `selector` parameter is used for a non-enumerated node, or the `firstElementIndex` is negative, higher than or equal to the number of values.
`403` *(Forbidden)*	The selected resource is hidden for the authenticated user.
`404` *(Not found)*	The selected resource is not found.

Optimistic locking

To prevent an update or a delete operation on a record that was previously read but may have changed in the meantime, an optimistic locking mechanism is provided.

To enable optimistic locking, a select request must set the parameter `includeTechnicals` to `true`.

See Technical data [p 265] for more information.

The value of the `lastUpdateDate` property must be included in the following update request. If the record has been changed since the specified time, the update or delete will be cancelled.

- **Record**: update whole or partial content of the selected record.

  The property `lastUpdateDate` should be added to the request body to prevent update of a modified record.

  See the JSON [p 246] example of a record.
- **Field**: update of a single field of the selected record.

  The property value `lastUpdateDate` must be declared in the request URL by the `checkNotChangedSinceLastUpdateDate` parameter to prevent the update of a modified record.

The property value `lastUpdateDate` can also be used in the request URL `checkNotChangedSinceLastUpdateDate` parameter to prevent deletion on a modified record.

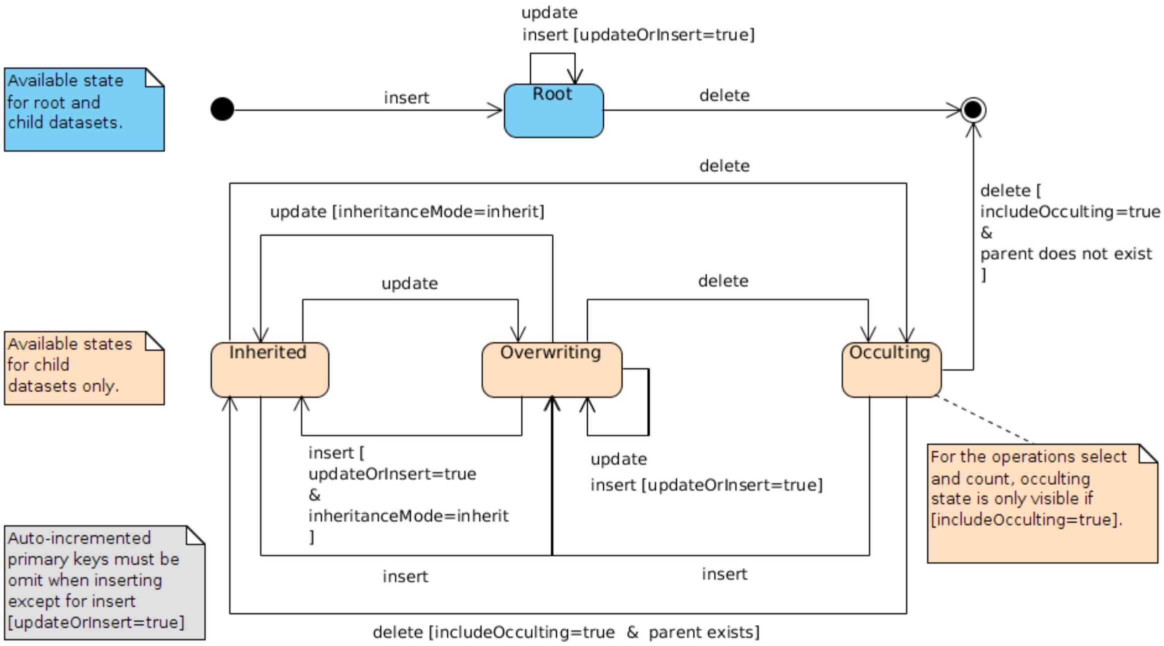
> **Note**
>
> The `checkNotChangedSinceLastUpdateDate` parameter may be used more than once but only on the same record. This implies that if the request URL returns more than one record, the request will fail.

Inheritance

EBX5 inheritance features are supported by RESTful operations using specific properties and automatic behaviors. In most cases, the inheritance state will be automatically computed by the server according to the record and field definition or content. Every action that modifies a record or a field may have an indirect impact on those states. In order to fully handle the inheritance life cycle, direct modifications of the state are allowed under certain conditions. Forbidden or incoherent explicit alteration attempts are ignored.

See also *Inheritance and value resolution [p 286]*

Inheritance life cycle in RESTful operations


update
insert [updateOrInsert=true]
Available state for root and child datasets.
insert
Root
delete
delete
delete [
includeOcculting=true
&
parent does not exist
]
update [inheritanceMode=inherit]
update
delete
Available states for child datasets only.
Inherited
Overwriting
Occulting
insert [
updateOrInsert=true
&
inheritanceMode=inherit
]
update
insert [updateOrInsert=true]
For the operations select and count, occulting state is only visible if [includeOcculting=true].
Auto-incremented primary keys must be omit when inserting except for insert [updateOrInsert=true]
insert
insert
delete [includeOcculting=true & parent exists]

Inheritance properties

The following table describes properties related to the EBX5 inheritance features.

Property	Location	Description
`inheritance`	record or table metadata	Specifies if dataset inheritance is activated for the table. The value is computed from the data model and cannot be modified through RESTful operations. **See also** *`inheritance` property in metadata.* [p 254]
`inheritedField`	field metadata	Specifies the field's value source. The source data are directly taken from the data model and cannot be modified through RESTful operations. **See also** *`inheritedField` property in metadata.* [p 255]
`inheritanceMode`	record in child dataset	Specifies the record's inheritance state. To set a record's inheritance from `overwrite` to `inherit`, its `inheritanceMode` value must be explicitly provided in the request. In this specific case, the `content` property will be ignored if present. `occult` and `root` explicit values are always ignored. An `overwrite` explicit value without a `content` property is ignored. **Note** Inherited record's fields are necessarily `inherit`. **Note** Root records in child dataset will always be `root`. Possible values are: `root`, `inherit`, `overwrite`, `occult`. For more information, see Record lookup mechanism [p 287].
	field in `overwrite` record	Specifies the field's inheritance state. To set a field's inheritance to `inherit`, its `inheritanceMode` value must be explicitly provided in the request. The `content` property will be ignored in this case. `overwrite` explicit value without a `content` property is ignored. **Note** `inheritanceMode` at field level does not appear for `root`, `inherit` and `occult` records. **Note** `inheritedFieldMode` and `inheritanceMode` properties cannot be both set on the same field. Possible values are: `inherit`, `overwrite`. For more information, see Inheritance and value resolution [p 286].
`inheritedFieldMode`	inherited field	Specifies the inherited field's inheritance state. To set a field's inheritance to `inherit`, its `inheritanceMode` value must be

Property	Location	Description
		explicitly provided in the request. The `content` property will be ignored in this case. `overwrite` explicit values without a `content` property are ignored. **Note** `inheritedFieldMode` and `inheritanceMode` properties cannot be both set on the same field. **Note** `inheritedFieldMode` has priority over the `inheritanceMode` property. Possible values are: `inherit`, `overwrite`. For more information, see Value lookup mechanism [p 288].

38.6 Limitations

General limitations

- Indexes, in request URL `<pathInDataSet>` or `<pathInRecord>`, are not supported.
- Nested aggregated lists are not supported.
- Dataset nodes and field operations applied to nodes that are sub-terminal are not supported.

Read operations

- Within the `selector`, the pagination context is limited to the `nextPage` property.
- Within the `viewPublication` parameter, the hierarchical view is not supported.

Write operations

- Association fields cannot be updated, therefore, the list of associated records cannot be modified directly.
- Control policy `onUserSubmit-checkModifiedValues` of the user interface is not supported. To retrieve validation errors, invoke the select operation on the resource by including the `includeValidation` parameter.

Directory operations

- Changing or resetting a password for a user is not supported.

CHAPTER 39

JSON format (RESTful)

This chapter contains the following topics:

39.1 Introduction

The JSON (JavaScript Object Notation) is the data-interchange format used by EBX5 RESTful operations [p 217].

This format is lightweight, self-describing and can be used to design UIs or to integrate EBX5 in a company's information system.

- The data context is exhaustive, except for association fields and selection nodes that are not directly returned. However, these fields are included in the response with a URL link named `details` included by default. It can be indirectly used to get the fields content.
- The volume of data is limited by a pagination mechanism activated by default, it can be configured or disabled.

URL formatted links allow retrieving:

- Tables, records, dataset non-terminal nodes, foreign keys, resource fields (property `details`).
- Possible values for foreign keys or enumerations (`selector` parameter).

> **Note**
>
> JSON data are always encoded in UTF-8.

39.2 Global structure

JSON Request body

Request body is represented by a JSON `Object` whose content varies according to the operation and its category.

Auth category

The request body holds several properties directly placed in the root JSON `Object`.

- **Token creation**

  Specifies the `login` and `password` to use for an authentication token creation attempt.

```
{
  "login": "...",                // JSON String
  "password": "..."              // JSON String
}
```

  See also *Create token operation* *[p 222]*

Data category

The request body contains at least a `content` property where master data values will be defined.

- **Dataset node**

  Specifies the target values of terminal nodes under the specified node. This request is used on the dataset node update operation.

```
{
   "content": {
      "nodeName1": {
         "content": true
      },
      "nodeName2": {
         "content": 2
      },
      "nodeName3": {
         "content": "Hello"
      }
   }
}
```

  See also *Update operation* *[p 234]*

- **Record**

  Specifies the target record content by setting the value for each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is used on table record insert or on record update.

  See also *Inheritance* *[p 242]*

  Some technical data can be added beside the `content` property such as `lastUpdateDate`.

  See also *Optimistic locking* *[p 241]*

```
{
   ...
   "lastUpdateDate": "2015-12-25T00:00:00.001",
   ...
   "content": {
      "gender": {
         "content": "Mr."
      },
      "gender-fr": {
         "content": "M.",
         "inheritedFieldMode": "overwrite"
      },
      "lastName": {
         "content": "Chopin"
      },
      "firstName": {
         "content": "Frederic"
      },
      "birthDate": {
         "content": "1810-03-01"
      },
      "birthDate-fr": {
         "inheritedFieldMode": "inherit"
      },
      "deathDate": {
         "content": "1849-10-17"
      },
      "jobs": {
         "content": [
            {
               "content": "CM"
            },
            {
               "content": "PI"
            }
         ]
      },
      "infos": {
         "content": [
            {
               "content": "https://en.wikipedia.org/wiki/Chopin"
            }
         ]
      }
   }
}
```

See also

Insert operation *[p 231]*

Update operation *[p 234]*

- **Record fields**

  Specifies the target values of fields under the record terminal node by setting the value of each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is only used for table record updates.

  See also *Inheritance* *[p 242]*

```
{
   "content": [
      {
         "content": "CM"
      },
      {
         "content": "PI"
      }
   ]
}
```

  See also *Update operation* *[p 234]*

- **Record table**

Defines the content of one or more records by indicating the value of each field. For missing fields, the behavior depends on the parameter of the request `byDelta`. This structure is used upon insert or update of records in the table.

```
{
   "rows": [
      {
         "content": {
            "gender": {
               "content": "M"
            },
            "lastName": {
               "content": "Saint-Saëns"
            },
            "firstName": {
               "content": "Camille"
            },
            "birthDate": {
               "content": "1835-10-09"
            },
            ...
         }
      },
      {
         "content": {
            "gender": {
               "content": "M"
            },
            "lastName": {
               "content": "Debussy"
            },
            "firstName": {
               "content": "Claude"
            },
            "birthDate": {
               "content": "1862-10-22"
            },
            ...
         }
      }
   ]
}
```

See also

Insert operation *[p 231]*

Update operation *[p 234]*

- **Record table to be deleted**

  Defines one or more records. This structure is used upon deleting several records from the same table.

```
{
   "rows": [
      {
         "details": "http://.../root/table/1"
      },
      {
         "details": "http://.../root/table/2"
      },
      {
         "primaryKey": "./oid=3"
      },
      {
         "foreignKey": "4"
      },
      ...
   ]
}
```

See also *Delete operation* *[p 236]*

- **Field**

Specifies the target field content. This request is used on the field update.

The request has the same structure as defined in node value [p 258] by only keeping the `content` entry. Other entries are simply ignored.

See also *Update operation [p 234]*

Only writable fields can be mentioned in the request, this excludes the following cases:

- Association node,
- Selection node,
- Value function,
- JavaBean field that does not have a setter,
- Unwritable permission on node for authenticated user.

JSON Response body

The response body is represented by a JSON `Object` whose content depends on the operation and its category.

Auth category

The response body contains several properties directly placed in its root JSON `object`.

- **Token creation**

  Contains the token value and its type.

```
{
  "accessToken": "...",            // JSON String
  "tokenType": "..."               // JSON String
}
```

See also *Create token operation [p 222]*

Data category

The selection operation contains two different parts.

The first one named `meta` contains the exhaustive structure of the response.

The other, regrouping `content`, `rows`, `pagination`...etc, contains the values corresponding to the request.

- **Dataset tree**

  Contains the hierarchy of `table` and non-terminal `group` nodes.

```
{
   "meta": {
      "fields": [
         {
            "name": "rootName",
            "label": "Localized label",
            "description": "Localized description",
            "type": "group",
            "pathInDataset": "/rootName",
            "fields": [
               {
                  "name": "settings",
                  "label": "Settings",
                  "type": "group",
                  "pathInDataset": "/rootName/settings",
                  "fields": [
                     {
                        "name": "settingA",
                        "label": "A settings label",
                        "type": "group",
                        "pathInDataset": "/rootName/settings/settingA"
                     },
                     {
                        "name": "settingB",
                        "label": "B settings label",
                        "type": "group",
                        "pathInDataset": "/rootName/settings/settingB"
                     }
                  ]
               },
               {
                  "name": "table1",
                  "label": "Table1 localized label",
                  "type": "table",
                  "minOccurs": 0,
                  "maxOccurs": "unbounded",
                  "pathInDataset": "/rootName/table1"
               },
               {
                  "name": "table2",
                  "label": "Table2 localized label",
                  "type": "table",
                  "minOccurs": 0,
                  "maxOccurs": "unbounded",
                  "pathInDataset": "/rootName/table2"
               }
            ]
         }
      ]
   },
   "validation": [
      {
         "level": "error",
         "message": "Value must be greater than or equal to 0.",
         "details": "http://.../rootName/settings/settingA/settingA1?includeValidation=true"
      },
      {
         "level": "error",
         "message": "Field 'Settings A2' is mandatory.",
         "details": "http://.../rootName/settings/settingA/settingA2?includeValidation=true"
      }
   ],
   "content": {
      "rootName": {
         "details": "http://.../rootName",
         "content": {
            "settings": {
               "details": "http://.../rootName/settings",
               "content": {
                  "weekTimeSheet": {
                     "details": "http://.../rootName/settings/settingA"
                  },
                  "vacationRequest": {
                     "details": "http://.../rootName/settings/settingB"
                  }
               }
            },
            "table1": {
               "details": "http://.../rootName/table1"
            },
            "table2": {
               "details": "http://.../rootName/table2"
            }
         }
      }
   }
}
```

The `meta` and `validation` properties are optional.

See also

Meta-data [p 254]

Validation [p 256]

Select operation [p 223]

- **Dataset node**

  Contains the list of terminal nodes under the specified node.

```
{
   "meta": {
      "fields": [
         {
            "name": "nodeName1",
            "label": "Localized label of the field node 1",
            "description": "Localized description",
            "type": "boolean",
            "minOccurs": 1,
            "maxOccurs": 1,
            "pathInDataset": "/rootName/.../nodeName1"
         },
         {
            "name": "nodeName2",
            "label": "Localized label of the field node 2",
            "type": "int",
            "minOccurs": 1,
            "maxOccurs": 1,
            "pathInDataset": "/rootName/.../nodeName2"
         }
      ]
   },
   "content": {
      "nodeName1": {
         "content": true
      },
      "nodeName2": {
         "content": -5,
         "validation": [
            {
               "level": "error",
               "message": "Value must be greater than or equal to 0."
            }
         ]
      }
   }
}
```

  See also *Select operation [p 223]*

- **Table**

  JSON `Object` containing the properties:

  - (Optional) The table meta data [p 254],
  - (Optional) The sort criteria applied,
  - (Optional) The table validation report,
  - `rows` containing a table of selected records. Each record is represented by an object, if no record is selected then the table is empty.
  - (Optional) `pagination` containing pagination [p 262] information if activated.

```
{
   "rows": [
      {
         "label": "Claude Levi-Strauss",
         "details": "http://.../root/individu/1",
         "content": {
            "id": {
               "content": 1
            },
         ...
         }
      },
      {
         "label": "Sigmoud Freud",
         "details": "http://.../root/individu/5",
         "content": {
            "id": {
               "content": 2
            },
         ...
         }
      },
   ...
      {
         "label": "Alfred Dreyfus",
         "details": "http://.../root/individu/10",
         "content": {
            "id": {
               "content": 30
            },
   ...
         }
      }
   ],
   "sortCriteria": [
      {
         "path": "/name",
         "order": "lasc"
      },
      ...
   ],
   "pagination": {
      "firstPage": null,
      "previousPage": null,
      "nextPage": "http://.../root/individu?pageFirstRecordFilter=./id=1&pageSize=9&pageAction=next",
      "lastPage": "http://.../root/individu?pageSize=9&pageAction=last"
   }
}
```

See also *Select operation* *[p 223]*

- **Record**

  JSON `object` containing:

  - The label,
  - (Optional) The record URL,
  - (Optional) The technical data [p 265],
  - (Optional) The table metadata [p 254],
  - (Optional) The record validation report,
  - (Optional) The inheritance mode of the record is: `root`, `inherit`, `overwrite` or `occult`. This value is available for a child dataset,
  - The record content.

```
{
   "label": "Name1",
   "details": "http://.../rootName/table1/pk1",
   "creationDate": "2015-02-02T19:00:53.142",
   "creationUser": "admin",
   "lastUpdateDate": "2015-09-01T17:22:24.684",
   "lastUpdateUser": "admin",
   "inheritanceMode": "root",
   "meta": {
      "name": "table1",
      "label": "Table1 localized label",
      "type": "table",
      "minOccurs": 0,
      "maxOccurs": "unbounded",
      "primaryKeys": [
         "/pk"
      ],
      "inheritance": "true",
      "fields": [
         {
            "name": "pk",
            "label": "Identifier",
            "type": "string",
            "minOccurs": 1,
            "maxOccurs": 1,
            "pathInRecord": "pk",
            "filterable": true,
            "sortable": true
         },
         {
            "name": "name",
            "label": "Name",
            "type": "string",
            "minOccurs": 1,
            "maxOccurs": 1,
            "pathInRecord": "name",
            "filterable": true,
            "sortable": true
         },
         {
            "name": "name-fr",
            "label": "Nom",
            "type": "string",
            "minOccurs": 1,
            "maxOccurs": 1,
            "inheritedField": {
                "sourceNode": "./name"
            },
            "pathInRecord": "name-fr",
            "filterable": true,
            "sortable": true
         },
         {
            "name": "parent",
            "label": "Parent",
            "description": "Localized description.",
            "type": "foreignKey",
            "minOccurs": 1,
            "maxOccurs": 1,
            "foreignKey": {
               "tablePath": "/rootName/table1",
               "details": "http://.../rootName/table1"
            },
            "enumeration": "foreignKey",
            "pathInRecord": "parent",
            "filterable": true,
            "sortable": true
         }
      ]
   },
   "content": {
      "pk": {
         "content": "pk1"
      },
      "name": {
         "content": "Name1"
      },
      "name-fr": {
         "content": "Name1",
         "inheritedFieldMode": "inherit"
      },
      "parent": {
         "content": null,
         "validation":
         [
            {
               "level": "error",
               "message": "Field 'Parent' is mandatory."
            }
         ]
      }
   },
   "validation": {
      ...
   }
}
```

See also *Select operation [p 223]*

- **Fields**

  For association or selection nodes, contains the target table with associated records if, and only if, the `includeDetails` parameter is set to `true`.

  For other kinds of nodes, contains the current node value [p 258], by adding `meta` entry if enabled.

See also *Select operation* *[p 223]*

> **Note**
>
> Node, records and field in `meta`, `rows` and `content` may be hidden depending on their resolved permissions.

39.3 Meta-data

This section can be activated on demand with the `includeMeta` [p 226] parameter. It describes the structure and the JSON typing of the `content` section.

This section is deactivated by default for selection operations.

See also *Select operation* *[p 223]*

Structure of table

Table meta-data is represented by a JSON `object` with the following properties:

JSON property	JSON format	Description	Required
`name`	`String`	Name of the table defined in schema.	Yes
`label`	`String`	Table label. If undefined, the name of the schema node is returned.	Yes
`description`	`String`	Table description.	No
`type`	`String`	Always equal to: `table`.	Yes
`minOccurs`	`Number`	Number of minimum authorized record(s).	Yes
`maxOccurs`	`Number` or `String`	Number of maximum authorized record(s) or `unbounded`.	Yes
`primaryKeyFields`	`Array`	Array of the paths corresponding to the primary key.	Yes
`inheritance`	`Boolean`	Specifies whether the dataset inheritance is activated for the table. Its value is `true` if inheritance is activated, `false` otherwise. **See also** *Inheritance and value resolution* *[p 286]*	No
`fields`	`Array`	Array of fields, that are direct children of the record node. Each field may also recursively contain sub-fields.	Yes

Structure of field

Each authorized field is represented by a JSON `object` with the following properties:

JSON property	JSON format	Description	Required
`name`	`String`	Name of the current authorized schema node.	Yes
`label`	`String`	Node label. If undefined, the name of the schema node is returned.	Yes
`description`	`String`	Node description.	No
`type`	`String`	Node type: simple type [p 260], `group`, `table`, `foreignKey`, etc.	Yes
`minOccurs`	`Number`	Number of minimum authorized occurrence(s).	Yes
`maxOccurs`	`Number` or `String`	Number of maximum authorized occurrence(s) or `unbounded`.	Yes

`inheritedField`	`Object`	Holds information related to the inherited field's value source.
		`"inheritedField": {`
		`    "sourceRecord": "/path/to/record", // (optional)`
		`    "sourceNode": "./path/to/Node"`
		`}`
		See also *Inheritance and value resolution* [p 286]
			No

`foreignKey`	`Object`	Contains information related to the target table.
		`{`
		`   "dataspace": "BAuthors",`
		`   "dataset": "Authors",`
		`   "tablePath": "/root/Authors",`
		`   "details": "http://.../BAuthors/Authors/root/`
		`Authors"`
		`}`
			No (*)

`dataspace`	`String`	Target dataspace or snapshot identifier. This property is placed under the `foreignKey` property.	No (*)
`dataset`	`String`	Target dataset identifier. This property is placed under the `foreignKey` property.	No (*)
`tablePath`	`String`	Target table path. This property is placed under the `foreignKey` property.	Yes
`details`	`String`	Target table URL. This property is placed under the `foreignKey` property and is included by default.	No
`enumeration`	`String`	Specifies if the field is an enumeration value. Possible values are: • `foreignKey`	No

JSON property	JSON format	Description	Required
		• static • dynamic • programmatic • nomenclature • resource Specifies whether the field can be used for retrieving possible values by using the `selector` request parameter.	
`pathInRecord`	`String`	Relative field path starting from the table node.	No (*)
`filterable`	`Boolean`	Specifies whether the field can be used for filtering record using `filter` request parameter.	No (*)
`sortable`	`Boolean`	Specifies whether the field can be used in sort criteria using `sort` request parameter.	No (*)
`fields`	`Array` of `Object` elements	Contains the structure and typing of each group field.	No

(*) Only available for table, record and record field operations.

39.4 Sort criteria information

The sort criteria applied to the request can be returned on demand, by using the `includeSortCriteria` [p 227] parameter (deactivated by default). If it is activated, a `sortCriteria` property is directly added to the response root node.

A `sortCriteria` property contains a JSON `Array` that contains ordered sort criteria, and for each sort criterion, a JSON `Object` is added with the following properties:

JSON property	JSON format	Description	Required
`path`	`String`	Field path.	Yes
`order`	`String`	Possible values are: `asc`, `lasc`, `desc` or `ldesc`.	Yes

39.5 Validation

The validation can be activated on demand with the `includeValidation` [p 227] parameter (deactivated by default). If it is activated, `validation` properties are directly added on target nodes with one or several messages. For messages without a target node path, a `validation` property is added on the root node.

A `validation` property contains a JSON `Array` and for each message, corresponding to a validation item, a JSON `Object` with properties:

JSON property	JSON format	Description	Required
`level`	`String`	Severity of the validation item, the possible values are: `info`, `warn`, `error`.	Yes
`message`	`String`	Description of the validation item.	Yes
`details`	`String` corresponding to an absolute URL.	URL of the resource associated with the validation item. Only available on the table and dataset scopes, if associated resources exist and if it is included. **See also** *`includeDetails` parameter* [p 226]	No

39.6 **Content**

This section can be deactivated on demand with the `includeContent` [p 226] parameter (activated by default). It provides the content of the record values, dataset, or field of one of the content fields for an authorized user. It also has additional information, including labels, technical information, URLs...

The content is represented by a JSON `Object` with a property set for each sub-node.

Node value

JSON property	JSON format	Description	Required
`content`	Content of simple type [p 260] Content of group and list [p 262]	Contains the node value. Available for all nodes except `association` and `selection`. However, their content can be retrieved by invoking the URL provided in `details`.	`No`
`details`	`String` corresponding to an absolute URL.	By invocation, the node details are returned. Response type after invocation depending on the meta type. • `foreignKey`: target record (available on table, record and field operation). • `association`: target table containing associated records (available on table and record operations). • `selection`: target table containing associated records (available on table and record operations). • `group`: target dataset group node (available on dataset tree operation). • `table`: target table (available on dataset tree operation). Example: `http://.../BReference/dataset/root/table/pk/associationField`	`No`
`label`	`String`	Contains the foreign key or enumeration label in the current locale. The default label is returned if the current locale is not supported.	`No`
`inheritanceMode`	`String`	Contains the node's inheritance state, considering only dataset inheritance. `inheritedFieldMode` and `inheritanceMode` properties cannot be both defined on the same node. **See also** *`inheritedFieldMode`* [p 258] *Record lookup mechanism* [p 287]	`No`
`inheritedFieldMode`	`String`	Contains the node's field inheritance state, considering dataset and field inheritance. When both inheritances are used, field inheritance has priority over the dataset one. `inheritedFieldMode` and `inheritanceMode` properties cannot be both defined on the same node. **See also** *`inheritanceMode`* [p 258] *Value lookup mechanism* [p 288]	`No`
`selector`	`String` corresponding to an absolute URL.	Correspond to the URL for selector [p 263] operation. Example:	`No`

JSON property	JSON format	Description	Required
		`http://.../BReference/dataset/root/table/pk/enumField?selector=true`	
`validation`	`Array`	Contains the validation report that concerns the current node context.	`No`

Content of simple type

A simple field value is stored in a JSON object and the content is the value of the `content` property.

XML Schema	JSON format	Examples	Meta type

`xs:string`
`xs:Name`
`osd:email`
`osd:html`
`osd:text`
	`String` (Unicode characters, cf. RFC4627)	"A text"
		"The escape of \"special character\" is preceded by a backslash."
		"<p>An HTML tag can thus be written without trouble</p>"
		"employee@mycompany.com"
		null
			`string`

`osd:locale`	`String` (Language tag, cf. RFC1766)	"en-US"	`locale`
`xs:string` (Foreign key)	`String` contains the value of the formatted foreign key.	`"0"` `"true|99"`	`foreignKey`
`xs:boolean`	`Boolean`	true false null	`boolean`
`xs:decimal`	`Number` or `null`	-10.5 20.001 15 -1e-13	`decimal`
`xs:date`	`String` with format: "yyyy-MM-dd"	"2015-04-13"	`date`
`xs:time`	`String` with format: • "HH:mm:ss" • "HH:mm:ss.SSS"	"11:55:00" "11:55:00.000"	`time`
`xs:dateTime`	`String` with format: • "yyyy-MM-ddTHH:mm:ss" • "yyyy-MM-ddTHH:mm:ss.SSS"	"2015-04-13T11:55:00" "2015-04-13T11:55:00.000"	`dateTime`
`xs:anyURI`	`String` (Uniform Resource Identifier, cf. RFC3986)	"https://fr.wikipedia.org/wiki/René_Descartes"	`anyURI`
`xs:int` `xs:integer`	`Number` or `null`	1596	`int`
`osd:resource`	`String`	"ebx-tutorial:ext-images:frontpages/Ajax for Dummies.jpg"	`resource`

<table>
<tr><th>XML Schema</th><th>JSON format</th><th>Examples</th><th>Meta type</th></tr>
<tr><td></td><td>contains the resource formatted value.</td><td></td><td></td></tr>
<tr><td>osd:color</td><td>String with format: "#[A-Fa-f0-9]{6}"
contains the formatted value for the color.</td><td>"#F6E0E0"</td><td>color</td></tr>
<tr><td>osd:datasetName</td><td>String with format: "[a-zA-Z_][-a-zA-Z0-9_.]*" and 64 characters max.
contains the formatted value of the dataset name.</td><td>"ebx-tutorial"</td><td>dataset</td></tr>
<tr><td>osd:dataspaceKey</td><td>String with format: "[BV][a-zA-Z0-9_:.\\-\\|]*" and 33 characters max.
contains the formatted key value of the dataspace.</td><td>"Bebx-tutorial"</td><td>dataspace</td></tr>
</table>

Content of group and list

XML Schema	JSON format	Examples	Meta type

Group
xs:complexType
	Object
	Contains a property per sub-node.
		Example for a simple-occurrence group.
		{
		   "road" : {"content" : "11 rue scribe"},
		   "zipcode" : {"content" : "75009"},
		   "country" : {"content" : "France"}
		}
			group

List
maxOccurs > 1
	Array
	Contains an array of all field occurrences represented by a JSON Object.
	Each object is represented as a node value [p 258].
		Example for a multi-occurrence field of the xs:int type.
		[
		   {"content": 0},
		   {"content": 1},
		   {"content": 2},
		   {"content": 3}
		]
		Example for a multi-occurrence group.
		[
		   {
		      "content":
		      {
		         "road": {"content": "11 rue
		 scribe"},
		         "zipcode": {"content": "75009"},
		         "country": {"content": "France"}
		      }
		   },
		   {
		      "content":
		      {
		         "road": {"content": "711 Atlantic
		 Ave"},
		         "zipcode": {"content": "MA 02111"},
		         "country": {"content": "United
		 States"}
		      }
		   }
		]
			Meta of simple type [p 260],
			or
			group

Pagination

This feature allows returning a limited and parameterizable number of data. Pagination can be applied to data of the following types: records, association values, selection node values and selectors. A context named pagination is returned only if it has been activated. This context allows browsing data similarly to the UI.

Pagination is activated by default.

See also *Select operation [p 223]*

Detailed information related to this context can be found hereafter:

JSON property	JSON format	Description	Required
`firstPage`	`String` or `null` (*)	URL to access the first page.	Yes (**)
`previousPage`	`String` or `null` (*)	URL to access the previous page.	Yes (**)
`nextPage`	`String` or `null` (*)	URL to access the next page.	Yes
`lastPage`	`String` or `null` (*)	URL to access the last page.	Yes (**)

> **Note**
>
> (*) Only defines if data is available in this context and not in the response.

> **Note**
>
> (**) Not present on selector.

Selector

By invoking the URL represented by the property `selector` on a field that provides an enumeration, this returns a JSON `Object` containing the properties:

- `rows` containing an `Array` of JSON `Object` where each one contains two entries, such as the returned `content` that can be persisted and the corresponding `label`. The list of possible items is established depending on the current context.
- (Optional) `pagination` containing pagination [p 262] information (activated by default).

```
{
   "rows": [
      {
         "content": "F",
         "label": "feminine"
      },
      {
         "content": "M",
         "label": "masculine"
      }
   ],
   "pagination": {
      "nextPage": null
   }
}
```

See also *includeSelector* *[p 227]*

Insert operation report

When invoking the insert operation with a record table, it can optionally return a report. The report includes a JSON `Object` that contains the following properties:

- `rows` contains a JSON `ObjectArray`, where each element corresponds to the result of a request element.
- `code` contains an `int` of the JSON `Number` type, and allows to know whether the record has been inserted or updated. This property is included if, and only if, the `updateOrInsert` parameter is set to `true`.

- `foreignKey` contains a `string` of the JSON `String` type, corresponding to the content to be used as a foreign key for this record. This property is included if, and only if, the parameter `includeForeignKey` is set to `true`.
- `label` contains a `string` of the JSON `String` type, and allows to retrieve the record label. This property is included if, and only if, the parameter `includeLabel` is set to `yes`.
- `details` containing a `string` of the JSON `String` type, corresponding to the resource URL. This property is included if, and only if, the parameter `includeDetails` is set to `true`.

```
{
   "rows": [
      {
         "code": 204,
         "foreignKey": "62",
         "label": "Claude Debussy",
         "details": "http://.../root/individu/62"
      },
      {
         "code": 201,
         "foreignKey": "195",
         "label": "Camille Saint-Saëns",
         "details": "http://.../root/individu/195"
      }
   ]
}
```

See also *Insert operation* *[p 231]*

Delete operation report

When invoking the delete operation, a report is returned. The report includes a JSON `Object` that contains the following properties:

- `deletedCount` containing an integer of the JSON `Number` type, corresponds to the number of deleted records.
- `occultedCount` containing an integer of the JSON `Number` type, corresponds to the number of occulted records.
- `inheritedCount` containing an integer of the JSON `Number` type, corresponds to the number of inherited records.

```
{
 "deletedCount": 1,
 "occultedCount": 0,
 "inheritedCount": 0
}
```

See also *Delete operation* *[p 236]*

Technical data

Each returned record is completed with the properties corresponding to its technical data, containing:

JSON property	JSON format	Description	Required
`creationDate`	`String`	Creation date.	Yes
`creationUser`	`String`	Creation user identifier.	Yes
`lastUpdateDate`	`String`	Last update date.	Yes
`lastUpdateUser`	`String`	Last update user identifier.	Yes

```
{
   ...
   "creationDate": "2015-12-24T19:00:53.158",
   "creationUser": "admin",
   "lastUpdateDate": "2015-12-25T00:00:00.001",
   "lastUpdateUser": "admin",
   ...
}
```

39.7 **Update modes**

The `byDelta` mode allows to ignore data model elements that are missing from the JSON source document. This mode is enabled (by default) through RESTful operations. The following table summarizes the behavior of insert and update operations when elements are not included in the source document.

State in the JSON source document	Behavior
The property does not exist in the source document	**If the `byDelta` mode is activated (default):** • For the `update` operation, the field value remains unchanged. • For the `insert` operation, the behavior is the same as when the `byDelta` mode is disabled. **If the `byDelta` mode is disabled through the RESTful operation parameter:** The target field is set to one of the following values: • If the element defines a default value, the target field is set to that default value. • If the element is of a type other than a string or list, the target field value is set to `null`. • If the element is an aggregated list, the target field value is set to an empty list value. • If the element is a string that differentiates `null` from an empty string, the target field value is set to `null`. If it is a string that does not differentiate the two, an empty string. • If the element (simple or complex) is hidden in the data services, the target value remains unchanged. Note The user performing the import must have the required permissions to create or change the target field value. Otherwise, the operation will be aborted.
The element is present and its value is `null` (for example, `"content": null`)	The target field is always set to `null` except for lists, in which case it is not supported.

39.8 Known limitations

Field values

The value of fields `xs:date`, `xs:time` and `xs:dateTime` does not contain a time zone associated with the JSON-primitive type.

CHAPTER **40**

XML import and export

This chapter contains the following topics:

40.1 Introduction

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

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

40.2 Imports

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

Insert and update operations

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

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

Set missing values as null

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

Ignore extra columns

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

Optimistic locking

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

40.3 **Exports**

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

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

The XML export options are as follows:

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

40.4 Handling of field values

Date, time & dateTime format

The following date and time formats are supported:

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

40.5 Known limitations

Association fields

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

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

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

Selection nodes

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

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

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

CHAPTER 41

CSV import and export

This chapter contains the following topics:

41.1 Introduction

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

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

41.2 Exports

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

The CSV export options are as follows:

Download file name	Specifies the name of the CSV file to be exported. This field is pre-populated with the name of the table from which the records are being exported.
File encoding	Specifies the character encoding to use for the exported file. The default is UTF-8.
Enable inheritance	In order to consider the inheritance [p 20] during a CSV export, the option has to be defined in the model. Specifies if inheritance will be taken into account during a CSV export. If inheritance is enabled, resolved values of fields are exported with the technical data that define the possible inheritance mode of the record or the field. If inheritance is disabled, resolved values of fields are exported and occulted records are ignored. By default, this option is disabled. **Note:** Inheritance is always ignored, if the table dataset has no parent or if the table has no inherited field.
User-friendly mode	Specifies whether exported values will be represented in a user-friendly way, or in a raw format. For example, in user-friendly mode, dates and numbers are formatted according to the user's locale, and foreign keys and enumerated values display their associated labels. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Include technical data	Specifies whether internal technical data will be included in the export. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Column header	Specifies whether or not to include column headers in the CSV file. • **No header** • **Label:** For each column in the spreadsheet, the CSV displays its label. Each label is localized according to the locale preference of the current session. If no user-friendly label is defined for a node, the technical name of the node is used.

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

41.3 Imports

Download file name

Specifies the name of the CSV file to be imported.

Import mode

When importing a CSV file, you must specify one of the following import modes, which will control the integrity of operations between the source and the target table.

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

File encoding

Specifies the character encoding to use for the exported file. The default is UTF-8.

Enable inheritance

In order to consider the inheritance [p 20] during a CSV import, the option has to be defined in the model.

Specifies whether the inheritance will be taken into account during a CSV import. If technical data in the CSV file define an inherit mode, corresponding fields or records are forced to be inherited. If technical data define an occult mode, corresponding records are forced to be occulted. Otherwise, fields are overwritten with values read from the CSV file. By default, this option is disabled.

Note: Inheritance is always ignored if the dataset of the table has no parent or if the table has no inherited field.

Column header

Specifies whether or not to include column headers in the CSV file.

- **No header**

	• **Label:** For each column in the spreadsheet, the CSV displays its label. Each label is localized according to the locale preference of the current session. If no user-friendly label is defined for a node, the technical name of the node is used. • **XPath:** For each column in the spreadsheet, the CSV displays the path to the node in the table.
Field separator	Specifies the field separator to use for exports. The default separator is comma, it can be modified under *Administration > User interface*.
List separator	Specifies the separator to use for values lists. The default separator is line return, it can be modified under *Administration > User interface*.

41.4 Handling of field values

Aggregated lists

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

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

Hidden fields

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

'Null' value for strings

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

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

41.5 Known limitations

Aggregated lists of groups

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

Terminal groups

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

As a consequence, some differences may appear during comparison after performing an export followed by an import. To ensure the symmetry of import and export, use XML import and export instead. See XML import and export [p 267].

Column label headers

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

Association fields

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

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

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

Selection nodes

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

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

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

CHAPTER **42**

Supported XPath syntax

This chapter contains the following topics:

42.1 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.

42.2 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')
```

Predicates on record label

```
osd:contains-record-label('dumesnil') or osd:contains-record-label('harmattan')
```

42.3 **Syntax specifications for XPath expressions**

Overview

Expression	**Format**	**Example**
XPath expression	*<container path>*[*predicate*]	`/books[title='xxx']`
<container path>	*<absolute path>* or *<relative path>*	
<absolute path>	`/a/b` or `//b`	`//books`
<relative path>	`../../b`, `./b` or `b`	`../../books`

Predicate specification

Expression	Format	Notes/Example
<predicate>	Example: A and (B or not(C)) A,B,C: *<atomic expression>*	Composition of: logical operators parentheses, not() and atomic expressions.

<atomic expression>	*<path><comparator><criterion>* or method(*<path>*,*<criterion>*)	`royalty = 24.5`
		`starts-with(title, 'Johnat')`
		`booleanValue = true`

<path>	*<relative path> or osd:label(<relative path>)*	Relative to the table that contains it: `../authorstitle`
<comparator>	*<boolean comparator>*, *<numeric comparator>* or *<string comparator>*	
<boolean comparator>	= or `!=`	
<numeric comparator>	`=` , `!=` , <, >, <=, or >=	
<string comparator>	`=`	
<method>	*<date method>*, *<string method>*, `osd:is-null` method or `osd:is-not-null` method	
<date, time & dateTime method>	`date-less-than`, `date-equal` or `date-greater-than`	
<string method>	`matches`, `starts-with`, `ends-with`, `contains`, `osd:is-empty`, `osd:is-not-empty`, `osd:is-empty-or-nil`, `osd:is-neither-empty-nor-nil`, `osd:is-equal-case-insensitive`, `osd:starts-with-case-insensitive`, `osd:ends-with-case-insensitive`, `osd:contains-case-insensitive`, or `osd:contains-record-label`	
<criterion>	*<boolean criterion>*, *<numeric criterion>*, *<string criterion>*, *<date criterion>*, *<time criterion>*, or *<dateTime criterion>*	
<boolean criterion>	`true` , `false`	
<numeric criterion>	An integer or a decimal	`-4.6`
<string criterion>	Quoted character string	`'azerty'`

Expression	Format	Notes/Example
<date criterion>	Quoted and formatted as 'yyyy-MM-dd'	`'2007-12-31'`
<time criterion>	Quoted and formatted as 'HH:mm:ss' or 'HH:mm:ss.SSS'	`'11:55:00'`
<dateTime criterion>	Quoted and formatted as 'yyyy-MM-ddTHH:mm:ss' or 'yyyy-MM-ddTHH:mm:ss.SSS'	`'2007-12-31T11:55:00'`

XPath 1.0 formula

It is possible to use an XPath 1.0 formula in the criterion value part of an atomic predicate expression (right-hand side).

For example, instead of `[./a=3]`, you may use the expression `[./a=(floor(./d)+ 2.0)]`.

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/tableA[ fkB = '123|2008-01-21' ]`, where the string "123|2008-01-21" is a representation of the entire primary key value.
- `/root/tableA[ fkB/id = 123 and date-equal(fkB/date, '2008-01-21') ]`, where this predicate is a more efficient equivalent to the one in the previous example.

- `/root/tableA[ fkB/id >= 123 ]`, where any number operator could be used, as the targeted primary key field is of type `xs:int`.
- `/root/tableA[ date-greater-than( ./fkB/date, '2007-01-01' ) ]`, where any date operator could be used, as the targeted primary key field is of type `xs:date`;
- `/root/tableA[ fkB = "" ]` is not valid as the targetted primary key has two columns.
- `/root/tableA[ osd:is-null(fkB) ]` checks if a foreign key is `null` (not defined).

Other

CHAPTER **43**

Inheritance and value resolution

This chapter contains the following topics:

43.1 Overview

The principle of inheritance is to mutualize resources that are shared by multiple contexts or entities. EBX5 offers mechanisms for defining, factorizing and resolving data values: *dataset inheritance* and *inherited fields*.

See also *Inheritance (glossary)* *[p 20]*

Dataset inheritance

Dataset inheritance is particularly useful when data applies to global enterprise contexts, such as subsidiaries or business partners.

Based on a hierarchy of datasets, it is possible to factorize common data into the root or intermediate datasets and define specialized data in specific contexts.

The dataset inheritance mechanisms are detailed below in Dataset inheritance [p 287].

Inherited fields

Contrary to dataset inheritance, which exploits global built-in relationships between datasets, inherited fields exploit finer-grained dependencies that are specific to the data structure. It allows factorizing and specializing data at the business entities-level.

For example, if the model specifies that a 'Product' is associated with a 'FamilyOfProducts', it is possible that some attributes of 'Product' inherit their values from the attributes defined in its associated 'FamilyOfProducts'.

> **Note**
>
> When using both inheritance in the same dataset, field inheritance has priority over the dataset one.

43.2 Dataset inheritance

Value lookup mechanism

The dataset inheritance lookup mechanism for values proceeds as follows:

1. If the value is locally defined, it is returned.
2. Otherwise, looks up the first locally defined value according to the built-in child-to-parent relationship of the dataset in the hierarchy of datasets.
3. If no locally defined value is found, the default value is returned.

   If no default value is defined, `null` is returned.

   Note: Default values cannot be defined on:

   - A single primary key node
   - Auto-incremented nodes
   - Nodes defining a computed value

Record lookup mechanism

Like values, table records can also be inherited as a unit by multiple contexts, but they can also be partially redefined (*overwritten*), defined for a specific context (*root mode*), or be *occulted*.

Formally, a table record has one of four distinct definition modes:

root record	Locally defined in the table and has no parent. This means that no record with the same primary key exists in the parent table, or that this parent is an occulting record.
overwriting record	Locally defined in the table and has a parent record. This means that a record with the same primary key exists in the parent table, and that this parent is not an occulting record. The overwriting record inherits its values from its parent, except for the values that it explicitly redefines.
inherited record	Not locally defined in the current table and has a parent record. All values are inherited.
occulting record	Specifies that, if a parent with the same primary key is defined, this parent will not be visible in table descendants.

See also *Dataset inheritance* *[p 101]*

Defining inheritance behavior at the table level

It is also possible to specify management rules in the declaration of a table in the data model.

43.3 Inherited fields

The specific inheritance mechanism allows fetching a value of a field according to its relationship to other tables.

Value lookup mechanism

The lookup mechanism for inherited fields values proceeds as follows:

1. If the value is locally defined, it is returned.
2. Otherwise, looks up the source record and value to inherit from, according to the properties that are defined in the data model.
3. The process is recursive; if the source node does not locally define a value, it is then looked up according to the inheritance behavior of the source node.

43.4 Optimize & Refactor service

EBX5 provides a built-in user service for optimizing the dataset inheritance in the hierarchy of datasets. This service performs the following functions:

- **Handles duplicated values:** Detects and removes all parameter values that are duplicates of the inherited value.
- **Mutualizes common values:** Detects and mutualizes the common values among the descendants of a common ancestor.

Procedure details

Datasets are processed from the bottom up, which means that if the service is run on the dataset at level *N*, with *N+1* being the level of its children and *N+2* being the level of its children's children, the service will first process the datasets at level *N+2* to determine if they can be optimized with respect to the datasets at level *N+1*. Next, it would proceed with an optimization of level *N+1* against level *N*.

> **Note**
>
> - These optimization and refactoring functions do not handle default values that are declared in the data model.
> - The highest level considered during the optimization procedure is always the dataset on which the service is run. This means that optimization and refactoring are not performed between the target dataset and its own ancestors.
> - Table optimization is performed on records with the same primary key.
> - Inherited fields are not optimized.
> - *The optimization and refactoring functions do not modify the resolved view of a dataset, if it is activated.*

Service availability

The 'Optimize & Refactor' service is available on datasets that have child datasets and have the 'Activated' property set to 'No' in their dataset information.

The service is available to any profile with write access on current dataset values. It can be disabled by setting restrictive access rights on a profile.

> **Note**
>
> For performance reasons, access rights are not verified on every node and table record.

CHAPTER **44**

Permissions

Permissions dictate the access each user has to data and actions.

This chapter contains the following topics:

44.1 Overview

Permissions are related to whether actions are authorized or not. They are also related to access rights, that is, whether an entity is hidden, read, or read-write. The main entities controlled by permissions are:

- dataspace
- Dataset
- Table
- Group
- Field

Users, roles and profiles

The definition and resolution of permissions make extensive use of the notion of *profiles*, which is the generic term applied to users or roles.

Each user can participate in several roles, and a role can be shared by several users.

Special definitions:

- An *administrator* is a member of the built-in role 'ADMINISTRATOR'.
- An *owner of a dataset* is a member of the *owner* attribute specified in the information of a root dataset. In this case, the built-in role 'OWNER' is activated when permissions are resolved in the context of the dataset.
- An *owner of a dataspace* is a member of the *owner* attribute specified for a dataspace. In this case, the built-in role 'OWNER' is activated when permissions are resolved in the context of the dataspace.

Permission rules

A permission rule defines the authorization granted to a profile for a particular entity.

User-defined permission rules are created through the user interface. See the section Defining permissions and access rights in the user interface [p 293].

Resolution of permissions

Permissions are always resolved in the context of an authenticated user session. Thus, the defined permission rules can take user's roles into account.

See also

Common principles about resolving permissions [p 297]

Resolving data access rights [p 298]

Owner and administrator permissions on a dataset

An administrator or owner of a dataset can perform the following actions:

- Manage its permissions
- Change its owner, if the dataset is a root dataset
- Change its localized labels and descriptions

Attention

While the definition of permissions can restrict an administrator or dataset owner's right to view data or perform certain actions, it remains possible for them to modify their own access, as they will always have access to permissions management.

Owner and administrator permissions on a dataspace

To be a *super owner* of a dataspace, a user must either:

- Own the dataspace and be allowed to manage its permissions, or
- Own a dataspace that is an ancestor of the current dataspace and be allowed to manage the permissions of that ancestor dataspace.

An administrator or super owner of a dataspace can perform the following actions:

- Manage its permissions of dataspace.
- Change its owner
- Lock it or unlock it
- Change its localized labels and descriptions

Attention

While the definition of permissions can restrict an administrator or dataspace owner's right to view data or perform certain actions, it remains possible for them to modify their own access, as they will always have access to permissions management.

Impact of merge on permissions

When a dataspace is merged, the permissions of the child dataset are merged with those of the parent dataspace if and only if the user specifies to do so during the merge process. The permissions of its parent dataspace are never impacted.

If some elements are hidden for the profile attempting to perform a merge, it will not be possible to proceed as the impacts of the merge on data will not be fully visible.

Important considerations about permissions

The following statements should be kept in mind while working with permissions:

- Resolution of custom display labels for tables ('defaultLabel' property) and relationships ('display' property) ignores permission, and fields usually hidden due to access rights restrictions will be displayed in such labels. As a result, these labels should not contain any confidential field. Otherwise, a permission strategy should also be defined to restrict the display of the whole label.
- To increase permissions resolution, a dedicated cache is implemented at the session-level; it only takes user-defined permission rules into account, not programmatic rules (which are not cached since they are contextual and dynamic). The session cache life cycle depends on the context, as described hereafter:
  - In the UI, the cache is cleared for every non-ajax event (i.e on page display, pop-up opening, etc.).
  - In programmatic procedures, the cache lasts until the end of the procedure, unless explicitly cleared (see below).

44.2 Defining permissions and access rights in the user interface

Each level has a similar schema, which allows defining permission rules for profiles.

dataspace permissions

For a given dataspace, the allowable permissions for each profile are as follows:

dataspace access	Defines access rights as read-write, read-only or hidden.
Restriction	Indicates whether this dataspace profile-permission association should have priority over other permissions rules.
Create a child dataspace	Indicates whether the profile can create child dataspaces from the current dataspace.
Create a child snapshot	Indicates whether the profile can create snapshots of the current dataspace.
Initiate merge	Indicates whether the profile can merge the current dataspace with its parent dataspace.
Export archive	Indicates whether the profile can export the current dataspace as an archive.
Import archive	Indicates whether the profile can import an archive into the current dataspace.
Close a dataspace	Indicates whether the profile can close the current dataspace.
Close a snapshot	Indicates whether the profile can close a snapshot of the current dataspace.
Permissions of child dataspace when created	Defines the permissions set upon the creation of a child dataspace.

Access rights on dataspaces

Mode	Authorization
Write	• Can view the dataspace. • Can access datasets according to dataset permissions.
Read-only	• Can view the dataspace and its snapshots. • Can view child dataspaces, if allowed by permissions. • Can view contents of the dataspace, though cannot modify them.
Hidden	• Can neither see the dataspace nor its snapshots. • If allowed to view child dataspace, can see the current dataspace but cannot select it. • Cannot access the dataspace contents, including datasets. • Cannot perform any actions on the dataspace.

Permissions on a new dataspace

When a user creates a child dataspace, the permissions of this new dataspace are automatically assigned to the profile's owner, based on the permissions defined under 'Permissions of child dataspace when created' in the parent dataspace. If multiple permissions are defined for the owner through different roles, resolved permissions [p 297] are applied.

Attention

Only the administrator and owner of a dataspace can define a new owner for the dataspace or modify associated permissions. They can also modify the general information on the dataspace and permissions of the users.

Furthermore, in a workflow, when using a "Create a dataspace" or "Create a snapshot" built-in script task, resolved permissions are computed using the owner defined in the script task's configuration, rather than the current session. This is because, in these cases, the current session is associated with a system user.

Permissions on dataset

For a given dataset, the allowable permissions for each profile are as follows:

Actions on datasets

Restricted mode	Indicates whether this dataset profile-permission association should have priority over other permissions rules.
Create a child dataset	Indicates whether the profile has the right to create a child data set of the current dataset.
Duplicate dataset	Indicates whether the profile has the right to duplicate the current dataset.
Change the dataset parent	Indicates whether the profile has the right to change the parent dataset of a given child dataset.

Actions on tables

The action rights on default tables are defined at the dataset level. It is then possible to override these default rights for one or more tables. The allowable permissions for each profile are as follows:

Create a new record	Indicates whether the profile has the right to create records in the table.
Overwrite inherited record	Indicates whether the profile has the right to overwrite inherited records in the table.
Occult inherited record	Indicates whether the profile has the right to occult inherited records in the table.
Delete a record	Indicates whether the profile has the right to delete records in the table.

Access rights on node values

Permissions defined on specific terminal nodes override their default access rights.

Read-write	Can view and modify node values.
Read	Can view nodes, but cannot modify their values.
Hidden	Cannot view nodes.

Permissions on services

An administrator or an owner of the current dataspace can modify the service default permission to either restrict or grant access to certain profiles.

Enabled	Grants service access to the current profile.
Disabled	Forbids service access to the current profile. It will not be displayed in menus, nor will it be launchable via web components.
Default	Sets the service permission to enabled or disabled, according to the default permission defined upon service declaration.

44.3 Common principles about resolving permissions

The resolution of access rights and permissions is always performed in the context of a user session according to the associated roles. In general, resolution of permissions is performed restrictively between a given level and its parent level. Thus, at any given level, a user cannot have a higher permission than the one resolved at a parent level.

Restriction policies

When a restrictive permission is defined for a profile, it takes precedence over the other permissions potentially granted by the user's other roles. Generally, for all user-defined permission rules that match the current user session:

- If some rules with restrictions are defined, the minimum permissions of these restricted rules are applied.
- If no rules having restrictions are defined, the maximum permissions of all matching rules are applied.

Permission resolution examples

Given two profiles *P1* and *P2* concerning the same user, the following table lists the possibilities when resolving that user's permission to a service.

P1 authorization	P2 authorization	Permission resolution
Enabled	Enabled	Enabled. Restrictions do not make any difference.
Disabled	Disabled	Disabled. Restrictions do not make any difference.
Enabled	Disabled	Enabled, unless P2's authorization is a restriction.
Disabled	Enabled	Enabled, unless P1's authorization is a restriction.

The same restriction policy is applied for data access rights resolution.

In another example, a dataspace can be hidden from all users by defining a restrictive association between the built-in profile "Profile.EVERYONE" and the access right "hidden".

44.4 Resolving data access rights

Resolving access rights defined using the user interface

Access rights defined using the user interface are resolved on three levels: dataspace, dataset and node.

If a profile is associated with restrictive access rights at a given level, the minimum of all restrictive rights defined at that level is resolved. If no restrictions are defined at that level, the maximum of all access rights defined at that level is resolved.

At any given level, the most restrictive access rights between those resolved at this level and higher levels are applied. For instance, if a user's dataset access permissions resolve to read-write access, but the container dataspace only allows read access, the user will only have read-only access to this dataset.

> **Note**
>
> The dataset inheritance mechanism applies to both values and access rights. That is, access rights defined on a dataset will be applied to its child datasets. It is possible to override these rights in the child dataset.

Access rights resolution example

In this example, there are three users who belong to the following defined roles and profiles:

User	Profile
User 1	• user1 • role A • role B
User 2	• role A • role B • role C
User 3	• user3 • role A • role C

The access rights of the profiles on a given element are as follows:

Profile	Access rights	Restriction policy
user1	Hidden	Yes
user3	Read	No
Role A	Read/Write	No
Role B	Read	Yes
Role C	Hidden	No

After resolution based on the role and profile access rights above, the rights that are applied to each user are as follows:

User	Resolved access rights
User 1	Hidden
User 2	Read
User 3	Read/Write

Resolving dataspace and snapshot access rights

At dataspace level, access rights are resolved as follows:

- If a user has several rights defined through multiple profiles:
  - If the rights include restrictions, the minimum of the restrictive profile-rights associations is applied.
  - Otherwise, the maximum of the profile-rights associations is applied.
- If the user has no rights defined:
  - If the user is an administrator or owner of the dataspace, read-write access is given for this dataspace.
  - Otherwise, the dataspace will be hidden.

Resolving dataset access rights

At the dataset level, the same principle applies as at the dataspace level. After resolving the access rights at the dataset level alone, the final access rights are determined by taking the minimum rights between the resolved dataspace rights and the resolved dataset rights. For example, if a dataspace is resolved to be read-only for a user and one of its datasets is resolved to be read-write, the user will only have read-only access to that dataset.

Resolving node access rights

At the node level, the same principle applies as at the dataspace and dataset levels. After resolving the access rights at the node level alone, the final access rights are determined by taking the minimum rights between the resolved dataset rights and the resolved node rights.

Specific access rights can be defined at the node level. If no specific access right is defined, the default access right is used for the resolution process.

> **Note**
>
> The resolution procedure is slightly different for table and table child nodes.

Special case for table and table child nodes

This describes the resolution process used for a given table node or table record *N*.

For each user-defined permission rule that matches one of the user's profiles, the access rights for *N* are either:

1. The locally defined access rights for *N*;
2. Inherited from the access rights defined on the table node;
3. Inherited from the default access rights for dataset values.

All matching user-defined permission rules are used to resolve the access rights for *N*. Resolution is done according to the restriction policy [p 297].

The final resolved access rights will be the minimum between the dataspace, dataset and the resolved access right for *N*.

CHAPTER **45**

Criteria editor

This chapter contains the following topics:

45.1 Overview

The criteria editor is included in several different areas of the user interface. It allows defining table filters, as well as validation and computation rules on data. This editor is based on the XPath 1.0 Recommendation.

Two types of criteria exist: atomic criteria and conditional blocks.

See also *Supported XPath syntax [p 279]*

45.2 Conditional blocks

Conditional blocks are made up of atomic criteria and other conditional blocks. They express a condition based on the criteria. The following types of blocks exist:

- **No criteria match**: None of the criteria in the block match.
- **Not all criteria match**: At least one criterion in the block does not match.
- **All criteria match**: All criteria in the block match.
- **At least one criterion matches**: One or more of the criteria match.

45.3 Atomic criteria

An atomic predicate is defined by a field, an operator, and an expression (either a value or an XPath formula).

Field	Specifies the field of the table to which the criterion applies.
Operator	Specifies the operator used. Available operators depend on the data type of the field.
Value	Specifies the value or expression. See Expression [p 302] below.
Code only	If checked, specifies searching the underlying values for the field instead of labels, which are searched by default.

Expression

The expression can either be a fixed value or a formula. When creating a filter, only fixed values are authorized. During creation of a validation or computation rule, a formula can be created using the wizard.

Known limitation: The formula field does not validate input values, only the syntax and path are checked.