



Product Documentation

EBX5 Version 5.4.0 Fix 002

Table of contents

User Guide

Introduction

1. How EBX5 works.....	9
2. Using the EBX5 user interface.....	11
3. Glossary.....	15

Data models

4. Introduction to data models.....	24
5. Using the Data Models area user interface.....	27

Implementing data models

6. Creating a data model.....	29
7. Data model configuration.....	31
8. Implementing the data model structure.....	33
9. Properties of data model elements.....	39
10. Data validation controls on elements.....	45
11. Working with existing data models.....	49

Publishing and versioning data models

12. Publishing data models.....	51
13. Versioning embedded data models.....	53

Data spaces

14. Introduction to data spaces.....	56
15. Main actions.....	59
16. Snapshot.....	63
17. Merging a data space.....	65
18. Permissions.....	69

Data sets

19. Introduction to data sets.....	74
20. Main actions.....	77
21. Custom views.....	83
22. Inheritance.....	87
23. Permissions.....	91

Workflow models

24. Introduction to workflow models.....	94
25. Main actions.....	97
26. Workflow modeling.....	101
27. User task.....	105
28. Workflow model publication.....	109

Data workflows

29. Introduction to data workflows.....	112
30. Using the Data Workflows area user interface.....	113
31. Work items.....	117

Managing data workflows

32. Launching and monitoring data workflows.....	121
33. Administration of data workflows.....	123

Data services

34. Introduction to data services.....	128
35. Main actions.....	131

Reference Manual

Integration

36. Using EBX5 as a web component.....	135
37. Built-in UI services.....	139
38. Data services.....	151

File import and export services

39. XML import and export.....	175
40. CSV import and export.....	179
41. Supported XPath syntax.....	183

Miscellaneous

42. Permissions.....	190
43. Inheritance and value resolution.....	199
44. Criteria Editor.....	203

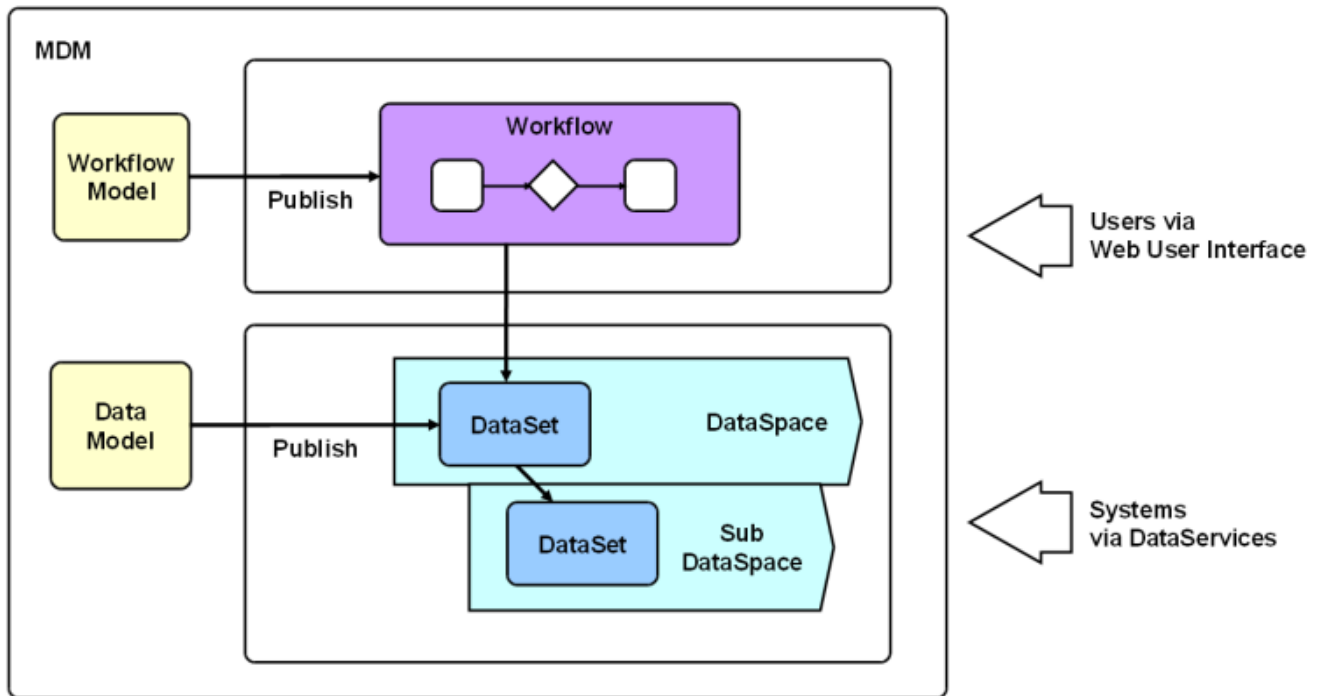
User Guide

Introduction

CHAPTER 1

How EBX5 works

Associated tools and notions



Master Data Management (MDM) is a way to better model, manage and ultimately govern your shared data. With data duplicated in many IT systems and shared by multiple business teams, having a single version and governance of your master data has become a critical issue.

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 a Master Data Management software that allows you to model any type of master data and apply governance thanks to rich features such as collaborative workflow, data authoring, hierarchy management, version control, and role-based security.

An MDM project on EBX5 starts by creating a data model. This is where you define tables, fields, links and business rules in order to describe your master data. Good examples are product catalogs, financial hierarchies, supplier lists or simply reference tables.

Your data model can then be published as a data set, which stores the actual content of your master data. A data set lives in a data space. A data space is a container which allows you to isolate your updates; this is very useful if you need to work on many parallel versions of your data, perform impact analysis or work in "staging areas".

Workflows are invaluable if you need to perform a controlled change management or approval of data through a step-by-step process involving multiple users. Once started, they send notifications to users of the number of work items available to them in a context of collaborative work.

Once everything is up and running, you can define data management processes, that you will express in the shape of workflow models in EBX5. This model will detail the tasks to be performed and any involved responsibility. It needs to be published, in order to become available for use in the shape of workflows.

Data services help integrating EBX5 to third-party systems (middleware), by allowing them to access data, or to manage data spaces and/or workflows.

Key words to understand are:

- [Data model](#)
- [Data set](#)
- [Data space](#)
- [Workflow model](#)
- [Workflow](#)
- [Data services](#)

Detailed definitions can be found in the [glossary](#).

CHAPTER 2

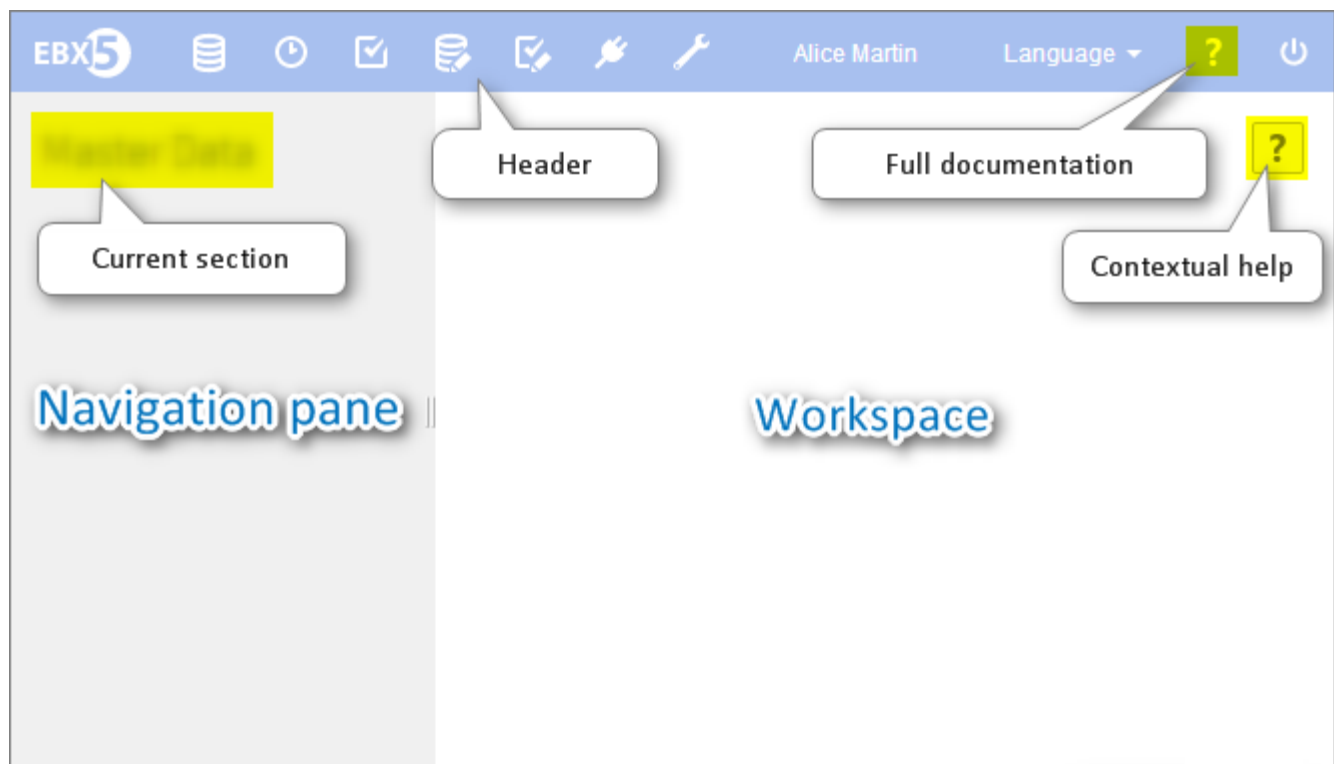
Using the EBX5 user interface

Finding your way around the EBX5 interface

The EBX5 interface is separated into several general regions, referred to as the following in the documentation:

- **Header:** Displays the user currently logged in, the user language preference selector, a link to the product documentation, and a logout button.
- **Menu bar:** The functional categories accessible to the current user.
- **Navigation pane:** Displays context-dependent navigation options.
- **Workspace:** Main context-dependent workarea of the interface.

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



Where to find help on EBX5

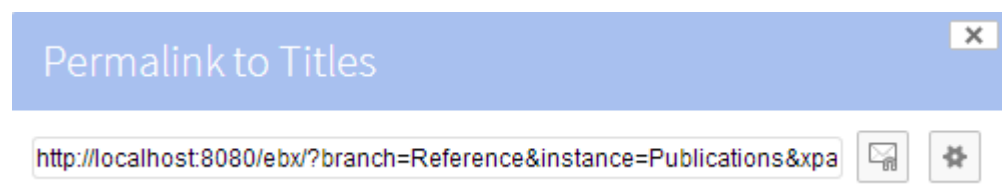
In addition to the full standalone product documentation, help is accessible in various forms within the interface.

Contextual Help

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



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



CHAPTER 3

Glossary

Governance

repository

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

See also [data space](#).

profile

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

See also [user](#), [role](#).

user

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

role

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

Whenever a role profile is specified in EBX5, the behavior resulting from that designation is applied to all users that are members of that role. For example, in a workflow model, a role may be specified when defining to whom work items are offered. As a result, all users belonging to that role can receive the same work item offer.

Data modeling

Main documentation section [Data models](#)

data model

A structural definition of the data to be managed in the EBX5 repository. A data model includes detailed descriptions of all included data, in terms of organization, data types, and semantic relationships. The purpose of data models is to define the structure and characteristics of data sets, which are instances of data models that contain the data being managed by the repository.

See also [data set](#).

Related concept [Data models](#).

field

A data model element that is defined with a name and a type. 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](#), [group](#), [table \(in data model\)](#), [validation rule](#), [inheritance](#).

Related concepts [Structure elements properties](#), [Controls on data fields](#).


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](#).

table (in data model)

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

Tables are represented by the icon .

See also [record](#), [primary key](#), [reusable type](#).

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](#).

reusable type

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

validation rule

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

data model assistant (DMA)

The EBX5 user interface includes a tool that aids the implementation of data models. It allows defining the structure of data models, creating and editing elements, as well as configuring and publishing data models.

See also [Data models](#).

Data management

Main documentation section [Data sets](#)

record

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

Records are represented by the icon .

See also [table \(in data set\)](#), [primary key](#).

table (in data set)

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

Tables are represented by the icon .

See also [record](#), [primary key](#).

data set

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

Data sets are represented by the icon .


See also [table \(in data set\)](#), [field](#), [group](#), [custom table view](#).

Related concept [Data sets](#).

inheritance

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

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

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

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

Inherited fields are represented by the icon .

custom table view

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

See also [hierarchical table view](#).

Related concept [Custom views](#).

hierarchical table view

A custom table view that offers a tree-based representation of the data in a table. Field values are displayed as nodes in the tree. A hierarchical view can be useful for showing the relationships between data. When creating a custom table view that uses the hierarchical format, dimensions can be selected

to determine the structural representation of the data. In a hierarchical view, it is possible to navigate through recursive relationships, as well as between multiple tables using foreign key relationships.

See also [custom table view](#).

Related concept [Hierarchies](#).

Data management life cycle

Main documentation section [Data spaces](#)

data space

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

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

Data spaces are represented by the icon .

See also [inheritance](#), [repository](#), [data space merge](#).

Related concept [Data spaces](#).

reference data space

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

See also [data space](#), [data space merge](#), [repository](#).


data space merge

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

Related concept [Merge](#).

snapshot

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

Snapshots are represented by the icon .

Related concept [Snapshot](#)

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](#), [transaction history view](#), [history profile](#).

history profile

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

See also [history profile](#).

table history view

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

transaction history view

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

Workflow modeling

Main documentation section [Workflow models](#)

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](#).

script task

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

Script tasks are represented by the icon .

See also [workflow model](#).

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](#).

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 .

data context

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

Data workflows

Main documentation section [Data workflows](#)

workflow publication

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

data workflow

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

See also [workflow model](#).


Related concept [Data workflows](#).

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](#).

Data services

Main documentation section [Data services](#)

data service

A method by which external systems can interact programmatically with EBX and the data managed within the EBX repository. Data services are standard web services, which can be used to access some of the same functionality as through the user interface.

Related concept [Data Services](#).

lineage

A mechanism by which user permission profiles can be established for non-human users, namely data services. The permission profiles established using lineage are used when accessing data via WSDL interfaces.

See also [data service](#).

Related concept [Generate WSDL for Lineage](#).

Data models

CHAPTER 4

Introduction to data models

Overview

What is a data model?

The first step towards managing data in EBX5 is to develop a data model. The purpose of a data model is to provide the detailed structural definition of the data that will be managed in the repository, in terms of organization, data types, and semantic relationships.

In order to implement a data model in the repository, you will first create a new data model, then define the details of the structure of its component table, field, and group elements, as well as their behavior and properties. When you have completed the entry or import of your data model structure in the repository, you will publish it to make it available for use by data sets. Once you have a publication of your data model, you and other users can create data sets based upon it to hold the data that is managed by the EBX5 repository.

Basic concepts used in data modeling

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

- [field](#)
- [primary key](#)
- [foreign key](#)
- [table \(in data model\)](#)
- [group](#)
- [reusable type](#)
- [validation rule](#)

Related concepts:

- [Data spaces](#)

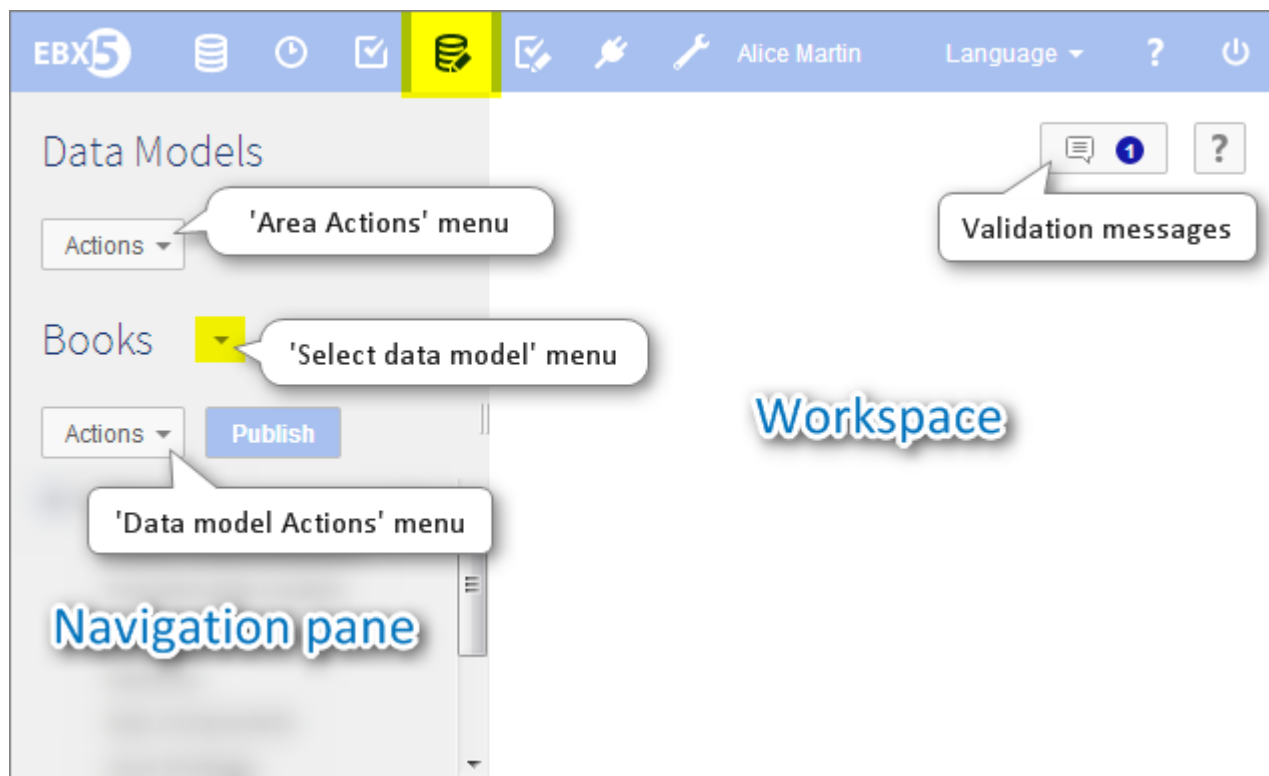
- [*Data sets*](#)

CHAPTER 5

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 **Modeling > Data Models** area. The EBX5 data model assistant (DMA) facilitates the development of data models.



The navigation pane is organized into the following sections:

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

Data model element icons

 [field](#)

 [primary key](#)

 [foreign key](#)

 [table](#)

 [group](#)


See also:

- [Implementing the data model structure](#)
- [Data model configuration](#)
- [Reusable types](#)

CHAPTER 6

Creating a data model

Creating a new data model

To create a new data model, select the ['Select data model'](#)  menu in the navigation pane, click the **Create** button in the pop-up, and follow through the wizard.

CHAPTER 7

Data model configuration

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'](#) menu for your data model in the navigation pane.

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 the rights to edit the data model's information and define its permissions.
Localized documentation	Localized labels and descriptions for the data model.

Permissions

To define the user permissions on your data model, select 'Permissions' from the [data model 'Actions'](#) menu for your data model in the navigation pane.

The configuration of the permissions of a data model are identical to the options for the permissions of a data set, as explained in [Permissions](#).

Data model properties

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

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

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 8

Implementing the data model structure

To work with the structural definition of your data model, select the data model you are working with using the **'Select data model'**  menu 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.

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

Add a new element beneath 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. You can then follow the element creation wizard to create the new element.

Note


The `root` element is always added upon data model creation.

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 fixed 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 except the `root` node can be deleted from the data structure using the down arrow  corresponding to its entry.

When a group or table that is not using a reusable type is deleted, the delete 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, use the up and down arrow buttons corresponding to its entry.

Note

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

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. You also have the option to have a reusable type defined upon the creation of a complex element (table or group). After your element has been created, the reusable type becomes available for creating more elements that will share the same structural definition and properties. Alternatively, you can convert tables and groups into reusable types after they have been created 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 consult the details of these included reusable types, however, they can only be edited locally in their original data models.

See [Included data models](#) for more information.

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.

Creating tables

During the creation of a table, you have the option to also create a reusable type based on the new table, or create the new table based on an existing reusable type. See [Reusable types](#) for more information.

At the end of the table element creation wizard, you may continue directly with the creation of a primary key field for the table. Every table requires the designation of at least one primary key field. If you choose not to create a new primary key field immediately, you can create one later from the 'Data structure' tree.


Creating groups

During the creation of a group, you have the option to also create a reusable type based on the new group, or create the new group based on an existing reusable type. See [Reusable types](#) for more information.

Creating primary key fields

At least one primary key is required for every table. After creating a table, you have the opportunity to immediately create one or more primary key fields for the new table.

At any point, you can create a primary key field for a table by adding a new primary key element under the table's entry in the 'Data structure' tree.

Besides creating a new field directly as a primary key, you can mark an existing field as a primary key using the down arrow  corresponding to its entry.

Creating or defining foreign key fields

Foreign key fields have the data type 'String'. You can create a foreign key field to a table in the current data model directly from the data structure, or convert an existing field of type 'String' into a foreign key.

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

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.

Creating catalogs of User Defined Attributes


A catalog of User Defined Attributes is required in order to create User Defined Attributes. When creating a new catalog of User Defined Attributes, you must specify its name. Once created, this catalog will be available for the creation of User Defined Attributes.

Creating User Defined Attributes

When creating a User Defined Attribute, you must specify a catalog of User Defined Attributes defined in the data model. This catalog contains attributes that will be associated with the new User Defined Attribute. The referred catalog can be changed once the User Defined Attribute has been created by modifying the property 'UDA catalog path' in the field's 'Advanced properties'.

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 using the down arrow  corresponding to its entry. Aside from no longer being part of the primary key, the field and its definition remain the same.

See [primary key](#) in the glossary.

CHAPTER 9

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](#)

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	<p>Minimum number of values for an element.</p> <p>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.</p>
Maximum number of values	<p>Maximum number of values for an element. When set to a value greater than '1', the element becomes multi-valued.</p> <p>As primary keys cannot be multi-valued, they must have this property set to '1' or 'undefined'.</p> <p>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.</p>
Validation rules	<p>This property is available for tables and fields in tables except <code>Password</code> 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.</p> <p>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.</p>

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	<p>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 criteria editor. This can be useful if the value depends on other values in the same record, but does not require a programmatic computation.</p> <p>The following limitations exist for computation rules:</p> <ul style="list-style-type: none">• Computation rules can only be defined on simple fields inside a table.• Computation rules cannot be defined on fields of type <code>OResource</code> or <code>Password</code>.• 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.

Advanced element properties

Advanced properties for fields

The following advanced properties are specific to fields.

Auto-increment

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

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

Inherited field

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

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

See [inheritance](#) in the glossary and [Inherited fields](#) for more information.

Advanced properties for tables

The following advanced properties are specific to tables.

Table

Primary key	<p>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.</p> <p>Each primary key field is denoted by its absolute XPath notation that starts under the table's root element.</p>
Indexes	<p>Defines the fields to index in the table. Indexing speeds up table access for requests on the indexed fields. No two indexes may contain the exact same fields.</p> <p>Index name: Unique name for this index.</p> <p>Fields to index: The fields to index, each represented by an absolute XPath notation starting under the table <code>root</code> element.</p>
Historization profile	<p>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.</p>
Label	<p>Defines the fields to provide the default and localized labels for records in the table.</p>
Specific filters	<p>Defines record display filters on the table.</p>
Rendering	<p>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.</p> <p>Enabled rendering for groups</p> <p>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.</p> <p>Default rendering for groups</p> <p>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.</p>

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.

Uniqueness constraints

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

Related concepts: [Data validation controls on elements](#)

CHAPTER 10

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](#)

Simple content validation

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

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

Advanced content validation

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

Foreign key constraint	
Table	Defines the table referenced by the foreign key. A foreign key references a table in the same data set by default. It can also reference a table in another data set in the same data space, or a data set in a different data space.
Mode	Location of the table referenced by the foreign key. 'Default': current data model. 'Other data set': different data set, in the same data space. 'Other data space': data set in a different data space.
Referenced table	XPath expression describing the location of the table. For example, <code>/root/MyTable</code> .
Referenced data set	Required if the table is located in another data set. The unique name of the data set containing the referenced table.
Referenced data space	Required if the table is located in another data space. The unique name of the data space containing the referenced table.
Label	Defines fields to provide the default and localized labels for records in the table.
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	<p>Defines an inclusive range of values that are not allowed for this field.</p> <p>Minimum excluded value: Lowest value not allowed for this field.</p> <p>Maximum excluded value: Highest value not allowed for this field.</p>
Enumeration filled by another node	Defines the possible values of this enumeration using a reference to another list or enumeration element.

Validation messages

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'.
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](#)

CHAPTER 11

Working with existing data models

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

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.

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'](#) 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](#) for more information on declaring modules.

To perform an import select 'Import XSD' from the [data model 'Actions'](#) 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'.

Duplicating a data model

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

Deleting a data model

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

Note

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

See [Publishing data models](#) for more information on the publication process.

CHAPTER 12

Publishing data models

About publications

Each data set based on an **embedded data model** in the EBX5 repository is associated with a publication of a data model, rather than directly to the data model itself. The first time you publish a data model using the **Publish** button in the navigation pane, a publication is created. Once the publication exists, data sets can be created based upon it.

Note

The **Publish** button is only displayed to users who have permission to publish the data model. See [Data model permissions](#) for more information.

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

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.

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](#) for more information on data model versions.

CHAPTER 13

Versioning data models

About versions


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

Accessing versions

To see the existing versions of your data model, select 'Manage versions' from the [data model 'Actions'](#) 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.

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.

Known limitations on data model versioning

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

Data spaces

CHAPTER 14

Introduction to data spaces

A data space contains data sets, and is used to isolate different versions of data sets from one another.

In EBX5, you may:

- create data spaces (see [creation](#)),
- label them and describe their purpose and content (see [information](#)),
- export data as an archive, or import data into a data space and validate it (see [import archive](#) and [export archive](#)),
- compare the content of two data spaces (see [actions](#)),
- save a copy of the current contents of this data space (see [snapshot](#)),
- apply the changes performed in a child data space to its parent data space (see [merge](#)),
- manage access rights for a given data space (see [permissions](#)),
- close a data space that is no longer required (see [closing](#)).

A data space is always created as a child of another data space, except the reference data space, which is the root of all data spaces in the repository.

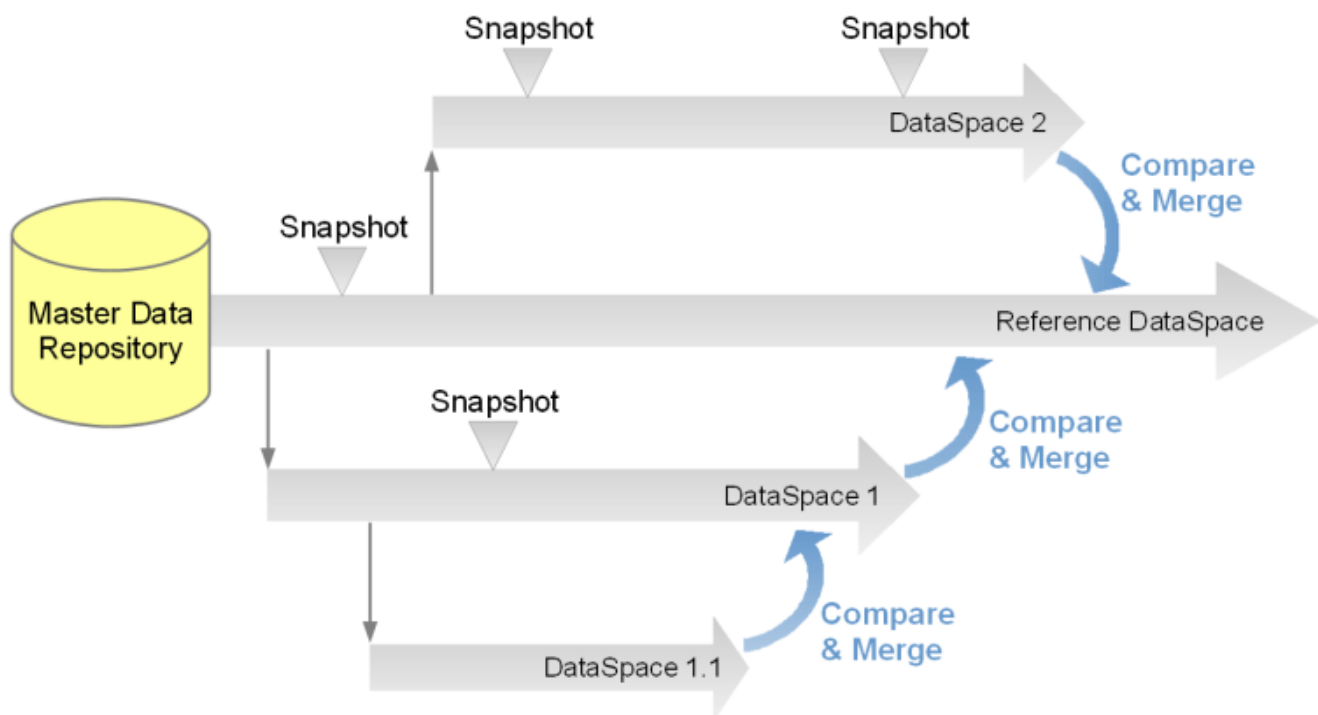
For detailed definitions of terms, refer to the [glossary](#).

Concepts

Data lifecycle is frequently complex. For example, a company needs to manage a current version of its data while working on several updates that will occur in the future. In addition, this company needs to keep track of its projects milestones as well.

EBX5 allows the creation and management multiple data spaces and snapshots. With the use of data space, it is possible to make concurrent updates on a data repository, to compare and merge them.

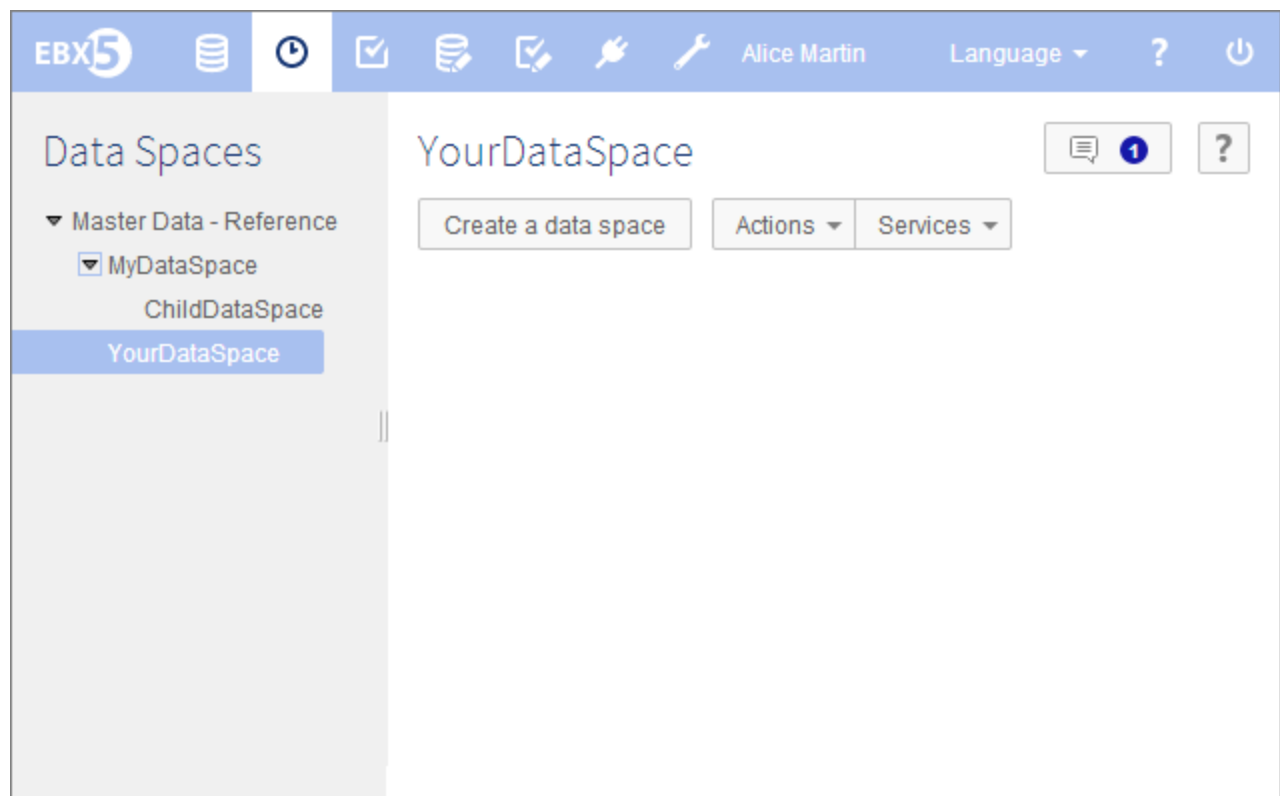
A snapshot allows you to keep the static state of a data space in case you need to revert to a stable version or detect subsequent updates.



Data space overview

Access within the interface

The navigation pane displays all existing data space in tree view, while the workspace is used to interact with a selected data spaces and lists its snapshots. To manage your data spaces, select *Data Spaces* in the menu bar.



Associated notions and tools

Snapshot	Frozen picture of a data space content at a point in time.
Reference data space	Ancestor of every other data spaces, which has no parent and cannot be merged.
Merge	Copy of the changes made to a child data space to its parent, which requires user arbitration in case of conflicts.
Relational data space	A relational data space is reserved for data models in relational mode. In this mode, data are managed more directly by the database and some features are not available: for example, it is not possible to create a snapshot or a child data space.

CHAPTER 15

Main actions

Creating a data space

You can create a new data space using the **Create a data space** button located in the workspace area. The new data space will be a child of the data space from which the creation was initiated and it will contain all the content of the parent at the time of creation.

The following information are required:

Identifier	Unique identifier for the data space.
Owner	Owner of the data space, allowed to modify its information and permissions. The owner is not necessarily the creator of the data space.
Label	Label and description associated with the data space, for which localized versions may be provided.

Editing a data space

Information

You can modify the information associated with a data space using the **Actions** button.

Current Owner	Owner of the data space, allowed to modify its information and permissions. The owner is not necessarily the creator of the data space.
Documentation	Label and description associated with the data space, for which localized versions may be provided.
Change owner	Whether the current owner of the data space is allowed to modify the <i>Current owner</i> value of the data space.
Change permissions	Whether the current owner of the data space is allowed to modify its information and permissions.
Child merge policy	<p>This merge policy only applies to merge processes initiated by a user. It does not apply to programmatic merges, for example, those performed by workflow script tasks.</p> <p>Merge policies are:</p> <ul style="list-style-type: none"> • Allows validation errors in result: Child data spaces 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 is valid.
Child data space sort policy	Defines the display order of child data spaces in data space trees. If not defined, the policy of the parent data space is considered. Default is 'by label'.

Validation

The content of a data space can be validated globally using the validation service at the data space level. This service can be accessed by clicking the *Actions* button in the workspace area.

Note

In order to use this service, a user must have permission to validate every data set contained in the data space.

Exporting an archive

The content of a data space can be exported to an archive by using the service accessible through the **Actions** button.

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

- **Name of the archive to create:** The name of the exported archive.
- **Export policy:** Required.

The default export policy is **The whole content of the data space**, which exports all selected data to the archive.

If you are working in a child data space, it may be useful to compare the data space with its parent and include the differences using a change set. There are two available export options to do so: **The updates with their whole content** and **The updates only**. The first option exports all the data and includes the change set, whereas the second option only exports the data if a difference exists in the change set. The granularity of the detection of differences is on the table level. The comparison screen the selection of differences to include in the change set.

- **Data sets to export:** The data sets that should be exported, in table format. For each data set, you can choose whether to export data values, permissions, and information.

Importing an archive

The content of an archive can be imported into a data space by using the service accessible using the *Actions* button.

If the selected archive contains a change set, you can decide whether you want to apply it. The comparison screen allows the selection of differences from the change set to import.

Merge

Changes in a data space can be applied to its parent data space by using the merge function, available through the *Actions* button. This process compares the parent and child data spaces being merged to find differences, which the user must verify.

See the [Merge](#) section for more information.

Snapshot

In order to save a read-only copy of your data space at a given point in time, you can create a snapshot. Snapshots can act as reference points if subsequent changes in the data space need to be undone.

See the [Snapshot](#) section for more information.

Permissions

It is possible for the owner and the administrator to define the access rights for a data space (read-only, write, or hidden) and fine tune them for all available actions in order to protect the data space and its content from being accessed and/or modified by unauthorized users.

See the [Permissions](#) section for more information.

Closing a data space

When a data space is no longer needed, it may be closed. This can be done using the *Actions* menu. Once closed, a data space can no longer be accessed. It may be reopened by an administrator if it has not yet been deleted.

CHAPTER 16

Snapshot

A snapshot is a complete, read-only copy of a data space at a given point in time.

Snapshot creation

A snapshot can be created using the *Create a Snapshot* button located in the workspace area.

The following information is required:

Identifier	Unique identifier for the snapshot.
Label	Label and description associated with the snapshot, for which localized versions may be provided.

Information

You can modify the information associated with a snapshot by clicking the *Actions* button and selecting *Information*.

Current Owner	Owner of the snapshot, allowed to modify its information and permissions. The owner is not necessarily the creator of the data space.
Documentation	Label and description associated with the snapshot, for which localized versions may be provided.
Change owner	Whether the current owner of the data space is allowed to modify the <i>Current owner</i> value of the snapshot.

View content

You can view the content of a snapshot using the *Actions* button.

Services

Several other services are accessible using *View or edit content* under *Actions*.

Validation

The content of a snapshot can be validated globally using the validation service at snapshot level.

Note

In order to use this service, a user must have permission to validate every data set contained in the snapshot.

Compare

This service allows the comparison of the contents of the current snapshot with another data space or snapshot. A side-by-side comparison view summarizes the differences found.

Export

This service allows the export a snapshot's content to an archive.

When exporting an archive, the following information is required:

- **Name of the archive to create:** The name of the exported archive.
- **Data sets to export:** The data sets that should be exported, in table format. For each data set, you can choose whether to export data values, permissions, and information.

Closing a snapshot

When a snapshot is no longer needed, it can be closed. The owner of the snapshot, or an authorized user, can close the snapshot using the *Actions* menu. Once a snapshot is closed it can no longer be accessed. It may be deleted by an administrator if it has not yet been purged.

CHAPTER 17

Merging a data space

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

1. Both the parent and child data spaces are locked to all users. The locks remain for the duration of the merge operation. This means their contents can be read, but they cannot be modified in any way.

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

2. Changes that were made in the child data space since its creation are integrated into its parent data space.
3. The child data space is closed.
4. The parent data space is unlocked.

Initiating a merge

To merge a data space into its parent data space:

1. Select that data space in the navigation pane of the Data Spaces area.
2. In the workspace pane, select **Merge data space** from the **Actions** menu.

Reviewing and accepting changes

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

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

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

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

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

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

Types of modifications

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

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

Types of conflicts

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

Conflicts are categorized as follows:

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

Finalizing a merge

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

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

If the child merge policy of the parent data space in your merge is the default policy, the merge is carried out regardless of whether there are validation errors in the resulting data space.

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

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

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

Setting the child merge policy of a data space

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

Note

In web component mode, the behavior of the child merge policy is the same as described; the policy defined in the parent data space is automatically applied when merging a child data space. However, this setting is ignored during programmatic merge, which includes script tasks in data workflows.

See also: [Child merge policy](#)

Abandoning a merge

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

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

You may unlock a data space by selecting that data space in the navigation pane, and clicking on the **Unlock** button in the workspace. Performing the unlock from the child data space unlocks both the child

and parent data spaces. Performing the unlock from the parent data space only unlocks the parent data space, thus you need to unlock the child data space separately.

CHAPTER 18

Permissions

Data space permissions can be accessed using the *Actions* button in the workspace area.

A permission is always attached to a profile.

Data space permissions

General permissions

- **Data space id:** indicates the data space on which the permissions will apply.
- **Profile selection:** indicates the profile affected by the rule.
- **Restriction policy:** indicates if the permissions defined here restricts the ones defined for other profiles. See also [restriction policy](#).
- **Data space access:** indicates the global access permission on the data space (read-only, write or hidden). See below:

Read-only

- Visualize the data space and its snapshot. See the children data space, according to the rights applying on each child.
- Visualize the contents of the data space without being able to modify the content, provided that the content rights give you access to it.

Write

- See the data space.
- Access data sets depending on the rights you have on them.

Hidden

- Neither the data space nor its snapshots can be seen.
 - From a child data space, the current data space can be seen but not selected.
 - No access to the data space content.
 - Not one action can be performed on the data space.
-

Choice of allowable actions

- **Create a child data space:** indicates if the profile can create a child data space.
- **Create a Snapshot:** indicates if the profile can create snapshot from the data space.
- **Initiate merge:** indicates if the profile can merge the data space with its parent.
- **Export archive:** indicates if the profile can access the export service.
- **Import archive:** indicates if the profile can access the import service.
- **Close data space:** indicates if the profile can close the data space.
- **Close snapshot:** indicates if the profile can close the data space's snapshot.
- **Rights on services:** specifies the access permissions for services. A service is not accessible to a profile if it is crossed-out.
- **Permissions of child data space when created:** specifies the access permissions that will be present for a newly created child data space.

Snapshot permissions

Initial snapshot permissions

The data space permissions apply on its initial snapshot.

Data space snapshot permissions

The data space permissions apply on its snapshots.

Data sets

CHAPTER 19

Introduction to data sets

Once a model has been created and published for your reference data, you may want to store and manage your reference data. In order to, you can:

- create a new data set in a data space (see [creation](#)),
- select a data space to work in, a data set to work with, then navigate through groups, tables and fields using the tree view (see [actions](#)),
- read, add, update or delete records in a data set table (see [tables](#)),
- access and manage a table by defining custom views and/or hierarchies (see [advanced viewing modes](#)),
- start working from an existing set of data by copying them (see [inheritance](#)),
- define who may or may not access some data (see [permissions](#)).

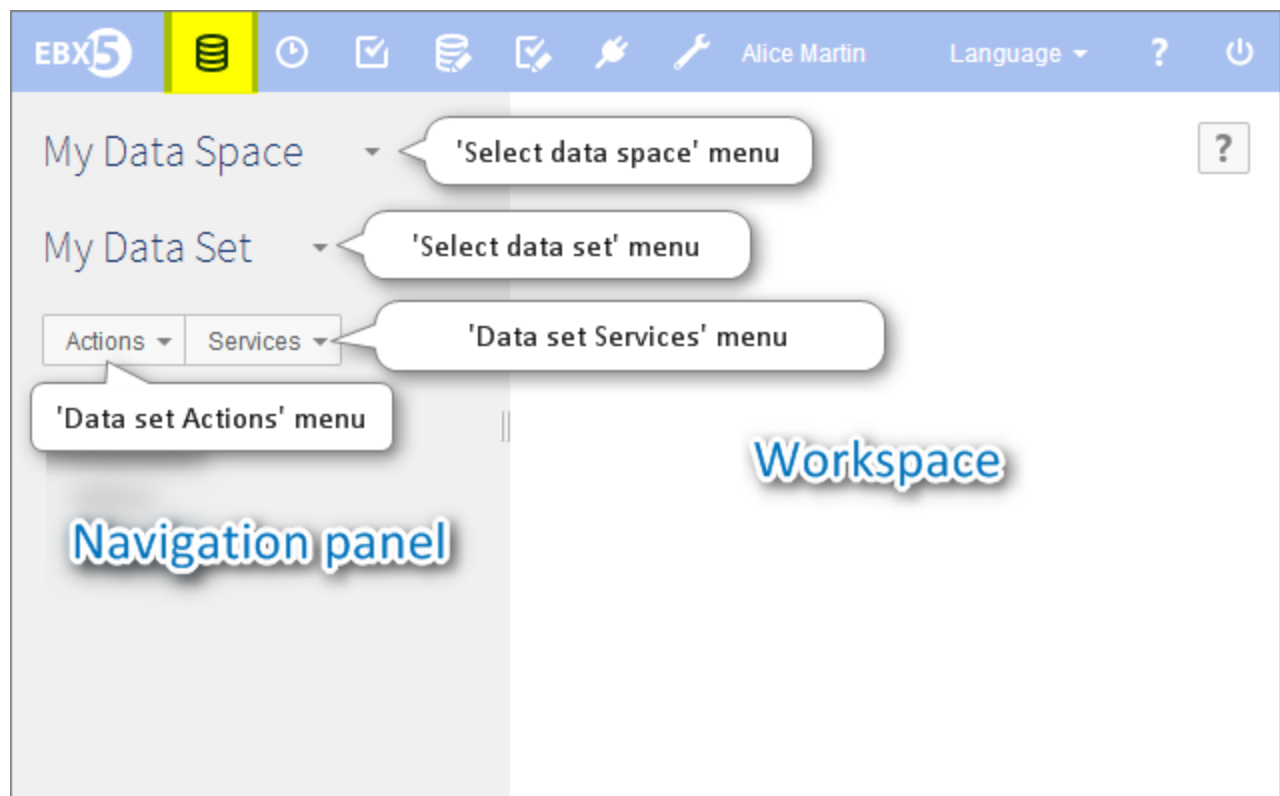
A data set corresponds to data displayed as tables or hierarchies, and that can be filtered using multi-criteria views. Access to the content of a data set can be restricted using permissions rules for users and role (that is, a group of users).

For a better understanding of those concepts, feel free to visit our [glossary](#).

Data set overview

Access within the interface

To interact with your data, please select *Data* in the menu bar. Then get started by opening the data space from the top section in the navigation pane.



Users can select a data set using the navigation pane. Two buttons are displayed, the first one allows you to select a data space while the second one allows you to select a data set. Once a data set has been selected, its structure appears in the navigation pane and can be used to select a group or a table. Values held by those elements are displayed in the workspace.

Associated notions and tools

Once in the *Data* section, you will come across the following notions and tools:

Data space	Data container, whose content can be updated in total isolation from other items close by.
Data set	Group of data predefined in a data model as having a common use or purpose.
Tree view	Way to visualize the content of a data set in the navigation pane.
Target table	Studied table, whose dependencies to others tables we wish to visualize. It is the last and most specific level of the hierarchy, like the leaves of a tree.
Record	Group of fields forming a unit of information entered by the user in a given data set, appearing as the content of a row in a table.
Hierarchy	Tree-based representation of a succession of dependencies between tables. It can be balanced, unbalanced, ragged or network.
Recursive relationship	Occurrence of a dependency link between two entities of the same dimension level.
Dimension	Possible axe of analysis of a target table, including various dimension levels (as an example: products, families, categories, etc.).

See also:

- [Data model](#)
- [Data space](#)
- [Workflow model](#)
- [Data workflows](#)
- [Data services](#)

CHAPTER 20

Main actions

Creating a data set

A data set can be created using the *Create a data set* button, accessible through the data set selector. If no data sets exist in the selected data space, the button is directly displayed in the navigation pane. You can then use the wizard to create a data set.

Information

The data set's information can be edited using the *Actions* button from the navigation pane. The following information can be edited:

- **Owner:** User allowed to edit the data set's information and permission rules. The owner is not necessarily the creator of the data set.
- **Documentation:** Label and description associated with the data set according to languages, for which localized versions may be provided.
- **Activated:** Data set activation allowing validation rules to be checked.

Editing a data set

Navigating within a data set

A data set may contain tables, fields and/or groups. To access them, use the tree view in the navigation pane. Their content is displayed in the workspace.

Permissions settings

To find out how to set customized permissions for each potential user, see the [permissions](#) section.

Validating a data set

To validate a data set 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 set in order to be able to rerun a full validation from scratch.

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

Data set tables

The content of a table is displayed in the workspace area, which is also where the user can interact with it.

Editing a record

Create	A new record can be created using the "+" button located on the top left of the table. A form is displayed to input the values. Mandatory data are indicated by a red asterisk.
Edit	A record can be edited by double-clicking on it. The displayed form allows you to edit the record and the <i>Revert</i> button allows you to reload the form without submitting any of the changes made.
Duplicate	<p>A record, that has been selected using the check box, can be duplicated using the <i>Actions</i> button.</p> <p>A form appears with its value already filled in. The primary key must then be changed in order to create this new record, unless this primary key is generated (auto-incremented value).</p>
Delete	One or several records, that have been selected using the check boxes, can be deleted using the <i>Actions</i> button.
Compare	<p>Two records, that have been selected using the check boxes, can be compared using the <i>Actions</i> button.</p> <p>Note: The content of complex terminal nodes, such as aggregated lists and user defined attributes, are not subject to comparison during this process. The compare service ignores any differences between the values of the complex terminal nodes in the records.</p>

Record import/export

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](#)

- [XML Services](#)


Sorting data

Sorting criteria control the order in which records are presented. They can be defined through the *View* button. The default order is by ascending primary key. The *Reset* option allows you to cancel any modifications.

It is defined by a column name and an order (ascending, lowest to highest value, or descending, highest to lowest value). A sort criteria can be added and removed using the *Add* and *Remove* links. The sort order of a criteria can be changed by clicking either on the *ascending* or on the *descending* links.

The order between criteria can be set by using the button located on the right of the criteria list.

Searching and filtering data

Specific tools are offered to search for records inside the current view. They can be accessed using the icon , which displays the filters panes.

When criteria are defined for a search or filter, a checkbox appears in its title bar to apply the filter. Unchecking this box removes the filter.

Note

Applying a custom view will reset and remove all applied filters.

Typed search

In simple mode, the typed search tool allows adding type-contextual search criteria on 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 computation of the search results.

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.

- If the entered text entered contains one or several words without wildcard characters (* or ?), matching fields must contain all the specified words. The words being quoted (for example "aa bb") are considered as one word.
- Usual wildcard characters are available: * (any text) or ? (any character). For performance reason, only one of these character can be entered.
- Wildcard characters can be considered as usual characters by escaping them with '\', for example '*'.
 *'

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

On big tables, it is recommended to select only one table field and in the case the field type is not of the string type, to try to match the format type:

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

Case sensitive options: search is sensitive to the case (that is, low-case and upper-case are distinguished).

Validation messages filter

The validation messages filter allows you to view records according to their status as of the last validation performed. You may select to view records of the levels 'Error', 'Warning', or 'Information'.

Note

This filter is only applied to the 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.

Custom views

You can customize the presentation of a table by creating custom views and applying them. By using the *View* button, it is possible to select an existing custom view or to create a new one. Once you applied

an existing view, it can be set as the *default view* for the table. By clicking on *withdraw all views*, anyone can deactivate all custom views and/or hierarchy previously applied.

See also: [Custom views](#)

CHAPTER 21

Custom views

There are two types of custom views:

- **Simple tabular view:** filters table entries according to given criteria
- **Hierarchical view:** organizes and links information from different tables in a tree view

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

View publication

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

This view publication will be usable by users who belong to the authorized profiles specified in the view (see below).

View description

This form allows you to specify the information related to the custom view that are common to all modes.

Owner	Name of the owner of the custom view, who can manage and modify it.
Documentation	Label and description associated with the custom view according to languages.
Authorized profiles	Profiles authorized to use the custom view.
View mode	Mode of the custom view (see above).

Simple tabular view

Simple tabular views offer the possibility to define criteria to filter records and to select the columns to be displayed.

Search criteria

This field allows you to define criteria which will be used to filter the records.

See also: [Criteria editor](#)

Sort criteria

This field allows you to define criteria which will be used to sort the records.

Column display

This field allows you to select, using the check boxes, the columns to display in the custom view.

Hierarchies

A hierarchy is a tree-based data representation allowing to highlight relationships between data. It can be structured on several levels, called dimension levels. Furthermore, it is possible to define criteria on levels in order to filter records.

Hierarchy dimension

A dimension is what defines the various dependencies in a hierarchy (example: products per categories). Starting from a target table, select link dependencies step by step until the last one.

Each dependency link becomes a new dimension level.

Hierarchy configuration

This forms allows you to configure the hierarchy levels. For each level, you can configure the labels and specify criteria to filter the records.

Label

In the tree, records are named using labels, written in various languages. Record fields can be easily inserted in a label using the assistant (use the button on the right of the field to access it).

Filter

This form allows you to define criteria, which will be used to filter the records.

See also: [Criteria editor documentation](#)

Ordering

It is possible to specify an ordering field which allows the user to position children nodes at will. An ordering field has to be an integer and in hidden mode, in the data model.

Except in the case where the ordering field is in 'read-only' mode or where the hierarchy is filtered, each field can be repositioned.

If no ordering field is specified, then children nodes are sorted according to the labels alphabetical order.

CHAPTER 22

Inheritance

Usually people create one data set from one data model. EBX5 allows you to create additional data sets, branched off this initial root data set. Those child data sets inherit from their parent. Several levels of inheritance can be created.


This can be used to adapt master data to various context, like geographical areas and/or countries.

Note

Standard behavior is to forbid data set inheritance. Be careful to explicitly activate it when creating your data model.

See also: [Data model configuration](#)

Data set tree

Once the root data set has been created, a child data set can be created using the  button located in the data set selection screen of the navigation pane. The user is then asked to provide a unique name for it. He can also, if he wishes to, give it an optional localized label and description.

Note:

- A data set cannot be deleted if it has children data set. Its children must be deleted first.
- If a child data set is duplicated, the newly created data set will be inserted into the existing data set tree as a sibling from the duplicated data set.

Value inheritance

When a child data set is created, it is inheriting all field's and table record's values from its father. A field or a record can either inherit its value or overwrite it.


Overwritten values uses the default style and inherited values can be identified by a mark on the top left corner of the cell.

The  button can be used to indicate if a value is inherited or overwritten.

Record inheritance

A table in a child data set will inherit the record from the table located in its ancestor. It is possible to edit and delete those records, new records can also be created and will be inherited in child data set. Several states are defined to differentiate those records.

Root	A root record is a record created in the current data set that doesn't exist in the ancestors data set. It will be inherited in the children data set.
Inherited	An inherited record is defined in one of the ancestor data set of the current one.
Overwritten	An overwritten record is an inherited record, whose values are edited in the current data set. The overwritten values will be inherited in the children data set.
Occulted	An occulted record is an inherited record which is deleted in the current data set. It will still appear in the current data set as a crossed out record but it will not be inherited in the children data set.

The  button indicates that a record is inherited. It can also be used for changing its state to overwritten or inherited. Please note it is the same button that allows you to change the status of an occulted record to inherited.

The following tables sum up what happens when creating, editing or deleting a record depending on its initial state.

State \ Operation	Create	Edit value	Delete
Root	Normal creation of a record. The newly created record will be inherited in children data set.	Normal editing of a record. The new values will be inherited in children data set.	Normal deletion of a record. The record will disappear from the inheriting data set.
Inherited	If a record is created using the primary key of an existing inherited record, the record state will be changed to overwritten and its value modified according to the one submitted at creation.	An inherited record must be declared as overwritten in order to edit its value.	Deleting an inherited record sets its state to occulted.
Overwritten	Not applicable. It is impossible to create a new record if the primary key is already used.	An overwritten record can be returned to the inherited state, but its specific value will be lost. Values from an overwritten record can be set to inherit or can be specified.	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.	An occulted record cannot be edited.	Not applicable. An occulted record is already considered as deleted and as such cannot be deleted further.

CHAPTER 23

Permissions

Data set permissions can be accessed using the *Actions* button in the navigation pane.

Permissions are always granted through a profile.

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. For more details, see [restriction policy](#).

Data set actions

This section specifies action permissions on the data set.

Create a child data set	Indicates if the profile can create a child data set. Inheritance also has to be activated in the data model.
Duplicate the data set	Indicates if the profile can duplicate the data set.
Delete the data set	Indicates if the profile can delete the data set.
Activate/Inactivate the data set	Indicates if the profile can modify the <i>Activated</i> property in the data set information .
Create a view	Indicates if the profile can create custom views and hierarchies.

Tables policy

This section specifies the default permissions for all the tables. Specific permissions can also be defined for a table by clicking on the *Add an occurrence* button.

Create a new record	Indicates if the profile can create records in the table.
Overwrite inherited record	Indicates if the profile can overwrite inherited records in the table. This permission is useful when using data set inheritance.
Occult inherited record	Indicates if the profile can occult inherited records in the table. This permission is useful when using data set inheritance.
Delete a record	Indicates if the profile can delete records in the table.

Values access policy

This section specifies the default access permissions for all the nodes of the data set and allows the definition of permissions for specific nodes. The default access permission is used if no specific permission has been granted for a node.

The specific policy selector allows granting a specific access permission for a node. The buttons *Hidden*, *Read-Only* and *Write* set the corresponding access permission to the selected node.

It is possible to remove a specific access permission using the *(default)* button.

Rights on services

This section specifies the access permissions for services. A service is not accessible to a profile if it is crossed-out.

Workflow models

CHAPTER 24

Introduction to workflow models

Collaborative work is a powerful way to produce, update, merge and validate data in a business. However it is not always easy to get people from various places and with different skills, to work together in order to meet in time a common deadline.

In this, you may find workflow modeling to be quite a useful tool. Indeed it will allow you to define data management processes involving your collaborators. To do so, you will need to:

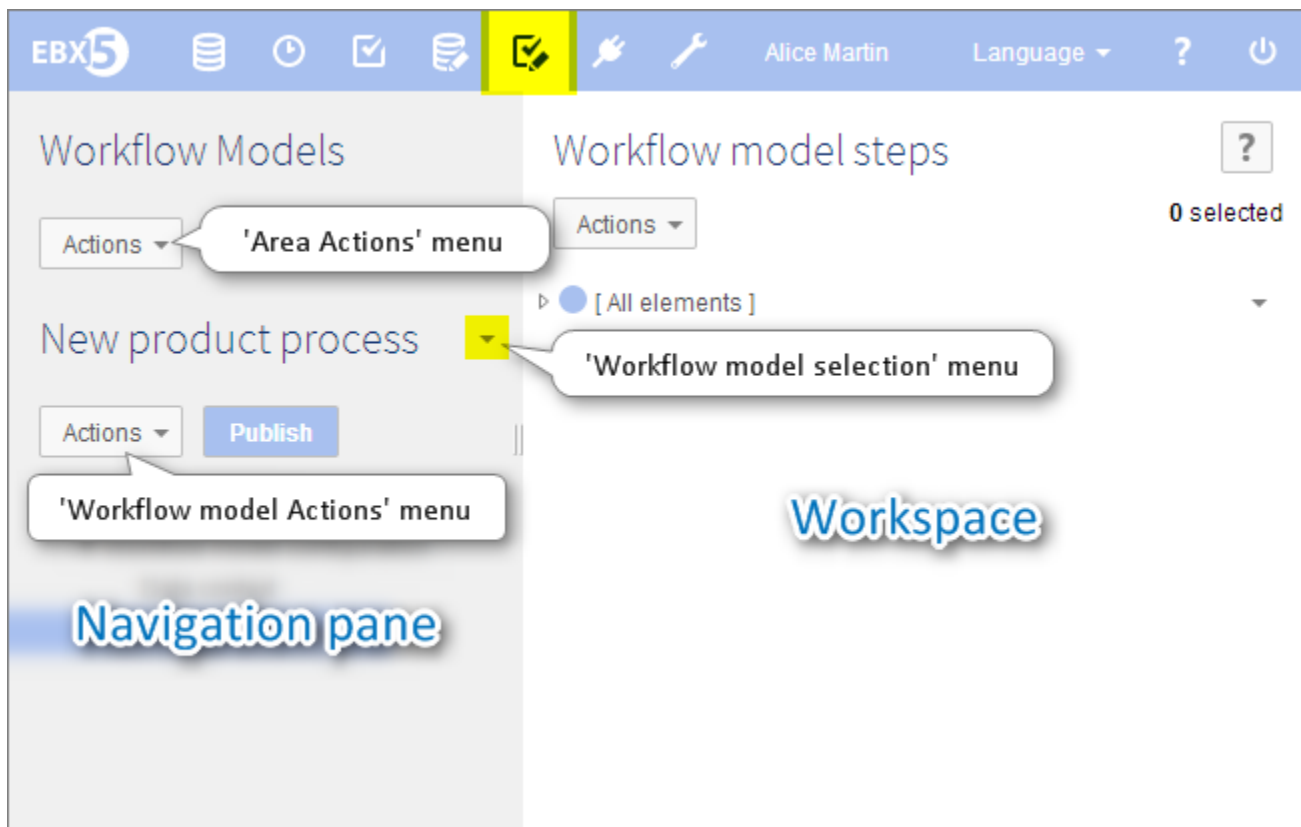
- define tasks to be performed by a user (see [user task](#)) or automatically by the system (see [script task](#)),
- specify everybody's responsibilities (see [creation](#)),
- send notification emails to colleagues, whose involvement is required (see [notification](#)),
- know the limits of your model; what can and cannot be done (see [limitations](#)),
- publish your model as a workflow to get it going (see [publication](#)).

A workflow model defines the tasks to be performed and the involved responsibilities. It can then be published as a workflow publication. It is a succession of two kinds of tasks: *script* and *user tasks*, with the possibility to have a conditional fork between two of them.

To find out a word's meaning, you can check it out in our [glossary](#).

Workflow model overview

Access within the interface



Associated notions and tools

Script task / <i>Script</i>	No user are involved in this kind of task. It can be, for instance, an automatic merge, a data space snapshot creation, etc.
User task	It involves at least one user and eventually several, who have to fulfil work items.
Work item	Basic task carried out by the user, it has been allocated to, and whose execution allows the workflow to move forward.
Conditional fork / <i>Condition</i>	It decides from the result of previous tasks, which route has to be taken during the workflow. This switch is made from two derivations. For instance, one can continue the workflow normally, while the other returns to a previous task; or there could be two different parallel routes.
Data context	It is a variable hosting temporary input/output data related to a workflow execution. Its aim is to facilitate the transfert of key information from one step to another (example: a data space name created in step 1 and reused in step 2 for another purpose).

CHAPTER 25

Main actions

Workflow modeling

Conception

First, you need to think what kind of evolution your data will regularly go through, and how many people are involved in those changes.

Second, you will have to put all this in a drawing, that will be translated as a workflow model in EBX5.

To find out how to proceed in the interface, please read the [creation](#) section.

Validation

Once drafted in the interface, your workflow model will have to be validated, in order to check there is no errors in its code. If there is any, they will have to be solved before you can publish and use your model.

Publication

Once validated, a workflow model is ready for publication. By publishing it, you generate a frozen picture of it, which becomes available for use in the workflow section.

Limitations

The following functionalities are not supported:

- **Parallel tasks**, two routes at least running simultaneously.
- **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.

Generic message templates

Notification emails can be sent to inform specific users of a given event during the execution of a workflow.

Templates can be defined and re-used in every workflow model by selecting the entry 'Message templates' in the global Workflow Models 'Actions' menu.

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

When creating a new template, two fields are required:

- **Label & Description:** specify the label and description associated with the template according to languages.
- **Message:** specify the object of the email and its body according to languages.

The message can be enhanced with data context variables such as: `${variable.name}`. System variables are also available:

Variables Syntax	Associated Meaning
system.time	System time.
system.date	System date.
workflow.lastComment	Last comment on previous user task.
workflow.lastDecision	Last decisions on previous 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 list, by means of the web component API.
workflow.currentStep.label	Label of the current step (script, condition or user task).
workflow.currentStep.description	Description of the current step (script, condition or user task).

This is a sample message: *"Today at \${system.time}, a new work item has been offered to you".*

This could give you the following email: *"Today at 15:19, a new work item has been offered to you".*

Workflow model evolution

Editing

Improvements can be made to an existing model. However it is advisable, when publishing it, to name it differently from previous ones.

History

The snapshots history of a workflow model can be managed through the history feature available using the *Actions* button.

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

Deletion

A workflow model can be deleted. However any version published previously will remain accessible in the workflow section. Furthermore, if you create a new workflow model with an identical name, a warning message will ask you if you wish to replace the previous publication.

CHAPTER 26

Workflow modeling

Creation

A workflow model can be created from the *Modeling / Workflow Models* section. The creation assistant only requires a name. This name must be unique so as to identify each workflow model specifically.

Therefore, the workflow model is created and the steps of the workflow model are initialized by the presence of an initial transition. The purpose of the workflow model is to describe the sequence of steps beyond this initial transition.

Inheritance

It is possible to use the inheritance mechanism with the following limitations:

- It is not possible to add steps at the end nor at the beginning.
- It is not possible to insert steps.
- It is not possible to change the relations between steps.

Steps

A workflow model defines steps corresponding to the different operations and conditions. The following sections introduce the different kinds of step.

Conditions

Library conditions

Furthermore, a label and a description can be specified according to languages.

EBX5 provides built-in conditions, thanks to which you can check if:

- Data is valid (data space, data set or table).
- A data space has been modified.
- What was the last user task accepted.
- That two values are equal.
- That a value is null or empty.

Some conditions are marked as "deprecated" because they are not compatible with I18N. It is recommended to use the new conditions compatible with I18N instead.

User task

The definition of a user task is described in a dedicated [chapter](#).

Script task

Library script tasks

Furthermore, a label and a description can be specified according to languages.

EBX5 provides built-in script tasks:

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

Some script tasks are marked as "deprecated" because they are not compatible with I18N. It is recommended to use the new script tasks compatible with I18N instead.

Data context

A Data Context is linked to each workflow. This data context can be used to define variables which can be used as input and/or output variable in different steps of the workflow.

Some adjustable parameters

Information

Workflow model's information can be edited using the *Actions* button from the navigation pane. Editable information are the following:

Editable fields	Content required
Owner	Person, who as the owner of the workflow model, can edit its information and define permission rules on it.
Localized documentation	Label and description associated with the workflow model in various languages.
Activation	A workflow model must be activated to be published.

Specific settings per model

Configuration for a workflow model is accessible in the navigation pane. Main properties are the following:

Configuration options	Content required
Notification of start	Specify a list of profiles, who should received a notification, chosen from the template list, when a workflow is started.
Notification of completion	List of profiles, who should received a notification, chosen from the template list, when a workflow is completed.
Priority	<p>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 set for the repository, the repository default priority will be used for any associated workflow with no priority. See Work item priorities for more information.</p> <p>Note: Only users that are defined as workflow administrators will be able to manually modify the priority level of any associated data workflows.</p>
Permissions	To find out how to define permissions for a workflow publication, please see below.
Programmatic action permissions	Defines a component that handles permissions. If set, overrides all permissions defined above.

Permissions on a workflow publication

Permissions	Authorized Profile Definition
Workflow administration	Allows you to perform administrative tasks on the workflow based on this workflow model. Authorization for specific tasks can be defined using the <i>Show advanced configuration</i> link.
Allocation management	Allows you to allocate, deallocate or reallocate work items to users. Authorization for specific operation can be defined using the <i>Show advanced configuration</i> link.
Workflow launching	To launch a workflow.
Workflow monitoring	To view the running workflow even if it is not directly concerned by the current work items of the workflow. Such a user can also view the workflow history of completed data workflows.

Note: a user who has no specific privileges assigned can only see a work item of this workflow if it is offered or allocated to that user.

See also: [Workflow administration](#)

CHAPTER 27

User task

Label and description

The label and the description of the task can be specified in various languages, in order to inform the user about the purpose of this task.

Definition


Profiles

The profiles define to whom the user task is intended for. A work item is created and offered to each specified profile. If a profile refers to a user instead of a role, the work item is directly allocated to him.

Service to be called

EBX5 provides several built-in services:

- Access to a content.
- Create a new record.
- Validate a data space, a snapshot or a data set.
- Merge a data space.
- Compare contents.
- Export data from a table in XML files.
- Export data from a table in CSV Files.
- Import data from a XML file into a table .
- Import data from a CSV file into a table.

Each service requires specific parameters, a wizard assistant is offered to select data space, snapshot or value from the data context: .

See also: [EBX5 built-in services](#)

Termination

Task termination criteria

A single user task may be assigned to multiple *participants* and generate multiple work items during workflow execution. When defining a user task in the workflow model, you may select one of the predefined methods for determining when 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 can designate three 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.'

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'

Customized labels

At the user tasks execution, the user can accept or reject his work item by clicking the matching button. In workflow modeling, it is possible for some user task to define a customized label and confirmation message for these buttons. This feature is useful to add a specific meaning to the accept or reject a work item action.

Enable confirmation request

By default, when the user saves his decision by clicking on accept or reject button after the task execution, a confirmation request is displayed.

It is possible to disable this confirmation request for the decision by setting this field to false.

Notification

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

Furthermore, it is possible to define a profile to whom every email sent will also be sent.

See also: [Message templates](#)

Reminder

Reminder emails for outstanding offered or allocated work items can be sent to the concerned users periodically. The recipients are the users to which the work item is offered or allocated, as well as any recipient to 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

A user task can have a deadline. When this date is passed and if the associated works items are not completed, a specific email is sent to the concerned users. This email will then be re-sent once a day until task completion.

There are two deadline types:

- *Absolute deadline*: a calendar date.
- *Relative deadline*: duration (in hours, days or months). The duration is evaluated from the reference date: beginning of the user task or beginning of the workflow.

CHAPTER 28

Workflow model publication

A workflow model becomes executable only after publication.

It is done by creating a snapshot, where the workflow publication will be stored. This mechanism insures the stability of a workflow publication during the execution of the associated workflow.

Workflow model publication can be accessed using the *Publish* button from the navigation pane.

Publication step

Workflow model selection

It is possible to publish several workflow models at once. When the publication service is called, the existing workflow models are displayed. The user must select the workflow models he wishes to publish.

Note

A workflow model must be activated in order to be published.

Snapshot and workflow publication information

A label and a description can be specified for the snapshot to be created. Default label is the date and time of publication and default description indicates who published the workflow model.

For each selected workflow model, the publication name must be filled and unique. When the workflow model has already been published, it is possible to update it. The names of available workflow publication associated with the same workflow model are displayed. It is possible to select one of them in order to reuse an existing workflow publication. In this case, the old workflow publication is made unavailable.

Data workflows

CHAPTER 29

Introduction to data workflows

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](#)
- [Launching and monitoring data workflows](#)
- [Administration of data workflows](#)
- [Permissions on a workflow publication](#)

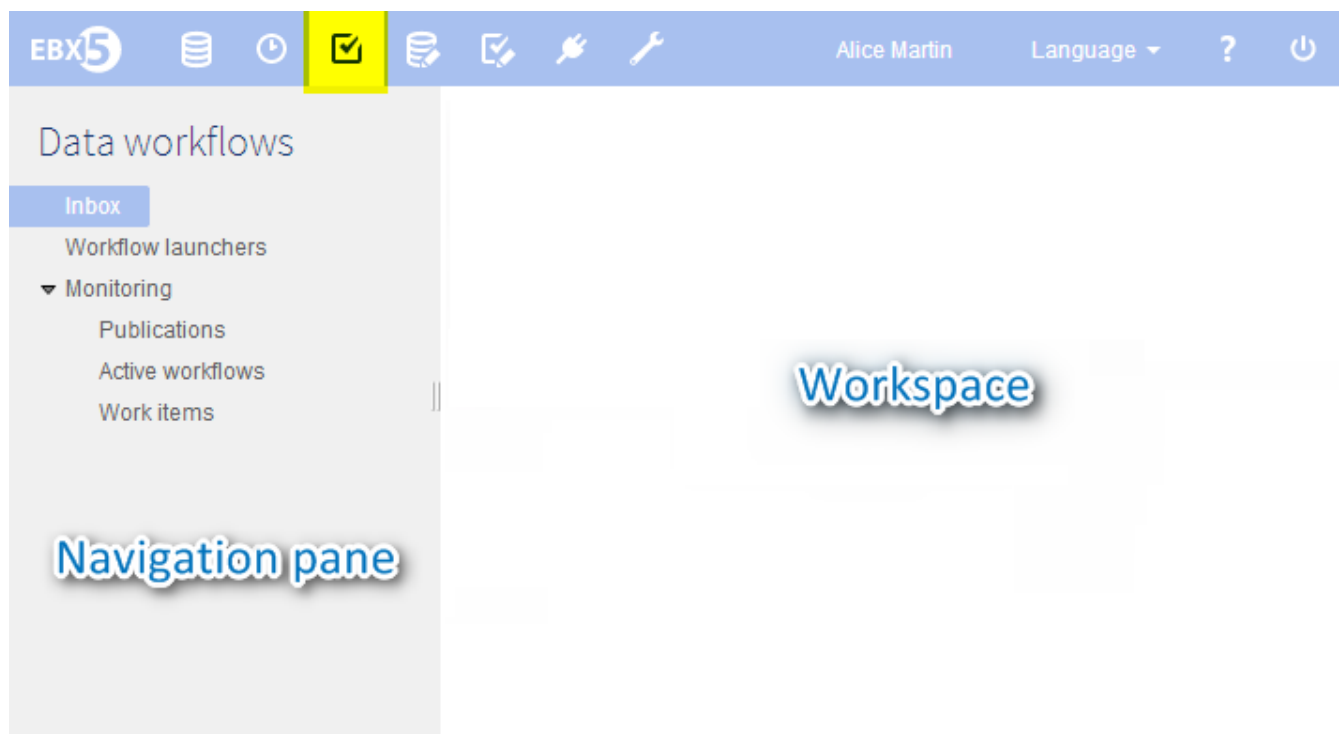
Related concepts: [Workflow models](#)

CHAPTER 30

Using the Data Workflows area user interface

Navigating within the interface

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

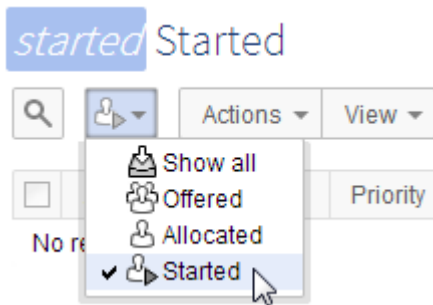


The navigation pane is organized into the following sections:


Work item 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.

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.



Graphical workflow view

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

CHAPTER 31

Work items

About work items

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

Work item states

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

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

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

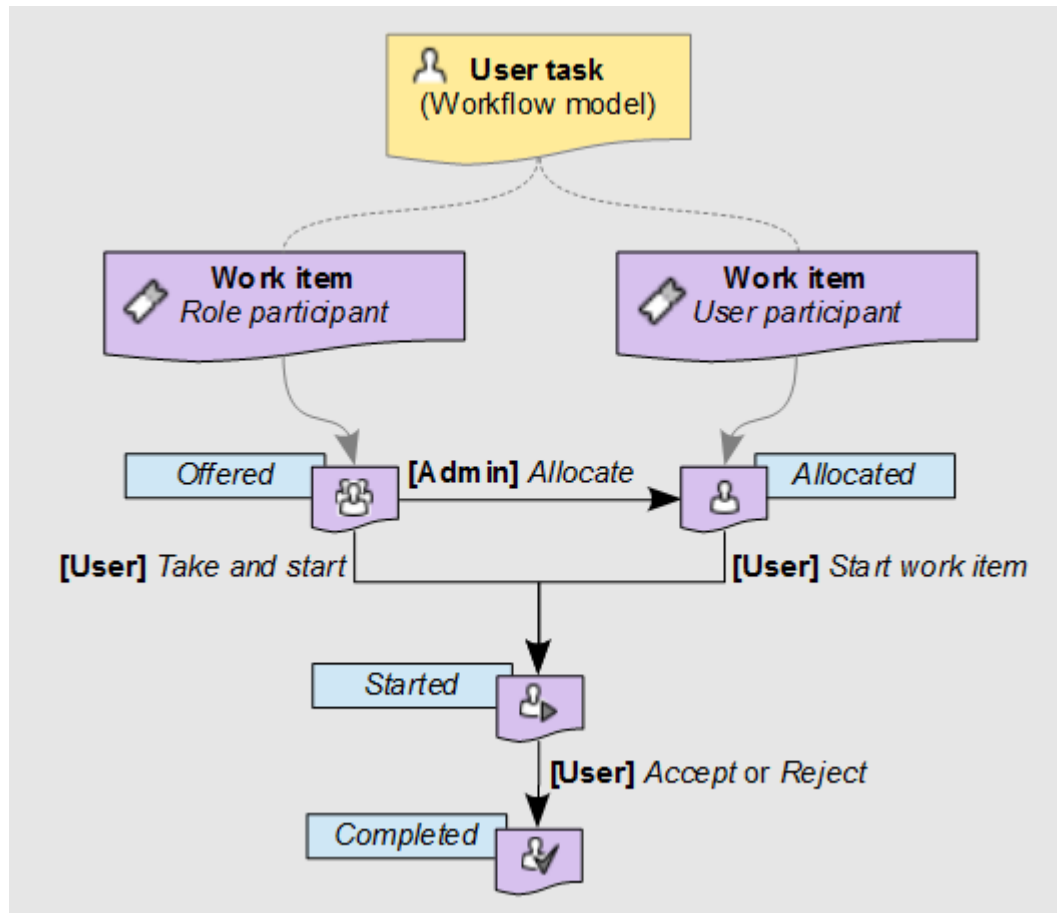
Before a user has claimed the offered work item, a manager of the data workflow can intervene to manually assign the work item to a single user, thus moving the work item to the *allocated* state. Then, when that user begins work on the work item by performing the action 'Start work item', the work item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

Note

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

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


Diagram of work item states



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.

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 \(glossary\)](#)

Related concepts: [User task](#)

CHAPTER 32

Launching and monitoring data workflows

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.

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.

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.

Note

These actions are not available for work items that do not define any role participants (only specific users).

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 <i>offered</i> state.
Deallocate	Reset a work item in the <i>allocated</i> state to the <i>offered</i> state.
Reallocate	Modify the user to whom a work item is allocated. This action is available for work items in the <i>allocated</i> state.

See also:

- [Work items](#)
- [Permissions on a workflow publication](#)

Related concepts: [Workflow models](#)

CHAPTER 33

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.

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.

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.
- **Finished:** The token has reached the end of the data workflow.
- **Error:** An error has occurred.

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](#).
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

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 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 an active data workflow

In order to stop 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.

Cleaning a completed data workflow

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

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 a workflow publication](#)

Data services

CHAPTER 34

Introduction to data services

Data services offers the possibility to access and interact with EBX5, in order to:

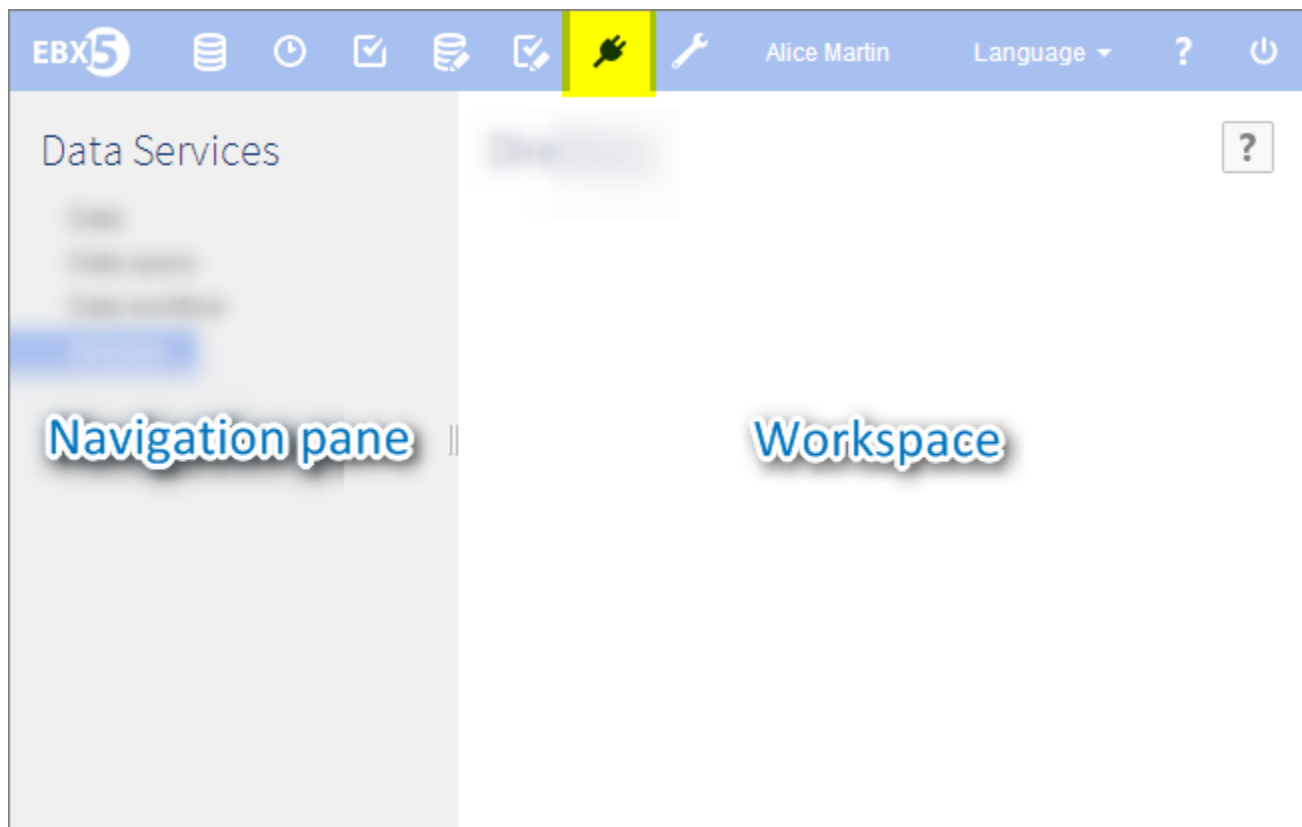
- access data stored in tables (see [data](#)),
- interact with data space (see [data space](#)),
- control workflows (see [workflow](#)),
- define lineage with 3rd-party software (see [lineage](#)).

A data service is a standard web service which exposes all features and data (example: creating a data space, updating records in a data set, etc.).

If you still wonder what those technical words refer to, do not hesitate to go to the [glossary](#).

Data services overview

Access within the interface



Associated notions and tools

Standard data services	Data services allowing access to a given data set/data space/workflow.
Lineage	Data services that take into consideration user permissions and avoids this way possible errors such as an access request to data hidden from the actual user logged in.

To know more about the WSDL (Web Services Description Language), please refer to the [World Wide Web Consortium \(W3C\)](#).

See also:

- [Data model](#)
- [Data space](#)
- [Data set](#)
- [Workflow model](#)

- [*Data workflows*](#)

CHAPTER 35

Main actions

Generate WSDL to access data

WSDL generation for data access can be done by selecting *Data* in the *Data Services* section.

Steps for generating a WSDL are the following:

1. Select a data space.
2. Input a data set identifier (or unique name) and click on next.
3. Select authorized operations for each tables.
4. Download the generated WSDL.

Available operations on a selected data set table

- *Select record(s).*
- *Insert record(s).*
- *Update record(s).*
- *Delete record(s).*
- *Count record(s).*
- *Get changes between data space or snapshot:* gives you a summary of the differences between this table and its equivalent, if it exists, in either a snapshot or another data space.
- *Run multiple operations across tables in the data set.*

Generate WSDL to access a data space

WSDL generation for data space manipulation is accessible by selecting *Data space* from the *Data Services* section. The generated WSDL is not specific to a data space and no information is required. It can be downloaded using the *Download WSDL* button located in the workspace area.

Available Operations

- *Create a data space.*
- *Close a data space.*
- *Create a snapshot.*
- *Close a snapshot.*
- *Merge a data space.*
- *Validate a data space or a snapshot.*
- *Validate a data set.*

Generate WSDL to control workflow

WSDL generation for workflow control is accessible by selecting *Data workflow* from the *Data Services* section. The generated WSDL is not specific to any workflow publication and no information is required. It can be downloaded using the *Download WSDL* button located in the workspace area.

Available Operations

- *Start a data workflow.*
- *End a data workflow.*

Generate WSDL for lineage

WSDL generation for lineage is accessible by selecting *Lineage* from the *Data Services* section, if some security profiles have been granted by an administrator profile in the *Lineage* administration.

WSDL generated to access tables and available operations are the same as for [WSDL generation for data access](#).

Steps for generating a WSDL are the following:

1. Select a role or user whose permission will be applied. Please note that a role or user must be authorized to be used for lineage by the administrator.
2. Select a data space.
3. Input a data set identifier (or unique name) and click on next.
4. Select tables and authorized operations.
5. Download the generated WSDL.

Reference Manual

Integration

CHAPTER 36

Using EBX5 as a web component

Overview

EBX5 can be used as a user interface web component, called through the HTTP protocol. An EBX5 web component can be integrated into any application that is accessible through a supported web browser. This method of access offers the major benefits of EBX5, such as user authentication, data validation, and automatic user interface generation, while additionally providing the ability to focus user navigation on specific elements of the repository.

Typical uses of EBX5 web components include integrating them into the intranet frameworks of organizations or into applications that manage the assignment of specific tasks to users.

Repository element and scope selection

When an EBX5 web component is called, the user must first be authenticated in the newly instantiated HTTP session. The web component then selects a repository element and displays it according to the *scope* layout parameter defined in the request.

The repository elements that can be selected are as follows:

- Data space or snapshot
- Data set
- Node
- Table or a published custom view
- Table record

The scope determines how much of the user interface is displayed to the user, thus defining where the user is able to navigate in the session. The default scope that the web component uses depends on the entity or service being selected or invoked by the request.

Request specifications

Base URL

In a default deployment, the base URL must be of the following form:

```
http://<host>[:<port>]/ebx/
```

User authentication and session information parameters

Parameter	Description	Required
login and password, or a <i>user directory-specific token</i>	Specifies user authentication properties. If neither a login and password pair nor a user directory-specific token is provided, user will be required to authenticate using through the repository login page.	No
trackingInfo	Specifies the tracking information of the new session. Tracking information is logged in history tables. Additionally, it can be used to programmatically restrict access permissions.	No
redirect	The URL to which the user will be redirected at the end of the component session, when they click on the button 'Close'. The close button is always displayed for record selections, but whether or not it is displayed for all other cases must be specified using the parameter <code>closeButton</code> .	No
locale	Specifies the locale to use. Value is either <code>en-US</code> or <code>fr-FR</code> .	No, default is the locale registered for the user.

Entity and service selection parameters

Parameter	Description	Required
branch	Selects the specified data space.	No
version	Selects the specified snapshot.	No
instance	Selects the specified data set. The value must be the reference of a data set that exists in the selected data space or snapshot.	Only if <code>xpath</code> or <code>viewPublication</code> is specified.
viewPublication	<p>Specifies the publication name of the tabular view to apply to the selected content.</p> <p>This publication name is the one declared during the publication of the view. It can be found in the 'Administration' area under Views configuration > User views.</p> <p>All settings of the view, that is, its filters, sort order, and displayed columns, are applied to the result. A data space and a data set must be selected in order for this view to be applied. The target table selection is not necessary, as it can be automatically determined based on the definition of the view. This parameter can be combined with the predicate specified in the <code>xpath</code> parameter as a logical 'AND' operation.</p>	No
xpath	<p>Specifies a node selection in the data set. Value must be a valid absolute path located in the selected data set. The notation must conform to a simplified XPath, with abbreviated syntax.</p> <p>If the parameter <code>viewPublication</code> is also specified, the table path is not necessary. The parameter can directly contain the predicate, surrounded by "[" and "]".</p> <p>For XPath syntax, see XPath supported syntax</p>	No
service	<p>Specifies the service to access.</p> <p>For more information on built-in UI services, see Built-in services.</p>	No

Example calls to an EBX5 web component

Minimal URI:

```
http://localhost:8080/ebx/
```

Logs in as the user 'admin' and selects the 'Reference' data space:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=Reference
```

Selects the 'Reference' data space and accesses the built-in validation service:

```
http://localhost:8080/ebx/?
login=admin&password=admin&branch=Reference&service=@validation
```

Selects the roles table in the default directory:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-
directory&instance=ebx-directory&xpath=/directory/roles
```

Selects the record 'admin' in the default directory:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-  
directory&instance=ebx-directory&xpath=/directory/user[./login="admin"]
```

Accesses the interface for creating a new user in the default directory:

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-  
directory&instance=ebx-directory&xpath=/directory/user&service=@creation
```

Compares the record 'admin' in the default directory with the record 'jSmith':

Compares the record 'R1' in the data set 'instanceld' in the data space 'Reference' with the record 'R0':

```
http://localhost:8080/ebx/?login=admin&password=admin&branch=ebx-  
directory&instance=ebx-directory&xpath=/directory/user[./  
login="admin"]&service=@compare&compare.branch=ebx-directory&compare.instance=ebx-  
directory&compare.xpath=/directory/user[./login="jSmith"]
```

CHAPTER 37

Built-in UI services

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

- when defining workflow model tasks

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

Access a data space

The data space selection service is automatically considered to be complete.

Service name parameter: `service=@selectDataSpace`

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - A data space is required for this service.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
trackingInfo	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
version	Snapshot	The identifier of the specified snapshot - A data space is required for this service.

Access data (default service)

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

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

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - A data space is required for this service.
disableAutoComplete	Disable Accept at start	By default, the interaction associated with this service is directly considered as complete. Therefore, the Accept button is automatically displayed at the opening of the work item. This parameter is useful to disable this behavior. If the value is 'true', the developer will be in charge of completing the interaction by using SessionInteraction in a UI service or a trigger, for example. The default value is 'false'.
instance	Data set	The value must be the reference of a data set that exists in the selected data space.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
trackingInfo	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
version	Snapshot	The identifier of the specified snapshot - A data space is required for this service.
viewPublication	View	The publication name of the view to display. The view must be configured for the selected table.
xpath	Data set node (XPath)	The value must be a valid absolute location path in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax.

Compare contents

The compare service is automatically considered complete.

Service name parameter: `service=@compare`

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - A data space or snapshot and a data space or snapshot to compare to are required for this service.
compare.branch	Data space to compare	The identifier of the data space to compare - A data space or snapshot and a data space or snapshot to compare to are required for this service.
compare.instance	Data set to compare	The value must be the reference of a data set that exists in the selected data space to compare.
compare.version	Snapshot to compare	The identifier of the snapshot to compare - A data space or snapshot and a data space or snapshot to compare to are required for this service.
compare.xpath	Table or record to compare (XPath)	The value must be a valid absolute location path of a table or a record in the selected data set to compare. The notation must conform to a simplified XPath, in its abbreviated syntax.
instance	Data set	The value must be the reference of a data set that exists in the selected data space.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
trackingInfo	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
version	Snapshot	The identifier of the specified snapshot - A data space or snapshot and a data space or snapshot to compare to are required for this service.
xpath	Table or record (XPath)	The value must be a valid absolute location path of a table or a record in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax.

Create a new record

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

Service name parameter: `service=@creation`

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - This field is required for this service.
instance	Data set	The value must be the reference of a data set that exists in the selected data space - This field is required for this service.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
trackingInfo	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
xpath	Data set table (XPath)	The value must be a valid absolute location path of a table in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Output parameters

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

Duplicate a record

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

Service name parameter: `service=@duplicate`

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - This field is required for this service.
instance	Data set	The value must be the reference of a data set that exists in the selected data space - This field is required for this service.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
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.

Export data from a table in CSV files

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

Service name parameter: `service=@exportToCSV`

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - A data space is required for this service.
instance	Data set	The value must be the reference of a data set that exists in the selected data space - This field is required for this service.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
trackingInfo	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
version	Snapshot	The identifier of the specified snapshot - A data space is required for this service.
xpath	Data set table to export (XPath)	The value must be a valid absolute location path of a table in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Export data from a table in XML files

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

Service name parameter: `service=@exportToXML`

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - A data space is required for this service.
instance	Data set	The value must be the reference of a data set that exists in the selected data space - This field is required for this service.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
trackingInfo	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
version	Snapshot	The identifier of the specified snapshot - A data space is required for this service.
xpath	Data set table to export (XPath)	The value must be a valid absolute location path of a table in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Import data into a table from an CSV file

The importFromCSV service is considered complete when import is performed.

Service name parameter: `service=@importFromCSV`

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - A data space is required for this service.
instance	Data set	The value must be the reference of a data set that exists in the selected data space - This field is required for this service.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
trackingInfo	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
xpath	Data set table to import (XPath)	The value must be a valid absolute location path of a table in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Import data into a table from an XML file

The importFromXML service is considered complete when import is performed.

Service name parameter: `service=@importFromXML`

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - A data space is required for this service.
instance	Data set	The value must be the reference of a data set that exists in the selected data space - This field is required for this service.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
trackingInfo	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
xpath	Data set table to import (XPath)	The value must be a valid absolute location path of a table in the selected data set. The notation must conform to a simplified XPath, in its abbreviated syntax - This field is required for this service.

Merge a data space

The merge service is considered complete when merger is performed and data space is closed.

Service name parameter: `service=@merge`

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - A data space is required for this service.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
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.

Validate a data space, a snapshot or a data set

The validation service is automatically considered complete.

Service name parameter: `service=@validation`

Input parameters

Parameter	Label	Description
branch	Data space	The identifier of the specified data space - A data space or snapshot is required for this service.
instance	Data set	The value must be the reference of a data set that exists in the selected data space.
scope	Scope	Defines the scope of the user navigation for this service, namely, the entities that the user is able to select during their session.
trackingInfo	Tracking information	Tracking information is logged into 'history' logs. It may also be used for any other purpose like access control or additional export information.
version	Snapshot	The identifier of the specified snapshot - A data space or snapshot is required for this service.

Output parameters

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

CHAPTER 38

Data services

Overview of data services

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

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

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

Other generic operations for:

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

SOAP interactions

Input and output message encoding

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

Tracking information

Depending on the data services operation being called, it may be possible to specify session tracking information in an optional SOAP header. For example:

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

Exceptions handling

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

```
<soapenv:Fault>
  <faultcode>soapenv:java.lang.IllegalArgumentException</faultcode>
  <faultstring />
  <faultactor>admin</faultactor>
  <detail>
    <m:StandardException xmlns:m="urn:ebx-schemas:dataservices_1.0">
      <code>java.lang.IllegalArgumentException</code>
      <label/>
      <description>java.lang.IllegalArgumentException:
        Parent home not found at
        com.orchestranetworks.XXX.YYY.ZZZ.AAA.BBB(AAAA.java:44) at
        com.orchestranetworks.XXX.YYY.ZZZ.CCC.DDD(CCC.java:40) ... </description>
    </m:StandardException>
  </detail>
</soapenv:Fault>
```

Data services security

Authentication

Authentication is mandatory. By default, several methods of authentication exist. They are described below, in the look-up order attempted by the Web Service connector.

Note

When using JMS, only the standard HTTP 'Base64 Basic Authentication' is attempted.

- Standard HTTP 'Base64 Basic Authentication' encoded using HTTP-Header Authorization, as described in [RFC 2324](#).

If the user agent wishes to send the userid "Aladdin" and password "open sesame", it will use the following header field: Authorization: Basic QWxhZGRpbjpvYVUyIHNlc2FtZQ==

- A simple authentication based on the specification [Web Services Security UsernameToken Profile 1.0](#).

Only the mode PasswordText is supported. This is done with the following SOAP header defined in the WSDL:

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

Overriding the default security header

It is possible to override the 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>

```

Lookup mechanism

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

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

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

Data service operations generated from a data model

These operations are generated using a given data model. For example, for a table located at the path `/root/XX/exampleTable`, the generated requests would follow the structure of its underlying data model and include the name of the table `<m:{operation}_exampleTable xmlns:m="urn:ebx-schemas:dataservices_1.0">`.

Common data model operation request parameters

Several parameters are common to several operations and are detailed below.

Element	Description	Required
branch	The identifier of the data space to which the data set belongs.	One of either this parameter or the 'version' parameter must be defined
version	The identifier of the snapshot to which the data set belongs.	One of either this parameter or the 'branch' parameter must be defined
instance	The unique name of the data set which contains the table to query.	Yes
predicate	XPath predicate 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 (used by the insert and update operations)	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 .	Yes
viewPublication	This parameter can be combined with the predicate parameter as a logical AND operation. The behavior of this parameter is described in the section EBX5 as a web component . It cannot be used if the 'viewId' parameter is used, and cannot be used on hierarchical views.	No
viewId	<i>Deprecated since version 5.2.3.</i> 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

Select operation

Select request

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

with:

Element	Description	Required
branch	See the description under Common parameters .	
version	See the description under Common parameters .	
instance	See the description under Common parameters .	
predicate	See the description under Common parameters . This parameter can be combined with the viewPublication parameter as a logical AND operation.	
viewPublication	See the description under Common parameters .	
includesTechnicalData	The response will contain technical data if 'true'. See also the optimistic locking section. Each returned record will contain additional attributes for this technical information, for instance: <pre>...<table ebxd:lastTime="2010-06-28T10:10:31.046" ebxd:lastUser="Uadmin" ebxd:uuid="9E7D0530-828C-11DF- B733-0012D01B6E76">... </pre>	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 be fetched, this value is provided by the previous response, as the element <code>lastRecordPredicate</code> . If the passed record is not found, the first page will be returned.	No

Select response

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

```

    <a>W</a>
    <b>W</b>
    <c>W</c>
    <d>W</d>
  </TableName>
</XX>
</credentials>
<lastRecordPredicate>./a='key1'</lastRecordPredicate>
</ns1:select_{TableName}Response>

```

See also the [optimistic locking](#) section.

Delete operation

Delete request

```

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

```

with:

Element	Description	Required
branch	See the description under Common parameters .	
version	See the description under Common parameters .	
instance	See the description under Common parameters .	
predicate	See the description under Common parameters .	
occultIfInherit	Occults the record if it is in inherit mode. Default value is 'false'.	No
checkNotChangedSinceLastTime	Timestamp used to ensure that the record has not been modified since the last read. Also see the optimistic locking section.	No

Delete response

```

<ns1:delete_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>00</status>
</ns1:delete_{TableName}Response>

```

with:

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

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 .
version	See the description under Common parameters .
instance	See the description under Common parameters .
predicate	See the description under Common parameters .

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>true</updateOrInsert>
  <byDelta>true</byDelta>
  <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 .	
version	See the description under Common parameters .	
instance	See the description under Common parameters .	
updateOrInsert	If 'true' and the record does not currently exist, the operation creates the record.	No
byDelta	If 'true' and an element does not currently exist in the incoming message, the target value is not changed. The complete behavior is described in the sections Insert and update operations .	No
data	See the description under Common parameters .	

Update response

```

<ns1:update_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>00</status>
</ns1:update_{TableName}Response>

```

with:

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

Insert operation

Insert request

```

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

```



```

    ...
    </TableName>
  </XX>
</data>
</m:insert_{TableName}>

```

with:

Element	Description	Required
branch	See the description under Common parameters .	
version	See the description under Common parameters .	
instance	See the description under Common parameters .	
data	See the description under Common parameters .	
byDelta	<p>If 'true' and an element does not currently exist in the incoming message, the target value is not changed.</p> <p>The complete behavior is described in the sections Insert and update operations.</p>	No

Insert response

```

<ns1:insert_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>00</status>
  <inserted>
    <predicate>./a='String'</predicate>
  </inserted>
</ns1:insert_{TableName}Response>

```

with:

Element	Description
status	'00' indicates that the operation has been executed successfully.
predicate	A predicate matching the primary key of the inserted record. When several records are inserted, the predicates follow the declaration order of the records in the input message.

Get changes operations

Get changes requests

Changes between two data sets:

```

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

```

```
<compareWithInstance>String</compareWithInstance>
<resolvedMode>boolean</resolvedMode>
</m:getChangesOnDataSet_{schemaName}>
```

Changes between two tables:

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

with:

Element	Description	Required
branch	See the description under Common parameters .	
version	See the description under Common parameters .	
instance	See the description under Common parameters .	
compareWithBranch	The identifier of the data space with which to compare. You must use only one parameter between this one and compareWithVersion .	No
compareWithVersion	The identifier of the snapshot with which to compare. You must use only one parameter between this one and compareWithBranch .	No
resolvedMode	Defines whether or not the difference is calculated in resolved mode. Default is 'true'.	No

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

- if the current data space or snapshot is a data space, the comparison is made with its initial snapshot (includes all changes made in the data space);
- if the current data space or snapshot is a snapshot, the comparison is made with its parent data space (includes all changes made in the parent data space since the current snapshot was created);
- returns an exception if the current data space is the 'Reference' data space.

Get changes responses

Changes between two data sets:

```
<ns1:getChangesOnDataSet_{schemaName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <getChanges_{TableName1}>
    ... see the getChanges between tables response example ...
  </getChanges_{TableName1}>
  <getChanges_{TableName2}>
    ... see the getChanges between tables response example ...
  </getChanges_{TableName2}>
</ns1:getChangesOnDataSet_{schemaName}Response>
```

```

</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>

```

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 .	
version	See the description under Common parameters .	
instance	See the description under Common parameters .	
viewPublication	See the description under Common parameters .	

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. 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>
  <request id="id1">
    <m:{operation}_{TableName}>
      ...
    </m:{operation}_{TableName}>
  </request>
  <request id="id2">
```

```

<m:{operation}_{TableName}>
...
</m:{operation}_{TableName}>
</request>
</m:multi>

```

with:

Element	Description	Required
branch	See the description under Common parameters .	
version	See the description under Common parameters .	
instance	See the description under Common parameters .	
request	<p>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.</p> <p>Operations such as <code>count</code>, <code>select</code>, <code>getChanges</code>, <code>getCredentials</code>, <code>insert</code>, <code>delete</code> or <code>update</code>.</p>	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:multiResponse xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <response id="id1">
    <ns1:{operation}_{TableName}Response>
      ...
    </ns1:{operation}_{TableName}Response>
  </response>
  <response id="id2">
    <ns1:{operation}_{TableName}Response>
      ...
    </ns1:{operation}_{TableName}Response>
  </response>
</ns1:multiResponse>

```

with:

Element	Description
response	<p>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.</p> <p>The content of the element corresponds to the response of a single operation, such as count, select, getChanges, getCredentials, insert, delete or update.</p>

Optimistic locking

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

In the select request, it is possible to ask for technical information by adding the element `includesTechnicalData`:

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

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

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

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

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

```
<instance>String</instance>
<predicate>String</predicate>
<checkNotChangedSinceLastTime>2010-06-28T10:10:31.046</checkNotChangedSinceLastTime>
</m:delete_{TableName}>
```

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

Data service operations on data sets and data spaces

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

<i>Element</i>	<i>Description</i>	<i>Required</i>
branch	Identifier of the target data space on which the operation is applied. When not specified, the 'Reference' data space is used except for the merge data space operation where it is required.	One of either this parameter or the 'version' parameter must be defined. Required for the data space merge and replication refresh operations.
version	Identifier of the target snapshot on which the operation is applied.	One of either this parameter or the 'branch' parameter must be defined
versionName	Identifier of the snapshot to create. If empty, it will be defined on the server side.	No
childBranchName	Identifier of the data space child to create. If empty, it will be defined on the server side.	No
instance	The unique name of the data set on which the operation is applied.	Required for the replication refresh operation.
ensureActivation	Defines if validation must also check whether this instance is activated.	Yes
details	<p>Defines if validation returns details.</p> <p>The optional attribute <code>severityThreshold</code> 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.</p> <p>The optional attribute <code>locale</code> (default 'en-US') defines the language in which the validation messages are to be returned.</p>	No. If not specified, no details are returned.
owner	Defines the owner.	No
branchToCopyPermissionFrom	Defines the identifier of the data space from which to copy the permissions.	No
documentation	Documentation for the data space or the snapshot to create. Multiple documentation elements may be used.	No
locale (nested under the documentation element)	Locale of the data space or snapshot documentation.	Only required when the <code>documentation</code> element is used
label (nested under the documentation element)	Label of the data space or the snapshot to create.	No

<i>Element</i>	<i>Description</i>	<i>Required</i>
description (nested under the documentation element)	Description of the data space or the snapshot to create.	No

Validate a data space

Validate data space request

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

Validate data space response

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

Validate a data set

Validate data set request

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

Validate data set response

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

```
</message>
<subject>
  <table>Path</table>
  <predicate>String</predicate>
  <path>Path</path>
</subject>
</reportItem>
</details>
</validationReport>
</ns1:validateInstance_Response>
```

Create a data space

Create data space request

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

Create data space response

```
<ns1:createBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>00</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>00</status>
  <versionName>String</versionName>
</ns1:createVersion_Response>
```

with:

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

Merge a data space

Merge data space request

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

with:

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

Note

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

Merge data space response

```
<ns1:mergeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>00</status>
</ns1:mergeBranch_Response>
```

with:

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

Close a data space or snapshot

Close data space or snapshot request

Close data space request:

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

Close snapshot request:

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

with:

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

Close data space or snapshot response

Close data space response:

```
<ns1:closeBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>00</status>
</ns1:closeBranch_Response>
```

Close snapshot request:

```
<ns1:closeVersion_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>00</status>
</ns1:closeVersion_Response>
```

Data service operations on data workflows

Parameters for operations on data workflows are as follows:

Element	Description	Required
publishedProcessKey	Identifier of the workflow to start.	Yes
documentation	Documentation for the process instance to be created.	No
parameters	Input parameters for the process instance to be created.	No
processInstanceId	Identifier of the process instance, as returned by the <code>workflowProcessInstanceStart</code> operation.	Yes

Start a workflow

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

End a workflow

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

Known limitations

Naming convention

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

Date, time & dateTime format

Data services only support the following date and time formats:

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

CHAPTER 39

XML import and export

XML imports and exports can be performed on tables through the user interface using the **Actions** menu in the workspace.

Both imports and exports are performed in the context of a data set.

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	<p>If 'by delta' mode is disabled (default):</p> <p>Target field value is set to one of the following:</p> <ul style="list-style-type: none"> • 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 <code>null</code>. • 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 <code>null</code> from an empty string, the target field value is set to <code>null</code>. If it is a string that does not distinguish between the two, an empty string. <p>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.</p> <p>If 'by delta' mode has been enabled through data services or the Java API:</p> <ul style="list-style-type: none"> • For the <code>update</code> operation, the field value remains unchanged. • For the <code>insert</code> operation, the behavior is the same as when <code>byDelta</code> mode is disabled.
Element exists but is empty (for example, <code><fieldA/></code>)	<ul style="list-style-type: none"> • For nodes of type <code>xs:string</code> (or one of its sub-types), the target field's value is set to <code>null</code> if it distinguishes <code>null</code> from an empty string. Otherwise, the value is set to empty string. • For non-<code>xs:string</code> type nodes, an exception is thrown in conformance with XML Schema.
Element is present and <code>null</code> (for example, <code><fieldA xsi:nil="true"/></code>)	<p>The target field is always set to <code>null</code> except for lists, for which it is not supported.</p> <p>In order to use the <code>xsi:nil="true"</code> attribute, you must import the namespace <code>xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code>.</p>

Optimistic locking

If the technical attribute `x:lastTime` exists in the source XML file, the import mechanism performs a verification to prevent an update operation on a record that may have changed since the last read. The timestamp associated with the current record will be compared to this timestamp. If they are different, the update is rejected.

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	<p>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.</p> <p>Note: If this option is selected, the exported file will not be able to be re-imported.</p>
Include technical data	<p>Specifies whether internal technical data will be included in the export.</p> <p>Note: If this option is selected, the exported file will not be able to be re-imported.</p>
Include computed values	Specifies whether computed values will be exported.
Is indented	Specifies whether the file should be indented to improve readability.

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

CHAPTER 40

CSV import and export

CSV imports and exports can be performed on tables through the user interface using the **Actions** menu in the workspace.

Both imports and exports are performed in the context of a data set.

Imports

When importing a CSV 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.

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.
User-friendly mode	<p>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.</p> <p>Note: If this option is selected, the exported file will not be able to be re-imported.</p>
Include technical data	<p>Specifies whether internal technical data will be included in the export.</p> <p>Note: If this option is selected, the exported file will not be able to be re-imported.</p>
Include computed values	Specifies whether computed values will be exported.
Column header	<p>Specifies the whether or not to include column headers in the CSV file.</p> <ul style="list-style-type: none"> • 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.
Separator	Specifies the field separator to use in the exported file. The default is to use commas.

Handling of field values

Aggregated lists

The CSV import and export services support multi-valued fields, namely aggregated lists. This is only supported for simple typed lists, such as lists of `string`, `date`, or `int`, and for table references. If a table

reference is linked to a composite primary key, each item in the list is a formatted string, for example, "true|99". Aggregated lists of groups are not exported.

At export, the items in the list are separated using line separators. In cases where the exported field already contains a line separator, for example in an `osd:html` or an `osd:text`, the code `_crnl_` is inserted in place of the field value's line separators. The same formatting is expected at import, with the whole field value surrounded by quotes.

Hidden fields

Hidden fields are be exported as `ebx-csv:hidden` strings. An imported hidden string will not modify a node's content.

'Null' value for strings

Using CSV import and export services, a string with a value set to `null` is exported as an empty string. Therefore, a round trip export-import procedure will end up replacing `null` string values with empty strings.

Date, time & dateTime format

The following date and time formats are supported:

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

Known limitations

Aggregated lists of groups

The CSV import and export services do not support importing multi-valued groups, that is, aggregated lists of complex type elements. Exporting such nodes will not cause any errors, however, no such values are 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](#).

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.

CHAPTER 41

Supported XPath syntax

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.

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,'yy') and osd:is-not-null(./col3))
```

```
contains(col3 , 'xxx') and ( not(coll=100) and date-greater-than(col2,'2007-12-30') )
```

Syntax specifications for XPath expressions

Overview

Expression	Format	Example
XPath expression	<i><container path></i> [<i>predicate</i>]	/books[title='xxx']
<i><container path></i>	<i><absolute path></i> or <i><relative path></i>	
<i><absolute path></i>	/a/b or //b	//books
<i><relative path></i>	../b, ./b or b	../books

Predicate specification

Expression	Format	Notes/Example
<predicate>	Example: A and (B or not(C)) A,B,C: <atomic expression>	Composition of: logical operators braces, not() and atomic expressions.
<atomic expression>	<path><comparator><criterion> or method(<path>,<criterion>)	royalty = 24.5 starts-with(title, 'Johnat')booleanValue = true
<path>	<relative path>	Relative to the table that contains it: ../authorstitle
<comparator>	<boolean comparator>, <numeric comparator> or <string comparator>	
<boolean comparator>	= or !=	
<numeric comparator>	= , != , < , > , <= , or >=	
<string comparator>	=	
<method>	<date method>, <string method>, osd:is-null method or osd:is-not-null method	
<date, time & dateTime method>	date-less-than, date-equal or date-greater-than	
<string method>	matches, starts-with, ends-with, contains, osd:is-empty, osd:is-not-empty, osd:is-equal-case-insensitive, osd:starts-with-case-insensitive, osd:ends-with-case-insensitive, or osd:contains-case-insensitive	
<criterion>	<boolean criterion>, <numeric criterion>, <string criterion>, <date criterion>, <time criterion>, or <dateTime criterion>	
<boolean criterion>	true , false	
<numeric criterion>	An integer or a decimal	-4.6
<string criterion>	Quoted character string	'azerty'
<date criterion>	Quoted and formatted as 'yyyy-MM-dd'	'2007-12-31'
<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)]`.

Contextual values

For predicates that are relative to a selected node, the criterion value right-hand side) 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)]`.

Extraction of foreign keys

The standard XPath syntax has been extended so as to extract the value of any targeted primary key field.

Example

If the table `/root/tableA` has an `osd:tableRef` field named 'fkB' whose target is `/root/tableB` and the primary key of `tableB` has two fields, `id` of type `xs:int` and `date` of type `xs:date`, then the following expressions would be valid:

- `/root/tableB[fkB = '123|2008-01-21']`, where the string "123|2008-01-21" is a representation of the entire primary key value.
- `/root/tableB[fkB/id = 123 and date-equal(fkB/date, '2008-01-21')]`, where this predicate is a more efficient equivalent to the one in the previous example.
- `/root/tableB[fkB/id >= 123]`, where any number operator could be used, as the targeted primary key field is of type `xs:int`.
- `/root/tableB[date-greater-than(./fkB/date, '2007-01-01')]`, where any date operator could be used, as the targeted primary key field is of type `xs:date`;
- `/root/tableB[fkB = ""]` is not valid as the targetted primary key has two columns.
- `/root/tableB[osd:is-null(fkB)]` checks if a foreign key is null (notdefined).

Miscellaneous

CHAPTER 42

Permissions

Permissions specify and regulate the access of each user to data and the actions he can execute.

Main principles

Permissions are related to actions (action authorized or not) and to access rights (hidden, read, read-write). The main entities controlled by permissions are:

- Data space
- Data set
- Table
- Group
- Field

Users, roles and profiles

The definition and the resolution of permissions make extensive use of the *profile* notion, a generic term that references either a user or a role.

Each user can participate in several roles and a role can be shared by several users.

Particular definitions:

- A user is an *administrator* when he participates in the built-in role ADMINISTRATOR.
- A user is an *owner of a data set* when he participates in the *owner* attribute specified on the root data set ("Infos" tab). In this case, the built-in role OWNER is activated when permissions are resolved in the context of the instance.
- A user is an *owner of a data space* when he participates in the *owner* attribute specified for the data space. In this case, the built-in role OWNER is activated when permissions are resolved in the context of the data space.

Permission rules

A permission rule defines the authorization granted to a profile on an entity.

User-defined permission rules are created through the use of EBX5 (see section [Definition of permissions with EBX5](#)).

Resolution of permissions

Permissions are always resolved in the context of an authenticated user session. Hence, the defined permission rules will be able to take into account the information associated with the current user, more particularly the roles in which he participates.

The resolution process is further detailed in section [Resolution of permissions](#) .

Particular privileges on data set

For a specific data set, the following tasks are accessible only if the user is an administrator or an owner of this data set:

- Manage permissions of data set.
- Change the owner of the root data set.
- Change the data set documentation (label and description).

Notes.

- An administrator or an owner of a data set can have a restricted access to a data set through permissions definitions, but in any case, he will keep the possibility to perform the tasks defined above.

Particular privileges on data space

A user is a "*super owner*" of a data space when:

- He is the owner of the data space and he is allowed to manage its permissions;
- or he is the owner of an ancestor data space and he is allowed to manage the permissions of this ancestor.

For a specific data space, the general rule is that only an administrator or a "super owner" may perform the following administration tasks:

- Manage permissions of data space.
- Change owner of data space.
- Lock and unlock data space.
- Change the data space documentation (label and description).

Notes. An administrator or an owner of a data space can have a restricted access to it through permissions definitions, but in any case, he will keep the possibility to perform administration tasks defined above. This implies that an administrator always sees all open data spaces and that any user sees all the data spaces that he owns.

Merge and permissions

When a data space is merged, the permissions of its parent data space are not impacted but the permissions of the data set of the parent data space are merged if and only if the user specifies it during the merge process. However when some elements are hidden for a profile, this profile will not be able to merge since he would not be aware of the actual impacts on data.

Definition of permissions with EBX5

As described below, each level has a similar schema that allows the definition of several permission rules.

Permissions on data space

For a given data space, several permission rules can be specified. For each defined profile, the allowed permissions are the following:

1. Data space access: defines access rights (read-write, read-only or hidden, see below for definition).
2. Restriction: indicates if "profile (role or user) - permission (right or action)" associations, for the current data space, should have priority.
3. Create a child data space: indicates if it is possible to create a child data space from the current data space.
4. Create a child snapshot: indicates if it is possible to create a snapshot of the current data space.
5. Initiate merge: indicates if a profile has the right to merge a data space with its parent data space.
6. Export archive: indicates if a profile has the right to export the current data space as an archive.
7. Import archive: indicates if a profile has the right to import an archive into the current data space.
8. Close a data space: indicates if a profile has the right to close the current data space.
9. Close a snapshot: indicates if a profile has the right to close a snapshot of the current data space.
10. Permissions of child data space when created: defined the same permissions as above but on current data space children.

Mode	Authorization
Write	<ul style="list-style-type: none"> The user can see the data space. The user has the right to access data set according to his rights on these.
Read-only	<ul style="list-style-type: none"> The user can visualize the data space and its snapshot. He can see the child data space if he has the right to do so. The user can at most visualize the contents of the data space. He cannot modify the data space contents.
Hiddens	<ul style="list-style-type: none"> The user can neither see the data space nor its snapshots. If the user has access to a child data space, then he can see the current data space but he cannot select it. The user cannot access the data space contents (data set contents). The user cannot perform any action on the data space.

Permissions on a new data space

When a user creates a child data space (if he is allowed to do so), the permissions of this new data space are automatically assigned to the profile "owner" and are determined from the *Permissions of child data space when created* section defined on the parent data space. If several permissions are defined through different roles, the [resolved permissions](#) are applied.

Note

Only the administrator and the owner of a data space can define a new owner for this data space or modify the associated permissions. They can also modify general information on the data space ("owner role" for data set) and also permissions of the different users.

Furthermore, if using the "Create a data space" or "Create a snapshot" built-in script tasks and since the current session is related to a system user, the resolved permissions is computed using the owner defined in the script task's configuration and is not based on the user associated with the session.

Permissions on data set

For a given data set, several permission rules can be specified. For each defined profile, permissions are the following:

Actions on data set

- Restricted mode: indicates if "profile (role or user) - permission (right or action)" associations, for the current data set, should have priority.
- Create a child data set: indicates if a given profile has the right to create a child data set.
- Duplicate data set: indicates if a given profile has the right to duplicate a data set.
- Change the data set parent: indicates if a given profile has the right to change the parent data set of a given child data set.

Default actions on tables

For a given table, several "profile-permissions" associations can be specified. For each defined profile, possible actions are the following:

- Create a record: indicates if a profile has the right to create records in a table.
- Modify a record: indicates if a profile has the right to modify a record (in case of inheritance of data in a data set tree).
- Hide a record: indicates if a profile has the right to hide a record in a table.
- Delete a record: indicates if a profile has the right to delete a record in a table.

Permissions on tables

The actions specified on tables modify the default actions (*cf.* above) on these tables in the data set.

Default access right on nodes values

- Read-write: nodes can be consulted and modified (modification of values).
- Read: nodes can only be consulted, not modified.
- Hidden: nodes are hidden.

Permissions on terminal nodes

Rights specified on terminal nodes modify the default rights (*cf.* above) on these terminal nodes.

Permissions on services

Permissions on data set services can also be defined. By default, all data set services are enabled. The administrator or the owner of the data set can modify these services permissions for a given profile.

Resolution of permissions

The resolution of permissions is always done in the context of a user's session and his associated roles. In general, a restrictive resolution is performed between a given level and its parent level. Hence at a given level, a user cannot have a permission higher than the one resolved at a parent level.

Restriction policy notion

Permissions defined with EBX5 can be restricted. Given a set of profiles to which a user belongs to, restricting some of them means to consider their permissions more important than those defined for non restricted profiles. Hence, generally:

- if restrictions are defined, the minimum permission over the restricted profiles is considered
- if no restriction is defined, the maximum permission over profiles is then taken

Example

Given two profiles P1 and P2, the following table lists the possibilities when resolving the access to a service.

P1 authorization	P2 authorization	Permission resolution
Allowed.	Allowed.	Allowed. Restrictions don't interfere.
Forbidden.	Forbidden.	Forbidden. Restrictions don't interfere.
Allowed.	Forbidden.	Allowed, unless P2 has a restriction.
Forbidden.	Allowed.	Allowed, unless P1 has a restriction.

Another example would be to hide a data space from all users, the administrator or the owner of this data space would define a restriction on the association between the built-in profile "Profile.EVERYONE" and the access right "hidden".

Solving access rights

Solving access rights defined with EBX5

For access rights resolution defined with EBX5, there are three levels of resolution: data space, data set and node.

If a user is associated with different access rights at a given level and if restrictions have been defined for these "user-rights" associations, then the minimum of the restricted rights is applied. If there is no restriction, then the maximum access right is applied for the given user. The following tables illustrate the resolution mechanism.

Three users exist and each one of them participates in different roles or profiles:

- User 1: user1 - role A - role B
- User 2: role A - role B - role C
- User 3: user3 - role A - role C

This table indicates the rights associated with those different profiles on a specific object:

Role/Profile	Rights	Restriction policy
user1	Hidden	Yes
user3	Read	No
Role A	Read/Write	No
Role B	Read	Yes
Role C	Hidden	No

This table lists the right which will be applied for each user after rights resolution:

User	Right applied
User 1	<i>hidden</i>
User 2	<i>read</i>
User 3	<i>read-write</i>

At a given level, the most restrictive right of this level and of the higher levels is applied. For instance, if a data set is in read-write access and the container data space allows only reading, then the user will have a read-only access right on this data set.

Note. Rights defined on a data set will be applied on its child data set. It is possible to modify these rights on the child data set. *The inheritance mechanism is therefore applied for either values or access rights.*

Data space/snapshot access right resolution

At data space level, the access right will be the following: if a user has several rights defined (one for each of his roles) and if restrictions have been defined, then the minimum of the restricted "user-rights" associations is applied. Otherwise, the maximum of the "user-rights" associations of the given user is applied.

On the other hand, if the user has no right defined, then if he is administrator or owner of the data space, he will have a read-write access to this data space, otherwise the data space will be hidden from him.

Data set access right resolution

At data set level, the same principle as the one applied to data space is used. Moreover, access right on data set is defined by the minimum between the right on the data space and the right on the data set (identified by using the "solving rights" principle similar to the one applied at the data space level). For instance, if a data space is in read-only access for the user U and a data set of the data space is in read-write access for the same user, then U will have a read-only access on the data set.

Node access right resolution

At node level, the same principle as the one applied to data space and to data set is used. Moreover, the right on the node is defined by the minimum between the right on the data set and the right on the node (identified by using the "solving rights" principle similar to the one applied at data space level and at data set level).

Note. At node level, one can define a specific access right for the target node. If no specific access right is defined, the default access right is then considered for the resolution process. However, the procedure is slightly different for table and table child node (see next section).

Specific case of table and table child node

This section describes the resolution process applied for a given table node or table record *N*.

For each user-defined permission rules that matches one of the user's profiles, the access right for *N* is either:

1. The locally defined access right for *N*;
2. Inherited from the access right defined on the table node;
3. Inherited from the default access right for instance's values.

Then, all matching user-defined permission rules will provide the resolved access right for *N*. Resolution is done according to [restriction policy](#).

The final resolved access right will be the minimum between data space, data set and the resolved access right for *N*.

Solving actions and services

Solving which actions and which services are accessible to a given user is done using the same process.

When several actions lists are defined for a given profile on a data set (respectively table), the actions list to consider is dynamically generated after an evaluation of each action type among the different lists (of actions) associated with this user. If some "user-(list of) actions" associations are restricted, then for each action type we verify (among the restricted associations) whether it is forbidden at least once to forbid it at all. If there is no restriction, then if at least one action type is authorized then the action is finally allowed (*cf.* tables below for actions on tables).

Two users exist and each one of them participates in different roles or profiles:

- User 1: user1 - role A - role B
- User 2: role C - role D

This table indicates the rights associated with those different profiles on a table:

Role/Profile	Create a record	Modify a record	Hide a record	Duplicate a record	Delete a record	Restriction policy
user1	Allowed	Forbidden	Allowed	Forbidden	Allowed	No
Role A	Allowed	Allowed	Forbidden	Allowed	Forbidden	Yes
Role B	Allowed	Forbidden	Allowed	Allowed	Forbidden	Yes
Role C	Allowed	Allowed	Forbidden	Forbidden	Forbidden	No
Role D	Allowed	Forbidden	Forbidden	Allowed	Forbidden	No

This table lists the actions that will be allowed for each user after rights resolution:

Users	Actions list applied
User 1	Create a record
	Duplicate a record
User 2	Create a record
	Modify a record
	Duplicate a record

CHAPTER 43

Inheritance and value resolution

Overview

The main idea behind inheritance is to mutualize resources that are shared by multiple contexts or entities. EBX5 offers several mechanisms for defining, factorizing and resolving data values: *data set inheritance* and *inherited fields*

Data set inheritance

Data set inheritance is particularly useful when data has to be specialized to various global enterprise contexts, like subsidiaries or business partners.

Based on a hierarchy of data sets, it allows you to factorize common data on the root data set (or on intermediate ones) and to specialize specific data to specific contexts.

Data set inheritance mechanisms are detailed in the [section below](#) .

Inherited fields

As opposed to data set inheritance (which exploits a global built-in relationship between data sets), inherited fields are able to exploit finer-grained dependencies that are specific to the data structure. It allows you to factorize and specialize 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 value from attributes defined in its associated family.

Note: It is not possible to use both attribute inheritance and data set inheritance for a same data set.

Data set inheritance

As described above, data set inheritance is based on a hierarchy of data set, the *data set tree*. This section's purpose is to precisely define data set inheritance mechanisms.

Lookup mechanism for values

Formally speaking, the data set inheritance lookup mechanism for values is as follows:

1. if the value is locally defined, returns it
2. otherwise, looks up the first locally defined value according to the built-in child-to-parent relationship of data set in the hierarchy tree
3. if no locally defined value is found, the default value is returned

`null` is returned if no default value is defined

Note: default values cannot be defined on:

- a single primary key node
- auto-incremented nodes
- nodes defining a computed value.

Lookup mechanism for records

Like values, table records can also be inherited as a whole by multiple contexts, but they can also be partially redefined (*overwritten*), be specific to a context (*root mode*) or even be occulted. More formally speaking, a table record has one of four distinct definition modes:

- A *root record* is 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.
- An *overwriting record* is 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.
- An *inherited record* has a parent record but is not locally defined in the current table. All values are inherited.
- An *occulting record* specifies that if a parent with same primary key is defined, this parent will not be seen in table descendants.

Inherited fields

The specific inheritance mechanism allows the fetching of the value of a node according to its relationship to other tables.

Lookup mechanism for value

The lookup mechanism for inherited fields values is the following:

1. If the value is locally defined, returns it.
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 further looked up, according to the inheritance behavior of the source node.

Optimize & Refactor

EBX5 provides a built-in UI service for optimizing data set inheritance in your data set tree. This service performs the following functions:

- *Handle duplicated values*: this procedure detects and removes all parameter values that are duplicates of the inherited value.
- *Mutualize common values*: this procedure detects and mutualizes the common values among the descendants of a common ancestor.

Procedure details

Data sets are processed from the bottom up, which means that if the service is run on the data set 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 data sets at level $N+2$ to determine if they can be optimized with respect to the data sets 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 data set on which the service is run. This means that optimization and refactoring is not performed between the target data set and its own parent.
- Table optimization is done for records with the same primary key.
- Inherited fields are not optimized.
- *Optimization and refactoring functions do not modify the resolved view of a data set if it is activated.*

Service availability

The 'Optimize & Refactor' service is available on data sets that have child data sets and also have the property 'Activated' set to 'No' in its data set information.

The service is available to any profile with write access on current data set values. It can be disabled by setting a restrictive access right on a profile.

Note

For performance reasons, access rights are not verified on every node and table record.

CHAPTER 44

Criteria Editor

EBX5 integrates a specific criteria editor. It is used in order to define filters on table, validation and computation rules on data. This editor is based on XPath 1.0 Recommendation.

Two kinds of criteria exist: atomic criteria and conditional blocks.

See also: [Supported XPath syntax](#)

Conditional Blocks

Conditional blocks, or conditional criteria, are made up of atomic criteria and conditional blocs. They express a condition over the criteria. Four kinds of block are defined:

- *Don't match any criteria*: no criteria must be matched.
- *Don't match one of criteria*: one of the criteria must not be matched.
- *Match all criteria* all criteria must be matched.
- *Match one of criteria* one of the criteria must be matched.

Atomic Criteria

An atomic predicate. It is defined by a field, an operator and an expression (a value or a XPath formula).

Field	Specify the field of the table on which the criterion applies.
Operator	Specify the operator used. Available operators depend on the field type.
Expression	Specify the value or expression (see section below).

Expression

The expression can either be a fixed value or a formula. While creating a filter, only fixed value is authorized. A formula can be created using the assistant interface while creating validation or computation rules.

Known limitation: field formula does not check input values, only syntax and path are checked.