



Product Documentation

EBX5 Version 5.4.0 Fix 002

Table of contents

User Guide

Introduction

1. How EBX5 works.....	11
2. Using the EBX5 user interface.....	15
3. Glossary.....	19

Data models

4. Introduction to data models.....	30
5. Using the Data Models area user interface.....	33

Implementing data models

6. Creating a data model.....	37
7. Data model configuration.....	39
8. Implementing the data model structure.....	43
9. Properties of data model elements.....	49
10. Data validation controls on elements.....	61
11. Working with existing data models.....	65

Publishing and versioning data models

12. Publishing data models.....	67
13. Versioning embedded data models.....	69

Data spaces

14. Introduction to data spaces.....	72
15. Main actions.....	75
16. Snapshot.....	79
17. Merging a data space.....	83
18. Permissions.....	87

Data sets

19. Introduction to data sets.....	92
20. Main actions.....	95
21. Custom views.....	101
22. Inheritance.....	105
23. Permissions.....	109

Workflow models

24. Introduction to workflow models.....	112
25. Main actions.....	115
26. Workflow modeling.....	119
27. User task.....	125
28. Workflow model publication.....	129

Data workflows

29. Introduction to data workflows.....	132
30. Using the Data Workflows area user interface.....	133
31. Work items.....	137

Managing data workflows

32. Launching and monitoring data workflows.....	141
33. Administration of data workflows.....	143

Data services

34. Introduction to data services.....	148
35. Main actions.....	151

Reference Manual

Integration

36. Overview of integration and extension.....	155
37. Using EBX5 as a web component.....	159
38. User interface services (UI services).....	165
39. Built-in UI services.....	175
40. Data services.....	187

File import and export services

41. XML import and export.....	213
42. CSV import and export.....	217
43. Supported XPath syntax.....	221

Miscellaneous

44. Permissions.....	228
45. Labeling and localization.....	239
46. Extending EBX5 internationalization.....	243
47. Inheritance and value resolution.....	245
48. Criteria Editor.....	251
49. Performance guidelines.....	253

Persistence

50. Overview of persistence.....	262
51. Relational mode.....	267
52. History.....	273
53. Replication.....	281
54. Data model evolutions.....	287

Administration Guide

Installation & configuration

55. Supported environments.....	293
56. Java EE deployment.....	297
57. EBX5 main configuration file.....	311
58. Installing a repository using the Configuration Assistant.....	325
59. Deploying and registering EBX5 add-ons.....	329

Technical administration

60. Repository administration.....	332
61. Front end administration.....	341
62. Users and roles directory.....	353
63. Audit Trail.....	357
64. Data model administration.....	361
65. Data workflow administration.....	363
66. Task scheduler.....	367

Distributed Data Delivery (D3)

67. Introduction to D3.....	374
68. D3 broadcasts and delivery data spaces.....	379
69. D3 administration.....	383

Developer Guide

70. Notes to developers.....[390](#)

Model design

71. Introduction.....[394](#)

72. Packaging EBX5 modules.....[397](#)

73. Types.....[401](#)

74. Tables, filters and selection nodes.....[411](#)

75. Constraints, triggers and functions.....[419](#)

76. Labels and messages.....[439](#)

77. Additional properties.....[445](#)

Java reference

78. Mapping to Java.....[452](#)

79. Tools for Java developers.....[459](#)

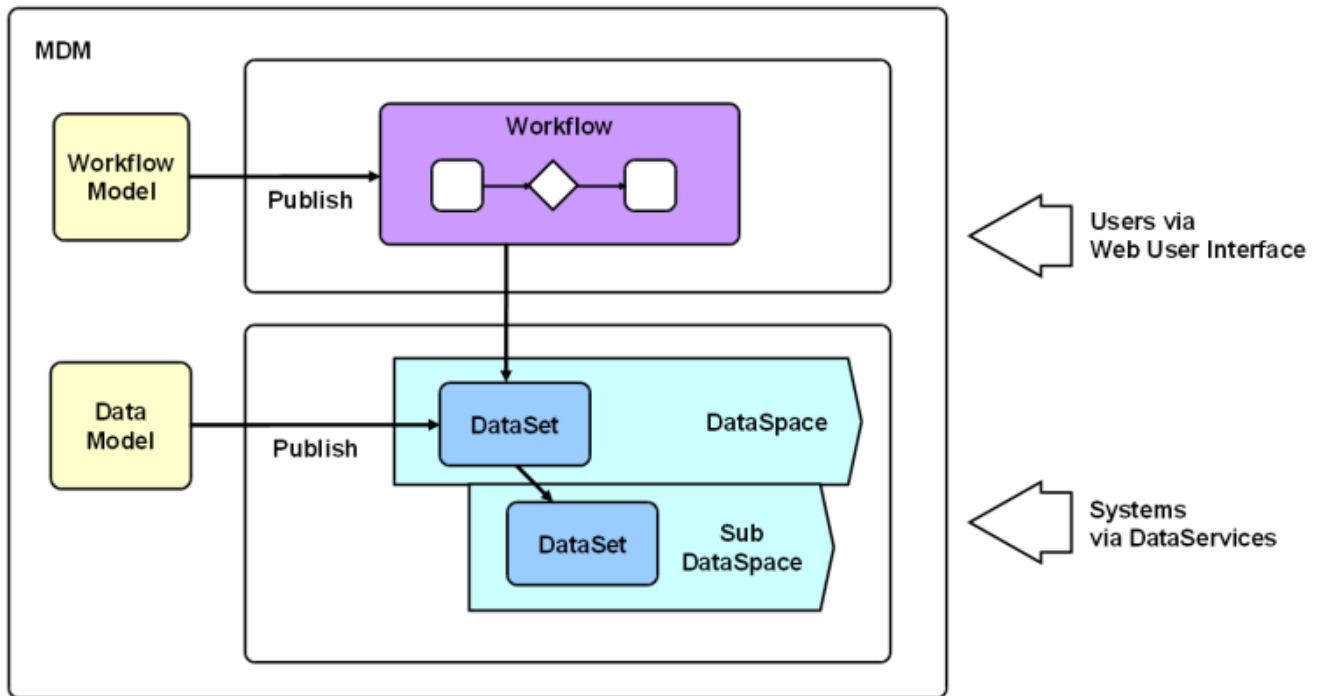
User Guide

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.

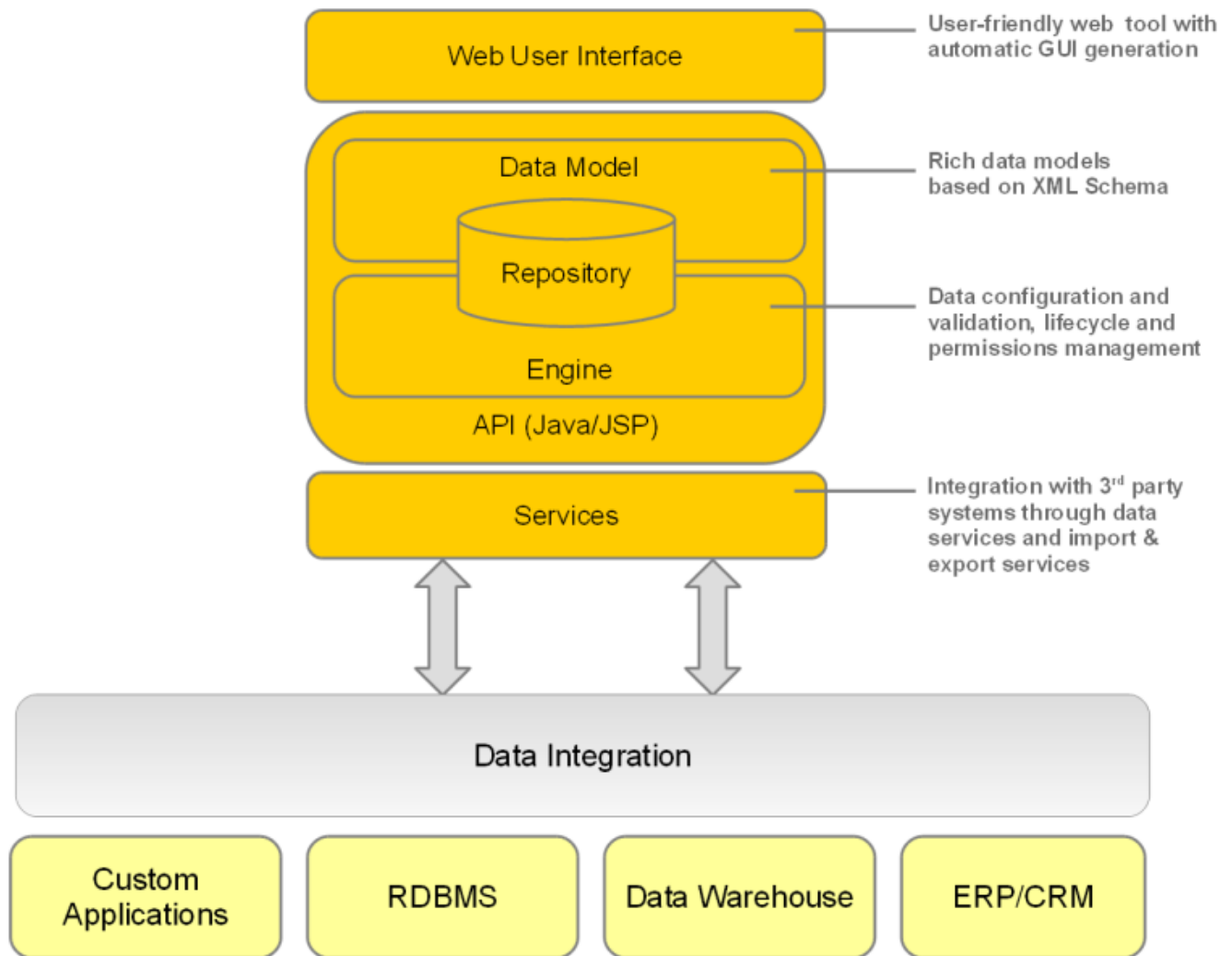
Key words to understand are:

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

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

Architecture

The following schema presents EBX5 architecture.



CHAPTER 2

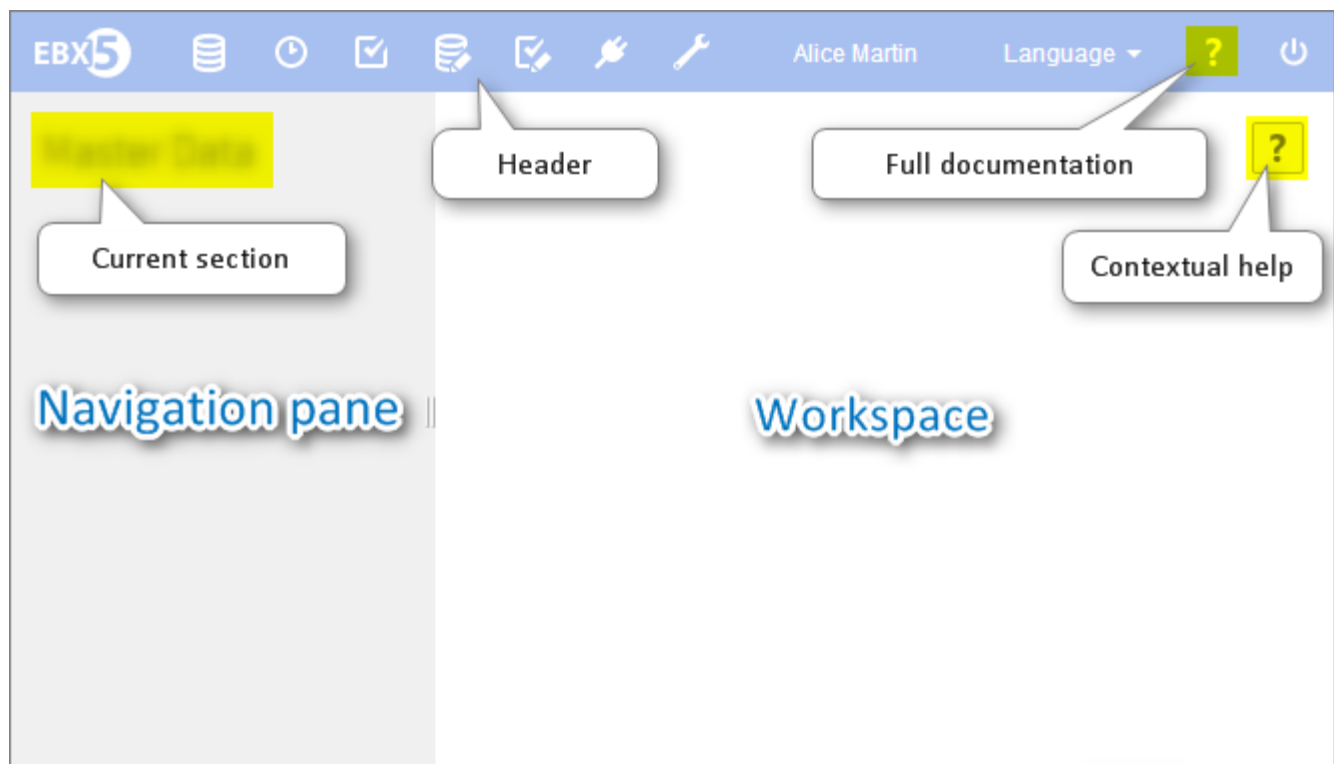
Using the EBX5 user interface

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.

Permalink to Titles

✕

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

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](#).

Related Java API **Profile** [`Profile`]^{API}.

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.

See also [user and roles directory](#), [profile](#).

Related concept [User and roles directory](#).

Related Java API **UserReference** [`UserReference`]^{API}.

role

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

Whenever a role profile is specified in 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.

See also [user and roles directory](#), [profile](#).

Related concept [User and roles directory](#).

Related Java API **Role** [Role]^{API}.

administrator

A predefined role that allows access to the technical administration and configuration of EBX5.

user and roles directory

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

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

Related concept [User and roles directory](#).

Related Java API **Directory** [Directory]^{API}, **DirectoryHandler** [DirectoryHandler]^{API}.

user session

A repository access context that is associated with a user when they have been authenticated against the user and roles directory.

Related concept [User and roles directory](#).

Related Java API **Session** [Session]^{API}.

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](#).

Related Java API **SchemaNode** [SchemaNode]^{API}.

primary key

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

Primary keys are denoted by the icon .

Related concept [Tables definition](#).

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](#).

Related concept [Constraints, triggers and functions](#).

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](#).

Related Java API **SchemaNode** [SchemaNode]^{API}.

reusable type

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

validation rule

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

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 .

Related concept [Inheritance and value resolution](#).

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

Main documentation section [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.

Related technical reference [History](#).

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.

Related technical reference [History](#).

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](#).

token

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

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](#)

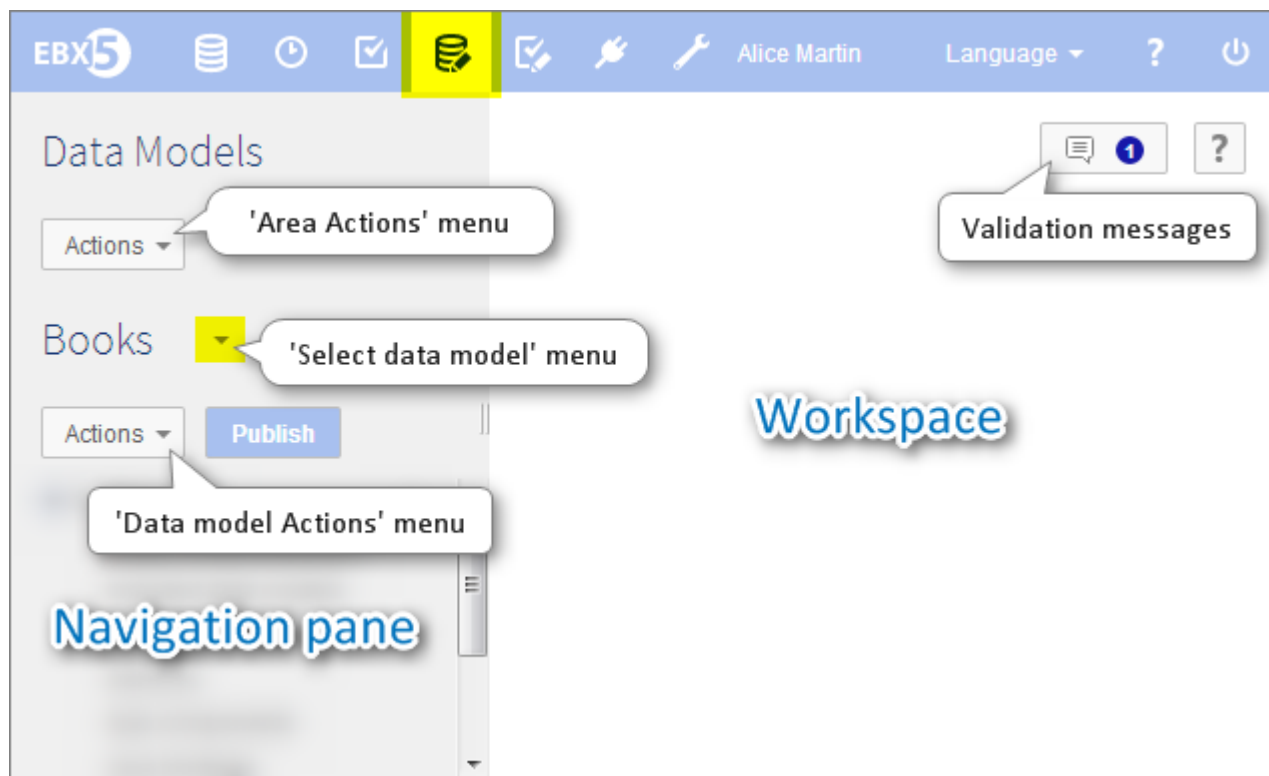
- [*Data sets*](#)

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.
Data model properties	The technical properties of the 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.
Component library	Defines the Java components available in the model. These provide programmatic features that can be used in the model, such as programmatic constraints, functions, and UI beans.
Services	The services available for use in the data model.
Ajax components	The Ajax components available for use in the data model.
Java bindings	The properties of the Java types to be generated from the data model.
Replications	The replication units available for this data 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.

Selecting a data model type

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

Semantic models

Semantic models enable the full use of data management features, such as life cycle management using data spaces, provided by EBX5. This is the default type of data model.

Relational models

Relational models are used when the tables created from a data model will eventually be mapped to a relational database management system (RDBMS). The primary benefit of using a relational model is the ability to query the tables created from the data model using external SQL requests. However, employing a relational model results in the loss of functionalities in EBX5 such as inheritance, multi-valued fields, and the advanced life cycle management provided by data spaces.

Note

A relational data model can only be used by a single data set, and the data space containing the data set must also be declared as being relational.

See also:

- [Relational mode](#)
- [Data spaces](#)

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:

Module name	Defines the module that contains the resources that will be used by this data model. This is also the target module used by the data model publication if publishing to a module.
Module path	Physical location of the module on the server's file system.
Sources location	The source path used when configuring Java components in the 'Component library'. If this path is relative, it will be resolved using the 'Module path' as the base path.
Publication mode	<p>Whether to publish the data model as an XML Schema Document within a module or as a publication completely embedded in the EBX5 repository. Embedded data models offer additional functionality such as versioning and rollback of publications.</p> <p>See Publication modes for more information.</p> <p>Model path in module: Defines the target file for the data model generation. It must start with '/'.</p>
Data set inheritance	<p>Specifies whether data set inheritance is enabled for this data model. Data set inheritance is disabled by default.</p> <p>See inheritance for more information.</p>
Documentation	<p>Documentation of the data model defined by a Java class. This Java class can programmatically specify labels and descriptions for the elements of the data model. The labels and descriptions defined in this Java class are displayed in associated data sets in preference to the ones defined locally on an element.</p> <p>See Dynamic labels and descriptions for more information.</p>
Special extensions	Access permissions defined by programmatic rules in a Java class.
Disable auto-increment checks	<p>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.</p> <p>See Auto-incremented values for more information.</p>

Included data models

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

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

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

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

See also: [Including external data models](#)

Replication of data to relational tables

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

To define a replication unit through the user interface, create a new record in the 'Replications' table under the data model configuration in the navigation pane. Each replication unit record is associated


with a particular data set in a given data space. A single replication unit can cover multiple tables, as long as they are in the same data set. A replication unit defines the following information:

Name	Name of the replication unit. This name identifies a replication unit in the current data model. It must be unique in this data model.
Data space	Specifies the data space relevant to this replication unit. It cannot be a snapshot or a relational data space.
Data set	Specifies the data set relevant to this replication unit.
Refresh policy	<p>Specifies the data synchronization policy. The possible policies are:</p> <ul style="list-style-type: none"> • On commit: The replicated table content in the database is always up to date with respect to its source table. Every transaction that updates the EBX5 source table triggers the corresponding insert, update, and delete statements on the replicated table. • On demand: The replicated table in the database is only updated when an explicit refresh operation is performed.
Tables	<p>Specifies the tables in the data model to be replicated in the database.</p> <p>Table path: Specifies the path of the table in the current data model that is to be replicated to the database.</p> <p>Table name in database: Specifies the name of the table in the database to which the data will be replicated. This name must be unique amongst all replications units.</p>

See also: [Replication](#)

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> <p>If a validation rule is defined on a table, it will be considered as a 'constraint on table' and each record of the table will be evaluated against it at runtime. See Constraints on table for more information.</p>

Basic properties for fields

The following basic properties are specific to fields:

Default value	<p>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.</p> <p>See Default value for more information.</p>
Conversion error message	<p>Internationalized messages to display to users when they enter a value that is invalid for the data type of this field.</p>
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

Common advanced properties

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

UI bean	<p>This property is available for all elements except tables. Specifies a Java class to customize the user interface associated with this element in a data set. A UI bean can display the element differently and/or modify its value by extending the UIBeanEditor [UIBeanEditor]^{API} class in the Java API.</p>
Transformation on export	<p>This property is available for fields and for groups that are terminal nodes. Specifies a Java class that defines transformation operations to be performed when exporting an associated data set as an archive. The input of the transformation is the value of this element.</p> <p>See NodeDataTransformer [NodeDataTransformer]^{API} for more information.</p>
Access properties	<p>Defines the access mode for the current element, that is, if its data can be read and/or written.</p> <ul style="list-style-type: none"> • 'Read & Write' corresponds to the mode <code>RW</code> in the data model XSD. • 'Read only' corresponds to the mode <code>R-</code> in the data model XSD. • 'Not a data set node' corresponds to the mode <code>CC</code> in the data model XSD. • 'Non-terminal node' corresponds to the mode <code>--</code> in the data model XSD. <p>See Access properties for more information.</p>
Default view	<p>Specifies:</p> <ul style="list-style-type: none"> • Whether or not this element is shown in the default view of a data set. Setting the property to 'Hidden' will hide the element from the default view of a data set without having to define specific access permissions. • Whether or not this element is shown in the text search of a data set. Setting the property to 'Hidden in text search' will hide the element in the text search.

See [Default view](#) for more information.

Comparison mode

Defines the comparison mode associated with the element, which controls how its differences are detected in a data set.

- 'Default' means the element is visible when comparing associated data.
- 'Ignored' implies that no changes will be detected when comparing two versions of modified content (records or data sets).

During a merge, the data values of ignored elements are not merged even if the content has been modified. However, values of ignored data sets or records being created during the operation are merged.

During an archive import, values of ignored elements are not imported when the content has been modified. However, values of ignored data sets or records being created during the operation are imported.

See [Comparison mode](#) for more information.

Node category

Defines a category for this element. Categories allow controlling the visibility of data in a data set to users. A node with the category 'Hidden' is hidden by default. Restriction: Category specifications other than 'Hidden' do not apply on table record nodes.

See [Categories](#) for more information.

Advanced properties for fields

The following advanced properties are specific to fields.

Check null input

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

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

Note

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

See [Constraints, triggers and functions](#) for more information.

UI bean

See [Common advanced properties](#).

Function (computed value)

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

A function can be created by implementing the **ValueFunction** [ValueFunction]^{API} interface.

Transformation on export

See [Common advanced properties](#).

Access properties

See [Common advanced properties](#).

Selection node

This property is only available for fields of type 'String'. Defines a record selection based on an XPath expression from a specified data set and table. This can be useful to link to records that are related to the current field. Records that satisfy the criteria will be displayed when the user clicks the generated link. When this property is set, the minimum and maximum number of values properties are automatically set to '0'.

See [Selection nodes](#) for more information.

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.

Auto-incremented values have the following behavior:

- The computation and allocation of the field value are performed whenever a new record is inserted and the field value is yet undefined.
- No allocation is performed if a programmatic insertion already specifies a `non-null` value. Consequently, the allocation is not performed for a record insertion in occulating or overwriting modes.
- If an archive import specifies the value, the imported value takes precedence.
- Whenever possible, the newly allocated value is unique in the scope of the repository.

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

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

See [Auto incremented values](#) for more information.

Default view

See [Common advanced properties](#).

Node category

See [Common advanced properties](#).

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> <p>See History configuration in the repository for more information.</p>
Label	<p>Defines the fields to provide the default and localized labels for records in the table.</p> <p>Can also specify a Java class to set the label programmatically, or set the label in a hierarchy. This Java class must implement either the UILabelRenderer [UILabelRenderer]^{API} interface or the UILabelRendererForHierarchy [UILabelRendererForHierarchy]^{API} interface.</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>See Ergonomics & behavior policy for more information.</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</p>

to have the option to display groups as tabs. Similarly, links must be enabled to have the option to display groups as links.

Default rendering for groups

Specifies the default display rendering mode to use for the groups contained in this table. If a group does not specify a default mode then the default mode defined for this table will be used. Links must be enabled to define the default rendering mode as 'Link'. Select a rendering mode according to network and browser performance. Link mode is lighter as its content is not displayed on the same page, whereas the expanded and collapsed modes are heavier.

Note: When the tabs view is enabled on a table, any groups that would otherwise have used links are automatically converted to collapsed mode. This is done to avoid the inherent display layout complexities that would arise from links and tabs coexisting in the same user interface.

Uniqueness constraints

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

Trigger

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

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

See [Triggers](#) for more information.

Access properties

See [Common advanced properties](#).

Default view

See [Common advanced properties](#).

Node category

See [Common advanced properties](#).

Advanced properties for groups

The following advanced properties are specific to groups.

Value container class (JavaBean)

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

UI bean

See [Common advanced properties](#).

Transformation on export

See [Common advanced properties](#).

Access properties

See [Common advanced properties](#).

Default view

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

Node category

See [Common advanced properties](#).

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.

See [XML schema supported facets](#).

Advanced content validation

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

See also: [Dynamic constraints](#)

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	<p>Location of the table referenced by the foreign key.</p> <p>'Default': current data model.</p> <p>'Other data set': different data set, in the same data space.</p> <p>'Other data space': data set in a different data space.</p>
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	<p>Defines fields to provide the default and localized labels for records in the table.</p> <p>Can also specify a Java class to set the label programmatically if 'XPath expression' is set to 'No'. This Java class must implement the TableRefDisplay [TableRefDisplay]^{API} interface of the Java API.</p>
Filter	<p>Defines a foreign key filter using an XPath expression.</p> <p>Can also specify a Java class that implements the TableRefFilter [TableRefFilter]^{API} interface of the Java API.</p>
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>
Specific constraint (component)	Specifies one or more Java classes that implement the Constraint [Constraint] ^{API} interface of the Java API. See Programmatic constraints for more information.
Specific (component) enumeration	Specifies a Java class to define an enumeration. The class must define an ordered list of values by implementing the ConstraintEnumeration [ConstraintEnumeration] ^{API} interface of the Java API.
Enumeration filled by another node	Defines the possible values of this enumeration using a reference to another list or enumeration element.

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.

See [Validation](#) for detailed information on incremental data validation.

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.

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

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

Note

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

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.

See [Viewing and creating publications](#) for more information on the use of different publications.

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

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

Note

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

Viewing and creating publications

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

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

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

CHAPTER 13

Versioning 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. See Archives directory for more information.
Import archive	Imports the content of an archive into the selected version. The archive to import must contain a data model with the same name as the data model associated with the version.

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

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

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.

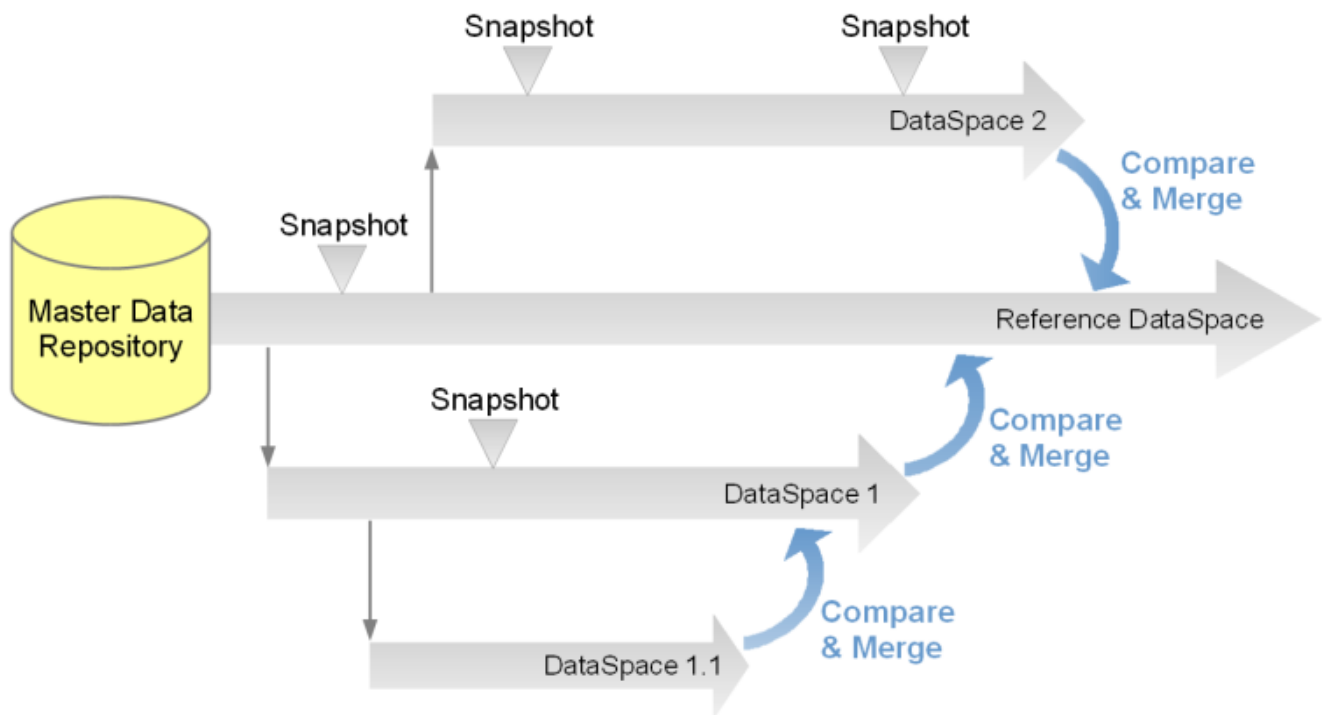
For detailed definitions of terms, refer to the [glossary](#).

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.

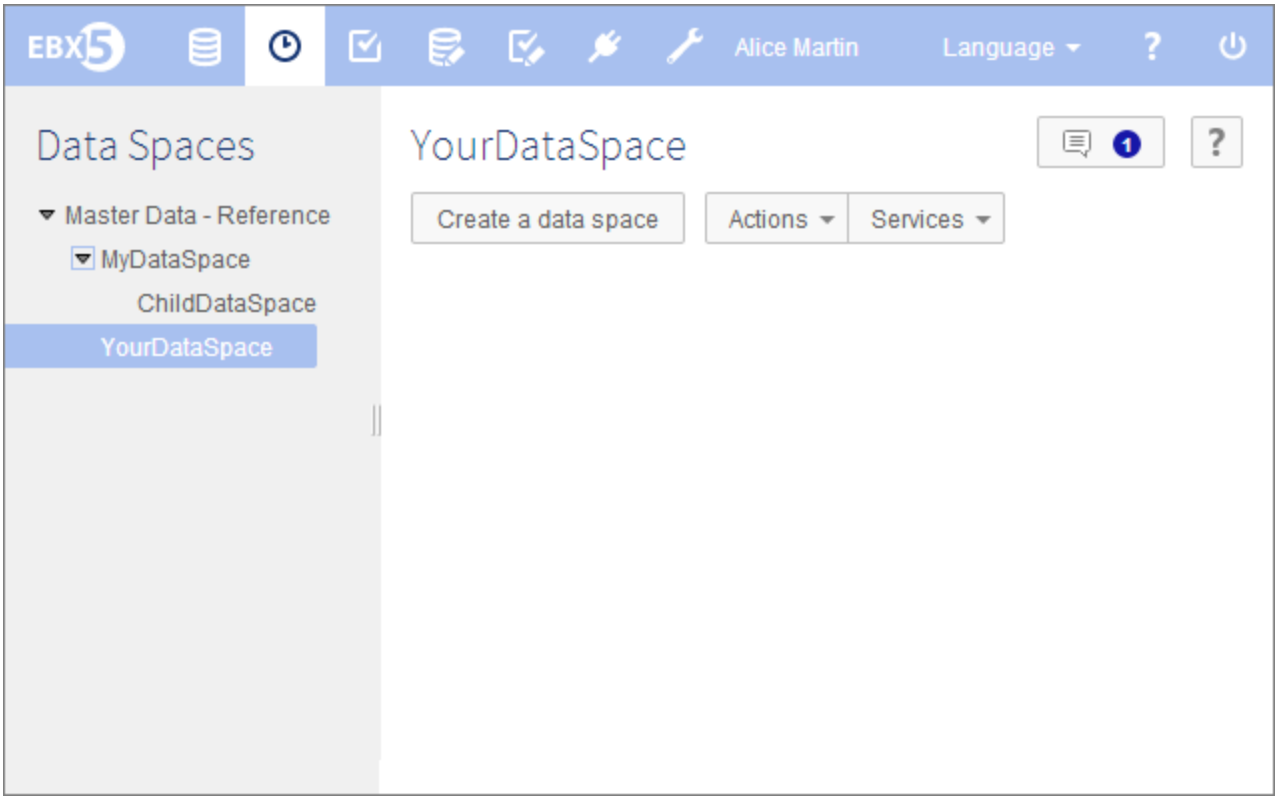
A snapshot allows you to keep the static state of a data space in case you need to revert to a stable version or detect subsequent updates.



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. See also: Relational mode

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.
Loading strategy	<p>Only the administrator can modify this setting.</p> <ul style="list-style-type: none"> • On demand loading and unloading: The main advantage of this strategy is the ability to free memory when needed. The disadvantage is that it implies a load cost when an accessed resource has not yet been loaded since server startup, or if it has been unloaded. This is the default mode. • Forced loading: This mode is recommended for data spaces with long life cycles where snapshots are used heavily. • Forced loading and prevalidation: This mode is recommended for data spaces with long life cycles where snapshots are used heavily and the validation process can take a long time. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Note</p> <p>Any change of loading strategy requires restarting the server.</p> </div>

Child merge policy

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.

Merge policies are:

- **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.

Note

Only the administrator can retrieve the exported archive. See [Archives directory](#) in the Administration Guide for more information.

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

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

The default export policy is **The whole content of the 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 options 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.
Loading strategy	<p>Only the administrator can modify this setting.</p> <ul style="list-style-type: none"> • <i>On demand loading and unloading</i>: The main advantage of this strategy is the ability to free memory when needed. The disadvantage is that it implies a load cost when an accessed resource has not yet been loaded since server startup, or if it has been unloaded. This is the default mode. • <i>Forced loading</i>: This mode is recommended for data spaces with long life cycles where snapshots are used heavily. • <i>Forced loading and prevalidation</i>: This mode is recommended for data spaces with long life cycles where snapshots are used heavily and the validation process can take a long time.

Note

Any modification of the loading strategy requires restarting the server.

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.

Note

This change set review and acceptance stage is bypassed when performing merges using data services or programmatically. For automated merges, all changes in the child data space override the data in the parent 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](#)).

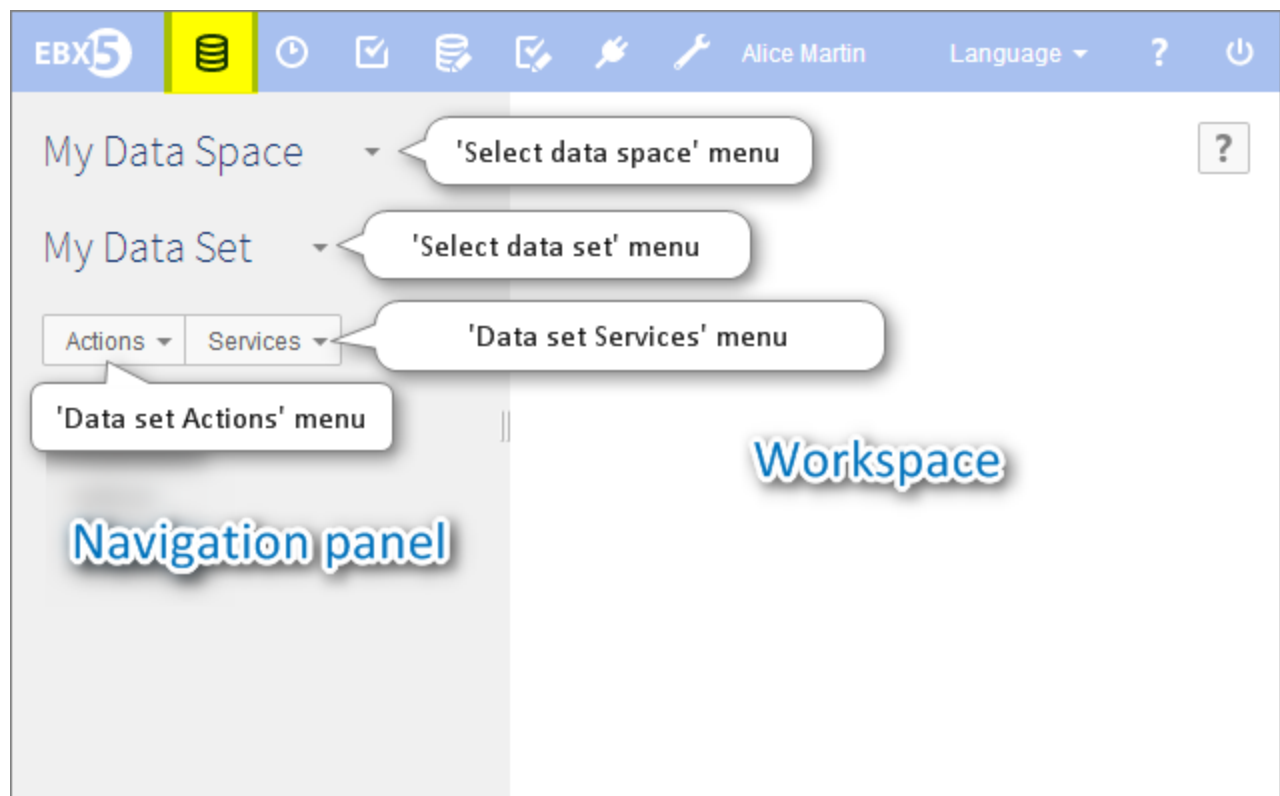
A data set corresponds to data displayed as tables or hierarchies, and that can be filtered using multi-criteria views. Access to the content of a data set can be restricted using permissions rules for users and role (that is, a group of users).

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.

See [Validation](#) for detailed information about incremental data validation.

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:

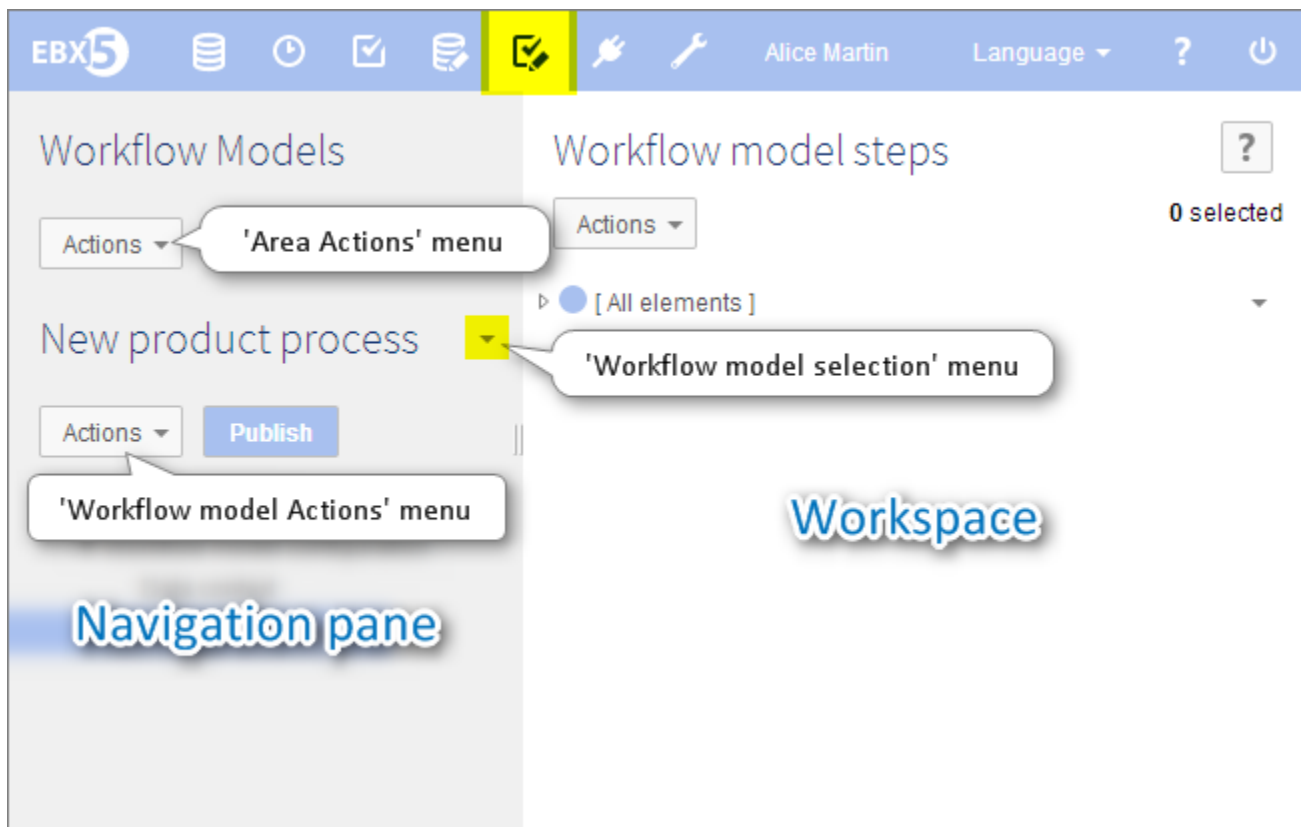
- define tasks to be performed by a user (see [user task](#)) or automatically by the system (see [script task](#)),
- specify everybody's responsibilities (see [creation](#)),
- send notification emails to colleagues, whose involvement is required (see [notification](#)),
- know the limits of your model; 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	<p>Basic task carried out by the user, it has been allocated to, and whose execution allows the workflow to move forward.</p> <p>A work item is related to the execution of a HttpManager Component.</p>
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	<p>Link to access the current work item in the work list, by means of the web component API.</p> <p>This link can be computed only if a current work item is defined and if the URL is configured in Workflow-executions, in the email configuration.</p>
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

Two condition types are available:

- *Library condition*: it must be declared in module.xml and you need to define its label, description and parameters. In definition condition, when a user selects a library condition, the associated parameters are dynamically displayed.
- *Specific condition*: it specifies only its class name in definition condition. Display is not dynamic. The associated class must belong to the module of workflow definition.

Library conditions

Library conditions are classes extending **ConditionBean** [ConditionBean]^{API} as shown in the following **example** [package-summary]^{API}.

It is possible to dynamically set variables of the bean if the bean follows the Java Bean Specification. Variables must be declared as parameters of the bean in module.xml. Data context is not accessible in a Java Bean.

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.

Specific conditions

Specific conditions are classes extending **Condition** [Condition]^{API} as shown in the following **example** [package-summary]^{API}.

It is not possible to set variables of the bean dynamically for specific conditions. But the data context is accessible in the Java Bean.

User task

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

Script task

Two script task types are available:

- *Library script task*: it must be declared in module.xml and you need to define a label, a description and parameters for it, to be able to use it afterwards. In a definition script task, when a user selects a library script task, its associated parameters are displayed dynamically.
- *Specific script task*: it specifies only its class name in the definition script task. Display is not dynamic. The associated class must belong to the module of workflow definition.

Library script tasks

Library script tasks are classes extending **ScriptTaskBean** [ScriptTaskBean]^{API}.

The method **executeScript()** [ScriptTaskBean]^{API} is called when the process reaches the matching step as in the following **example** [package-summary]^{API}. This method must not start any Thread because the process goes forward when this method is over. So, each operation must be synchronous in this method.

It is possible to dynamically set variables of the bean if the bean follows the Java Bean Specification. Variables must be declared as parameters of the bean in module.xml. Data context is not accessible in a Java Bean.

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.

Specific script tasks

Specific script tasks are classes extending **ScriptTask** [ScriptTask]^{API}.

The method **executeScript()** [ScriptTask]^{API} is called when the process reaches the matching step as in the following **example** [package-summary]^{API}. This method must not start any Thread because the process goes forward when this method is over. So, each operation must be synchronous in this method.

It is not possible to dynamically set variables of the bean for specific script tasks. But the data context is accessible in Java Bean.

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
Module name	Module containing specific scripts declarations and providing specific Java resources.
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. If no profile is specified, this means that it is possible to create a new workflow only in a programmatic way (example: from triggers).
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.'

Note

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

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.

Extension

By specifying a specific rule, the task is able to behave dynamically in the workflow execution context, for example if a specific behavior is needed, either for creating work items or for completing the user task.

The specified rule is a JavaBean that must extend **UserTask class** [UserTask]^{API}.

Warning: if a rule is specified and the `handleWorkItemCompletion` method is overridden, the completion strategy is no longer automatically checked. The developer himself must check for the completion in this method.

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](#)

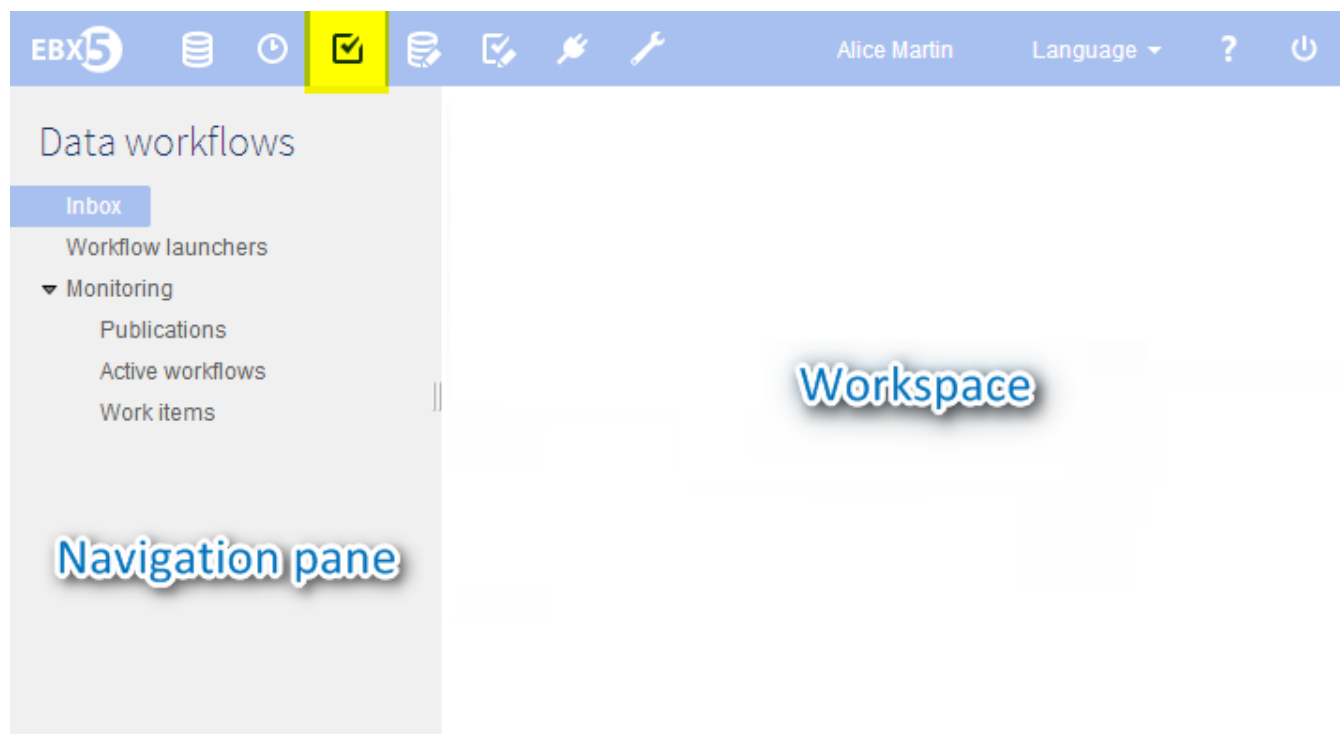
Related concepts: [Workflow models](#)

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.

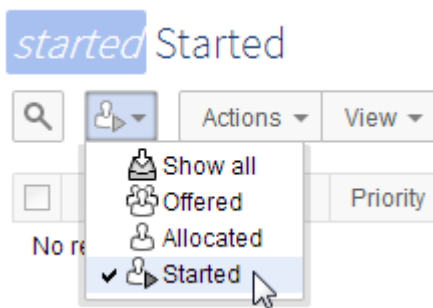


The navigation pane is organized into the following sections:


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.

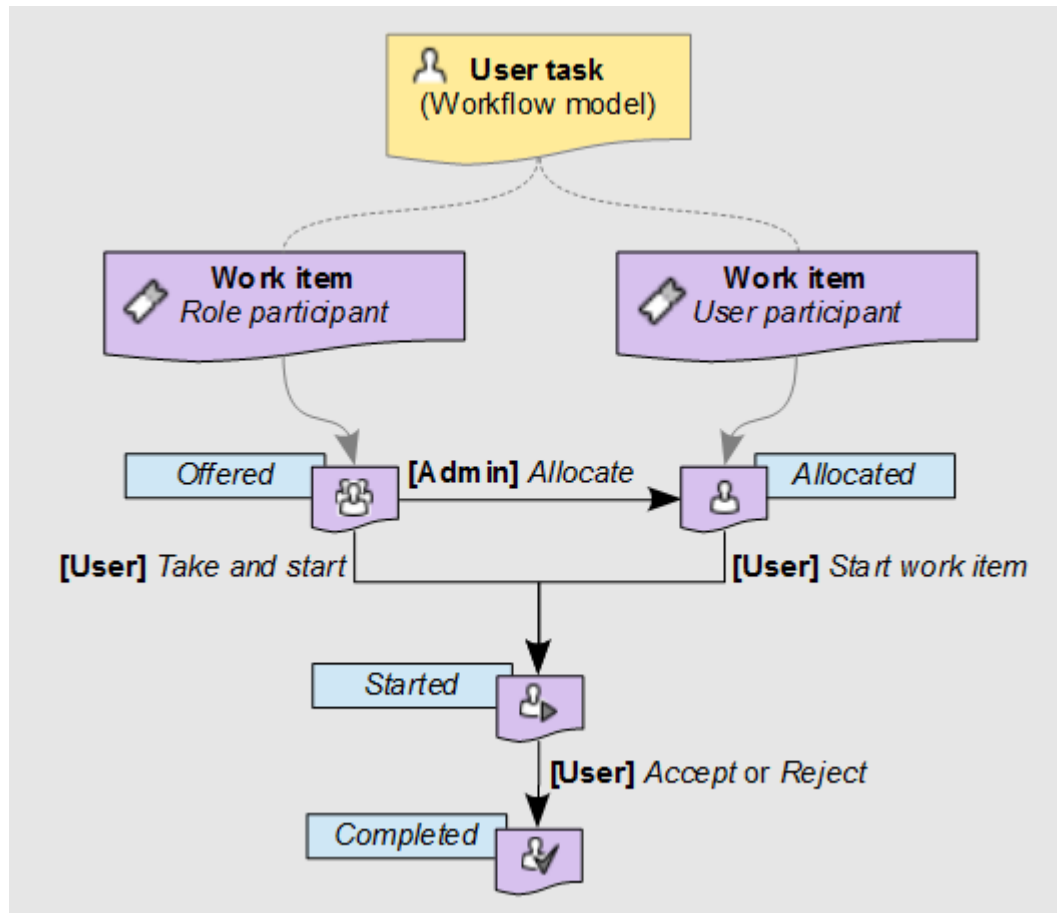
Before a user has claimed the offered work item, a manager of the data workflow can intervene to manually assign the work item to a single user, thus moving the work item to the *allocated* state. Then, when that user begins work on the work item by performing the action 'Start work item', the work item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

See also: [Data workflow administration](#)

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:

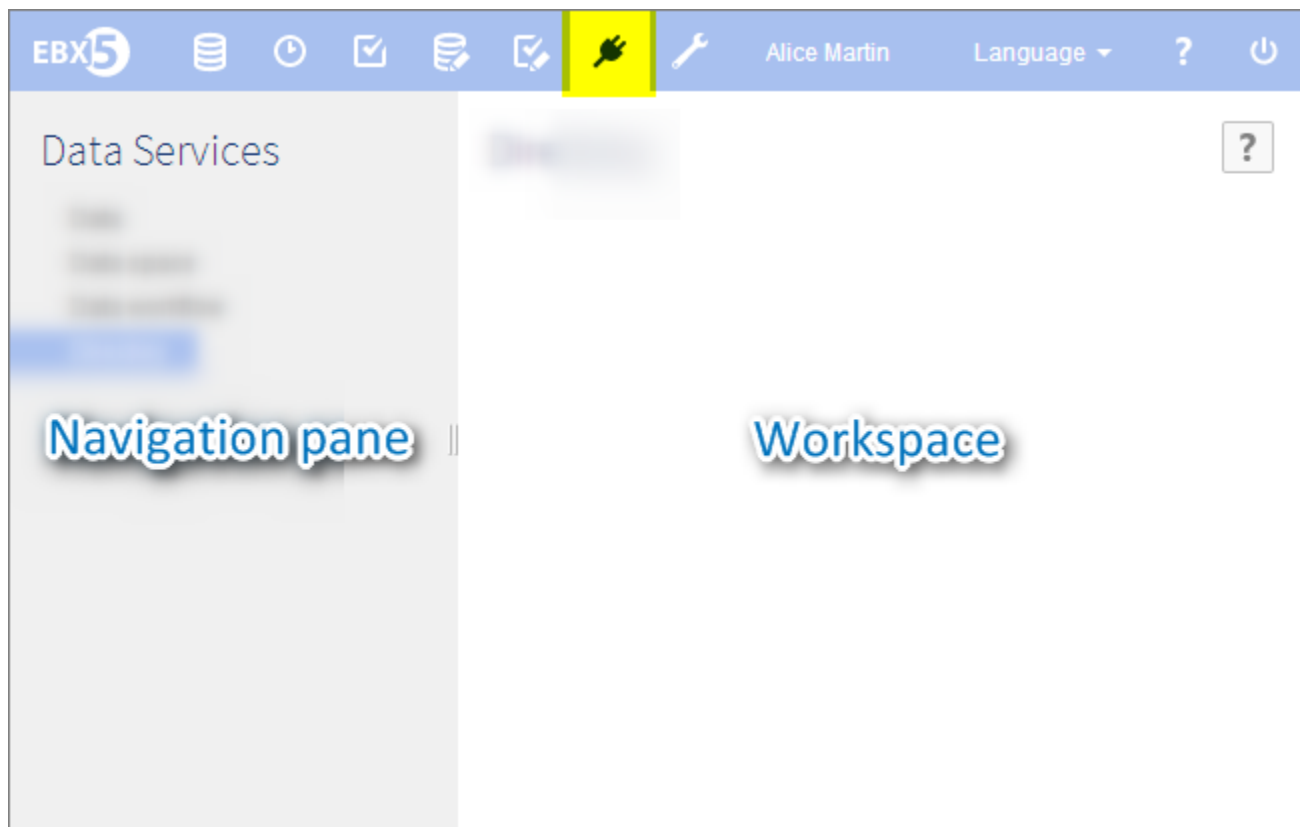
- access data stored in tables (see [data](#)),
- interact with data space (see [data space](#)),
- control workflows (see [workflow](#)),
- define lineage with 3rd-party software (see [lineage](#)).

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

Overview of integration and extension

Several service and component APIs allow you to develop custom extensions for EBX5 and integrate it with other systems.

Using EBX5 as a web component

It is possible to use EBX5 as a user interface web component, by calling it using the HTTP protocol. Such EBX5 web components can be integrated into any application that is accessible through a [supported web browser](#).

A typical use is to integrate EBX5 views into an organization's intranet framework. Web components can also be invoked from the EBX5 user interface using [UI services](#).

See also: [Using EBX5 as a web component](#)

User interface customization

User interface services (UI services)

User interface services (UI services) are HTTP resources (HTML pages, Java Servlets, or JavaServer Pages) that are integrated into the EBX5 interface. They allow users to access custom or advanced functionalities. Examples of UI services include:

- Importing data from an external system
- Performing mass updates on a table

See also: [UI service reference page](#)

Custom layout

A presentation layer provides the ability to override the default layout of forms in the user interface with highly customized form layouts. For example, it is possible to:

- Define field placement and ordering in an HTML layout
- Seamlessly integrate custom layout elements into the existing EBX5 user interface using the provided Java API for styles and pre-built HTML/Ajax artifacts
- Include auto-generated UI components in forms
- Automatically trigger validation and other standard EBX5 transactions
- Leverage user permissions, as in default forms
- Configure rendering parameters for each form field, such as showing or hiding icons, showing or hiding labels, aligning fields vertically or horizontally
- Rename buttons with custom labels

See also: **UIForm** [UIForm]^{API}

UI beans

UI beans are included in the Java API to allow the development of user interface components for fields or groups of fields. When a field or group declares an associated UI bean in its data model, EBX5 automatically generates the corresponding user interface by which users interact with this element.

UI beans can be used for various purposes, such as:

- Masking characters in password fields
- Incorporating JavaScript UI components

To use a UI bean on a field or group, first extend the **UIBeanEditor** [UIBeanEditor]^{API} Java class. Next, declare the UI bean on the element in its data model.

See also:

- **UIBeanEditor** [UIBeanEditor]^{API}
- [Properties of data model elements](#)

Ajax components

Ajax components allow the asynchronous exchange of data with a server without refreshing the currently displayed page.

By using a class on the server side that inherits `UIAjaxComponent`, it is possible for a UI service, UI bean, or custom layout to communicate with the repository or a server without impacting the display of the existing page. On the client side, a JavaScript implementation sends the parameters to the Ajax component, which then returns the data to be displayed.

Ajax components can be used for a wide range of purposes, including:

- Retrieving related data from the EBX5 repository based on a user selection in a form
- Sending requests an external server

See also: ***UIAjaxComponent*** [*UIAjaxComponent*]^{API}

Data services

The data services module provides a means for external systems to interact with EBX5 using the Web Services Description Language (WSDL) standard.

See also: [Data Services reference page](#)

XML and CSV import/export services

EBX5 includes built-in services for importing data from and export data to XML and CSV formats. Imports and exports for XML and CSV can be performed using the user interface, data services, or the Java API.

See also:

- [XML import and export](#)
- [CSV import and export](#)

Programmatic services

Programmatic services allow the development of Java or JSP applications that interact with EBX5 contexts.

Some examples of programmatic services include:

- Importing data from an external source,
- Exporting data to multiple systems,
- Data historization, launched by a supervisory system
- Optimizing and refactoring data if EBX5 **built-in optimization services** [*AdaptationTreeOptimizerSpec*]^{API} are not sufficient.

An easy way to use it is through a JSP. This implies that the process integrates login features so that EBX5 can determine whether the user is allowed to access the data space, data set, etc., or not.

Check out the **ProgrammaticService** [*ProgrammaticService*]^{API} Java class for more information.

CHAPTER 37

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.

See also: [Supported web browsers](#)

Integrating EBX5 web components into applications

A web application that calls an EBX5 web component can be:

1. A non-Java application, the most basic being a static HTML page.

In this case, the application must send an HTTP request that follows the EBX5 web component [request specifications](#).

2. A Java application, for example:

- A Java web application running on the same application server instance as the EBX5 repository it targets or on a different application server instance.
- An EBX5 [UI service](#) or [UI bean](#), in which case, the new session will automatically inherit from the parent EBX5 session.

Note

In Java, the recommended method for building HTTP requests that call EBX5 web components is to use the class `UIHttpManagerComponent` `[UIHttpManagerComponent]`^{API} in the API.

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.

See also: [Scope](#)

Request specifications

Base URL

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

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

Note

The base URL must refer to the servlet `FrontServlet`, defined in the deployment descriptor `/WEB-INF/web.xml` of the web application `ebx.war`.

User authentication and session information parameters

Parameter	Description	Required
login and password, or a <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. See Directory [Directory] ^{API} for more information.	No
trackingInfo	Specifies the tracking information of the new session. Tracking information is logged in history tables. Additionally, it can be used to programmatically restrict access permissions. See AccessRule [AccessRule] ^{API} for more information.	No
redirect	The URL to which the user will be redirected at the end of the component session, when they click on the button 'Close'. The close button is always displayed for record selections, but whether or not it is displayed for all other cases must be specified using the parameter <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> <p>In the Java API, see ServiceKey <code>[ServiceKey]^{API}</code> for more information.</p>	No

Layout parameters

Parameter	Description	Required
scope	Specifies the scope to be used by the web component. Value can be <code>full</code> , <code>data</code> , <code>dataspace</code> , <code>dataset</code> or <code>node</code> . See UIHttpManagerComponent.Scope [UIHttpManagerComponent.Scope] ^{API} for more information.	No, default will be computed according to target selection.
closeButton	Specifies how to display the session close button. Value can be <code>logout</code> or <code>cross</code> . See UIHttpManagerComponent.CloseButtonSpec [UIHttpManagerComponent.CloseButtonSpec] ^{API} for more information.	No. If scope is not <code>full</code> , no close button will be displayed by default.
viewFeatures	Specifies which features to display in a tabular or a hierarchy view. Syntax: prefix [":" features [*("," features)]] Prefix: "hide"/"show" Features: "create", "views", "selection", "filters", "services", "refresh", "title", "breadcrumb" See UIHttpManagerComponent.ViewFeatures [UIHttpManagerComponent.ViewFeatures] ^{API} for more information.	No.
recordFeatures	Specifies which features must be displayed in a record form. Syntax: prefix [":" features [*("," features)]] Prefix: "hide"/"show" Features: "services", "title", "breadcrumb", "save", "saveAndClose", "close", "revert" See UIHttpManagerComponent.RecordFeatures [UIHttpManagerComponent.RecordFeatures] ^{API} for more information.	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 38

User interface services (UI services)

Overview

A user interface service (UI service) is an HTTP resource, such as a JSP or Java servlet, that is integrated into EBX5. Through UI services, users can perform specific and advanced functions directly from the user interface.

The following types of UI services exist:

- [Built-in UI services](#) provided by EBX5,
- Custom UI services on data sets declared in the data model,
- Custom UI services on data spaces or snapshots that are declared in a module and are available to all data spaces or snapshots. These can be disabled on specific data spaces or snapshots using service permissions.

Additionally, to integrate a UI service with a workflow, it can be declared as an *interaction*. See [Integration of UI services with workflows](#) in this chapter for more information.

Accessing UI services

Depending on the nature of UI services and their declarations, users can access them in several ways:

- Directly in the EBX5 user interface, from a **Services** or **Actions** menu,
- From a data workflow, in which case the UI service must be declared as an interaction,
- From the Java API, using `UIHttpManagerComponent` [`UIHttpManagerComponent`]^{API}.

UI services on data sets

UI services can be declared in a specific data model contained in a module, which makes them available to data sets that implement that data model. These UI services can be defined to be executable on the

entire data set, or on tables or record selections in the data set. They are launched in the user interface as follows:

- For data sets, from the **Services** menu in the navigation pane.
- For tables, from the **Actions** menu in the workspace.
- For record selections, from the **Actions** menu in the workspace.

Common use cases

Common uses for UI services on data sets, tables, and record selections are:

- Updating a table or a user selection of table records. For example, the field values of a column 'Product price' can be adjusted by a ratio that is specified by the user.
- Importing data from an external source into the current data set.
- Exporting the selected records of a table.
- Implementing specific events in the life cycle of MDM entities. For example, the creation of a product, which impacts several tables, or the "closure" of a product at a given date.
- Displaying statistics on a table.

Declaration and configuration

A UI service on a data set, a table, or a record selection must be declared in the associated data model under the element `xs:complexType/xs:annotation/xs:appinfo/osd:service`.

Note

The data model must be packaged in a module.

Example

The following example declares the UI service 'Distribution' on a table:

```
<xs:complexType name="Distribution">
  <xs:annotation>
    <xs:appinfo>
      <osd:service resourcePath="/Distribution/Distribution.jsp"
        activatedForPaths="/root/Product" orderInMenu="1"/>
    </xs:appinfo>
    <xs:documentation xml:lang="en-US">
      Distribute data for table 'Product'
    </xs:documentation>
  </xs:annotation>
</xs:complexType>
```

Properties

Property	Definition	Mandatory
resourcePath	<p>Attribute that identifies the JSP or servlet resource to be called. The identifier is a path relative to the root of the EBX5 module containing the data model. It must begin with '/'. See also:</p> <ul style="list-style-type: none"> • ServiceContext [<i>ServiceContext</i>]^{API} • ServletContext.getRequestDispatcher(String) 	Yes
activatedForPaths	<p>Attribute that specifies where the UI service will be available. The list of paths is white-space delimited. Each path must be the absolute XPath notation of the node. It is possible to activate the UI service on nodes or on data sets. The path '/' activates the UI service globally on the data set. On <code>osd:table</code>, an additional syntax allows you to specify whether the UI service is activated globally on the table or on record selections. If the path to the table is <code>/root/myTable</code>:</p> <ul style="list-style-type: none"> • <code>/root/myTable</code> activates the UI service globally on the table. • <code>/root/myTable{n}</code> activates the UI service only if exactly 'n' records are selected, where 'n' is a positive integer. • <code>/root/myTable{+}</code> activates the UI service if one or more records are selected (an unbounded selection). 	No. If not defined, activates as a global UI service that is not attached to a particular node.
activatedForPathRegexp	<p>Attribute that specifies where the UI service will be available based on a regular expression. The UI service will be available on all nodes whose full path in the data model match the regular expression. The format of the regular expression is defined in the Java specification.</p>	No. When set, it is exclusive with <code>activatedForPaths</code> attribute.
class	<p>Attribute that specifies a Java class that implements ServicePermission [<i>ServicePermission</i>]^{API}.</p>	No. If not defined, service has no permission restriction.
osd:confirmation	<p>Element that specifies a confirmation message or that suppresses user confirmation before the UI service is launched. See Confirmation message below.</p>	No. If not defined, a default confirmation message is displayed.
orderInMenu	<p>Attribute that specifies the position of this UI service among the UI services defined in the schema. This position is used for the display order of UI services.</p>	No

Confirmation messages

By default, the user is shown a generic message and must confirm the execution of the UI service. The element `osd:confirmation` allows you to specify a custom localized confirmation message:

```
<osd:service...>
  <osd:confirmation>
    <osd:label xml:lang="fr-FR">
      Voulez-vous lancer le service ?
    </osd:label>
  </osd:confirmation>
</osd:service...>
```

```
<osd:label xml:lang="en-US">
  Do you want to launch the service ?
</osd:label>
</osd:confirmation>
</osd:service>
```

The element `osd:confirmation` can also be used to disable the user confirmation entirely with the attribute `disable="true"`.

Global UI services on all data spaces and snapshots

A UI service can also be declared in such a way that they are available to all data spaces and/or snapshots in the repository. The purpose of such services is to write high-level core business procedures that involve actions such as merges, imports and exports, and validations.

Common use cases

Common uses for UI services on data spaces and snapshots are:

- Importing data from an external source.
- Exporting data to multiple systems.
- Validating a data space or snapshot before exporting it
- Sending messages to monitoring systems before performing a merge

Declaration and configuration

The Java applications, JSPs, and servlets for UI services on data spaces or snapshots must be declared in a module. It is recommended to create a dedicated module for UI services on data spaces and snapshots; these UI services will be available on all data spaces and snapshots, but may have a life cycle that is independent of the data life cycle.

UI services on data spaces and snapshots must be declared in the module configuration file `module.xml`, under the element `services/service`, where `services` is the root of all UI services declared in the module.

See also: [Packaging EBX5 modules](#)

Example

The following example declares a UI service that validates and merges a data space:

```
<service name="AdvancedMerge">
  <resourcePath>/services/advancedMerge.jsp</resourcePath>
  <type>branch version</type>
  <documentation xml:lang="en-US">
    <label>Merge/Validate</label>
    <description>Merge current branch to parent if valid.</description>
  </documentation>
  <confirmation>
    <label>A default confirmation message.</label>
```



```

<label xml:lang="en-US">An English confirmation message.</label>
<label xml:lang="fr-FR">Un message de confirmation en français.</label>
</confirmation>
</service>

```

Properties

Property	Definition	Mandatory
name	Attribute that specifies the name of the UI service. This name must be unique within the scope of a module.	Yes
resourcePath	<p>Element that identifies the JSP or servlet resource to be called. The identifier is a path relative to the root of the EBX5 module containing the schema. It must begin with '/'.</p> <p>See also:</p> <ul style="list-style-type: none"> • ServiceContext [<i>ServiceContext</i>]^{API} • ServletContext.getRequestDispatcher(String) 	Yes
type	<p>Element that defines the availability of the UI service as a whitespace-delimited list.</p> <ul style="list-style-type: none"> • 'branch', which makes the UI service available to <i>data spaces</i> • 'version', which makes the UI service available to <i>snapshots</i> • 'workflow', which makes the UI service available to be defined for user tasks in workflow models <p>If <i>only</i> the value 'workflow' is specified, the UI service will be available on any data space or snapshot, but only in the context of a workflow and not in any Services menus.</p> <p>See Services on a data spaces or a snapshots for data workflows</p>	Yes
documentation	Localized label and description of the UI service, displayed in the menu where the UI service appears.	No
confirmation	Element that specifies a confirmation message or that suppresses user confirmation before the UI service is launched. See Confirmation message below.	No. If not defined, a default confirmation message is displayed.

Confirmation Message

By default, the user is shown a generic message and must confirm the execution of the UI service. The element `confirmation` allows you to specify a custom localized confirmation message:

```

<service name="AdvancedMerge">
  <resourcePath>/services/advancedMerge.jsp</resourcePath>
  <type>branch version</type>
  <documentation xml:lang="en-US">
    <label>Merge/Validate</label>
    <description>Merge current branch to parent if valid.</description>
  </documentation>
  <confirmation disable="true"/>
</service>

```

```
</service>
```

The element `confirmation` can also be used to disable the user confirmation entirely with the attribute `disable="true"`.

Integration of UI services with workflows

Overview

In order to make a UI service available to workflows, an additional declaration must be made in the configuration file `module.xml` of the module that implements the UI service.

Once this declaration has been made, the UI service becomes available in the Workflow Models area where the user defines user tasks. The user then maps the input parameters and output parameters that are specified in the module configuration file `module.xml`.

Once the workflow model has been published and a workflow has been started, for every user starting a work item, the corresponding user session on the application server will be associated with a **persistent interaction object** `[Session]API` that contains the valued input parameters. Through the Java API, the running UI service can access these input parameters and adapt its behavior accordingly. Once the corresponding work item and associated UI services are complete, the completion method must be invoked with the output parameters specifying the resulting state of the work item. For more information, the developer of the UI service should read the JavaDoc of the interface `SessionInteraction` `[SessionInteraction]API`.

Globally, UI services that are integrated into workflows benefit from a declarative and secure specification for a "component-oriented" approach.

Common properties for defining input and output parameters

Parameter declarations in the module configuration file `module.xml` are made under the element `services`. The element `properties` is used for declaring the input and output parameters of the UI service. These parameters will be available for the definition of user tasks that use the UI service.

For instance, the following sample specifies a service on a branch with one input parameter named `branch` and one output parameter named `choice`:

```
<properties>
  <property name="branch" input="true">
    <documentation xml:lang="en-US">
      <label>Branch</label>
      <description>
        This input parameter indicate the branch to work on.
      </description>
    </documentation>
  </property>
  <property name="choice" output="true" />
</properties>
```

The following table summarizes the XML tags to use for defining input and output parameters:

Property	Definition	Mandatory
properties	Element containing property declaration.	Yes. This element is unique for a service.
name	This attribute specifies the name of the property.	Yes
input	This attribute specifies if the property is an input one.	No
output	This attribute specifies if the property is an output one.	No
documentation	Label and description of the property.	No

Services on a data spaces or a snapshots for data workflows

The declaration of a UI service on data spaces and/or snapshots that can be used during a data workflow based on the declaration of a simple UI service on data spaces and/or snapshots. If the `type` element contains the value `workflow` in the whitespace-delimited list, the UI service will be available to the definition of user tasks in workflow models.

For UI services on data spaces in workflows, an element `properties/property` with the attributes `name="branch" input="true"` are also required. Similarly, for UI services on snapshots in workflows, an element `properties/property` with the attributes `name="version" input="true"` are required.

Example

This example declares a UI services that is available both in the **Services** menu for data spaces and for user tasks in workflow models. Note the mandatory inclusion of the element `property` with the attributes `name="branch" input="true"`.

```
<service name="AdvancedMerge">
  <resourcePath>/services/advancedMerge.jsp</resourcePath>
  <type>branch workflow</type>
  <documentation xml:lang="en-US">
    ...Merge a branch to its parent if valid...
  </documentation>
  <properties>
    <property name="branch" input="true"/>
  </properties>
</service>
```

Service on data sets for data workflows

UI services on data sets are declared at the data model-level. However, in order to make them definable for workflow user tasks, their declarations must include additional elements in the module configuration file `module.xml` of the module containing the data model.

Under the element `services/serviceLink`, several properties must be defined. The built-in parameters `branch` and `instance` are required, which respectively identify the data space and contained data set on which the UI service will be run.

Example

The following example declares a data set service that imports a spreadsheet into a table in a data set.

```
<serviceLink serviceName="ImportSpreadsheet">
  <importFromSchema>/WEB-INF/ebx/schema/my-ebx.xsd</importFromSchema>
  <properties>
    <property name="branch" input="true"/>
    <property name="instance" input="true"/>
    <property name="pathToTable" input="true"/>
  </properties>
</serviceLink>
```

Properties

The following table summarizes the elements and attributes under the element `services/serviceLink`:

Property	Definition	Mandatory
<code>serviceName</code>	Attribute of <code>serviceLink</code> that defines the name of the UI service as it is specified in the data model.	Yes
<code>importFromSchema</code>	Element that specifies the path to the data model, relative to the current module (web application).	Yes
<code>properties</code>	Element that defines input and output parameters.	Yes. At least the properties for "branch" and "instance" must be defined.

Service extensions

It is possible to extend UI services that have already been declared, by defining labels and descriptions and adding properties. You can extend built-in UI services, data space or snapshot UI services and data set UI services. However, it is not possible to further extend a service extension.

Service extensions must be declared in the module configuration file `module.xml`, under the element `services/serviceExtension`.

Extending a built-in UI service

```
<serviceExtension name="BuiltinCreationServiceExtension" extends="@creation" >
  <documentation xml:lang="en-US">
    <label>Built in creation service extension</label>
  </documentation>
  <properties>
    ...
  </properties>
```

```
</serviceExtension>
```

Property	Definition	Mandatory
name	Attribute that specifies the name of the service extension.	Yes
extends	Attribute that specifies the name of the built-in UI service being extended. The value is the ServiceKey [ServiceKey] ^{API} of the built-in UI service. For the default built-in UI service 'Access data', this attribute must be empty.	Yes
properties	Element that declares properties being added to the built-in UI service.	No
documentation	Localized labels and descriptions to add to the built-in UI service.	No

Extending data space and snapshot UI services

The service extension and the UI service being extended must be defined in the same `module.xml` module configuration file.

```
<serviceExtension name="AdvancedMergeExtension" extends="AdvancedMerge">
  <documentation xml:lang="en-US">
    ...
  </documentation>
  <properties>
    ...
  </properties>
</serviceExtension>
```

Property	Definition	Mandatory
name	Attribute that specifies the name of the service extension.	Yes
extends	Attribute that specifies the name of the UI service being extended.	Yes
properties	Element that declares properties being added to the UI service.	No
documentation	Localized labels and descriptions to add to the UI service.	No

Extending data set UI services

Since the service extension and the UI service being extended must be defined in the same `module.xml` module configuration file, you must first declare the UI service as an interaction using element `serviceLink`.

```
<serviceExtension name="ImportSpreadsheetExtension" extends="ImportSpreadsheet"
  fromSchema="/WEB-INF/ebx/schema/my-ebx.xsd">
```

```

<documentation xml:lang="en-US">
  ...
</documentation>
<properties>
  ...
</properties>
</serviceExtension>

```

Property	Definition	Mandatory
name	Attribute that specifies the name for the service extension.	Yes
extends	Attribute that specifies the name of the UI service being extended.	Yes
fromSchema	Attribute that specifies the path to the schema that defines the UI service.	Yes
properties	Element that declares properties being added to the UI service.	No
documentation	Localized labels and descriptions to add to the UI service.	No

CHAPTER 39

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
- as extended UI services when used with service extensions
- when using EBX5 as a web component

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

See also:

- ***UIHttpManagerComponent*** [*UIHttpManagerComponent*]^{API}
- ***ServiceKey*** [*ServiceKey*]^{API}

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 40

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

Enabling and configuring data services

Data services are enabled by deploying the module `ebx-dataservices` along with the other EBX5 modules. See [Java EE deployment overview](#) for more information.

For specific deployment, for example using reverse-proxy mode, the URL to `ebx-dataservices` must be configured through lineage administration.

The default method for accessing data services is over HTTP, although it is also possible to use JMS. See [JMS configuration](#) and [Accessing data services using JMS](#) for more information.

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>
```

For more information, see **Session.getTrackingInfo()** [Session]^{API} in the Java API.

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>
```

Accessing data services using JMS

The JMS architecture relies on one mandatory queue and one optional queue, see configuration [JMS](#). The mandatory queue is the input queue. Request messages must be put in the input queue, and response messages are put by EBX5 in the `replyTo` queue of the JMS request. The optional queue is the failure queue which allows you to replay an input message if necessary. If the queue is set in the configuration file and an exception occurs while handling a request message, this input message will be copied in the failure queue.

The relationship between a request and a response is made by copying the `messageId` message identifier field of the JMS request into the `correlId` correlation identifier field of the response.

JMS endpoints must be defined in the Lineage administration to provide them in the WSDL. If no specific endpoint is given, the default value will be `jms:queue:jms/EBX_QueueIn`.

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.

- Authentication based on HTTP Request and a call to the method **Directory.authenticateUserFromHttpRequest** [Directory]^{API}. This requires an override of the implementation of this method. For example, it can extract a password-digest or a ticket from an HTTP request.
- 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 QWxhZGRpbjpvYVUHNlc2FtZQ==

- 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>
```

To handle this non-default header, you must implement the method: **Directory.authenticateUserFromSOAPHeader()** [Directory]^{API}.

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).

Monitoring

Data service events can be monitored through the log category `ebx.dataServices`, as declared in the EBX5 main configuration file `ebx.properties`. For example, `ebx.log4j.category.log.dataServices= INFO`, `ebxFile:dataservices`.

See [Logging](#) for more information.

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	<p>This parameter can be combined with the predicate parameter as a logical AND operation.</p> <p>The behavior of this parameter is described in the section EBX5 as a web component.</p> <p>It cannot be used if the 'viewId' parameter is used, and cannot be used on hierarchical views.</p>	No
viewId	<p><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.</p> <p>This parameter cannot be used if the 'viewPublication' parameter is used.</p>	No
disableRedirectionToLastBroadcast	<p>This property is available for all data service operations.</p> <p>If <code>true</code>, access to a delivery data space on a D3 master node is not redirected to the last broadcast snapshot. Otherwise, access to such a data space is always redirected to the last snapshot broadcast.</p> <p>If this parameter is not present, the default is <code>false</code> (redirection on a D3 master enabled), unless the configuration property ebx.dataservices.disableRedirectionToLastBroadcast.default has been set.</p> <p>If the specified data space is not a delivery data space on a D3 master node, this parameter is ignored.</p>	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 .	

See also: [Optimistic locking](#)

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}>

```

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'. See Resolved mode [DifferenceHelper] ^{API} for more information.	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.

See also: *DifferenceHelper* [\[DifferenceHelper\]](#)^{API}

Get changes responses

Changes between two data sets:

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

Changes between two tables:

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

    <data>
      <XX>
        <TableName>
          <a>AVALUE2</a>
          <b>BVALUE2.1</b>
          <c>CVALUE2.1</c>
          <d>DVALUE2</d>
        </TableName>
      </XX>
    </data>
  </updated>
  <deleted>
    <predicate>./a='AVALUE1'</predicate>
  </deleted>
</ns1:getChanges_{TableName}Response>
```


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 count, select, getChanges, getCredentials, insert, delete or update.</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.

See also:

- **Procedure** [*Procedure*]^{API}

- **Repository** [*Repository*]^{API}

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:

Element	Description	Required
branch	Identifier of the target data space on which the operation is applied. When not specified, the 'Reference' data space is used except for the merge data space operation where it is required.	One of either this parameter or the 'version' parameter must be defined. Required for the data space merge 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	<p>Defines the owner.</p> <p>Must respect the inner format as returned by Profile.format() [Profile]^{API}.</p>	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

<i>Element</i>	<i>Description</i>	<i>Required</i>
label (nested under the documentation element)	Label of the data space or the snapshot to create.	No
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>
      </reportItem>
    </details>
  </validationReport>
</ns1:validateInstance_Response>
```

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

Create a data space

Create data space request

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

Create data space response

```
<ns1:createBranch_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>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	<p>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.</p> <p>When this parameter is not specified in the request, the default value is 'false'. It is possible to redefine the default value by specifying the property <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> in ebx.properties.</p> <p>See Deleting data and history for more information.</p>	No
deleteHistoryOnMerge	<p>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'.</p> <p>When this parameter is not specified in the request, the default value is 'false'. It is possible to redefine the default value by specifying the property <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> in ebx.properties.</p> <p>See Deleting data and history for more information.</p>	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	<p>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.</p> <p>When this parameter is not specified in the request, the default value is 'false'. It is possible to redefine this default value by specifying the property <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> in ebx.properties.</p> <p>See Deleting data and history for more information.</p>	No
deleteHistoryOnClose	<p>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'.</p> <p>When this parameter is not specified in the request, the default value is 'false'. It is possible to redefine the default value by specifying the property <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> in ebx.properties.</p> <p>See Deleting data and history for more information.</p>	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>
```

Replication refresh

Replication refresh request

```
<m:replicationRefresh_${schema} xmlns:m="urn:ebx-schemas:dataservices_1.0">
  <branch>String</branch>
  <instance>String</instance>
  <unitName>String</unitName>
</m:replicationRefresh_${schema}>
```

with:

Element	Description	Required
branch	See the description under Common parameters .	Yes
instance	See the description under Common parameters .	Yes
unitName	Name of the replication unit. <i>See also: Replication refresh information</i>	Yes

Replication refresh response

```
<ns1:replicationRefresh_${schema}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <status>00</status>
</ns1:replicationRefresh_${schema}Response>
```

with:

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

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
processInstanceCid	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 41

XML import and export

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

Imports and exports can also be done programmatically.

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.

See the data services operations [update](#) and [insert](#), as well as **setByDelta** [ImportSpec]^{API} in the Java API for more information.

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. <p>See also EBX5 whitespace management for data types.</p>
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 42

CSV import and export

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

Imports and exports can also be done programmatically.

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.

Programmatic CSV imports are performed using the classes **ImportSpec** `[ImportSpec]API` and **ExportImportCSVSpec** `[ExportImportCSVSpec]API` in the Java API.

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.

Programmatic CSV exports are performed using the classes **ExportSpec** `[ExportSpec]API` and **ExportImportCSVSpec** `[ExportImportCSVSpec]API` in the Java API.

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.

Using programmatic services, the specific value `ebx-csv:nil` can be assigned to strings with values set to `null`. If this is done, the `null` string values will not be replaced by empty strings during round trip export-import procedures. See **ExportImportCSVSpec.setNullStringEncoded()** `[ExportImportCSVSpec]API` in the Java API for more information.

Date, time & dateTime format

The following date and time formats are supported:

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

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 43

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)]`.

Due to the strong dependence of predicates on the data model node and the node type of the criterion, the path portion of the atomic predicate expression (left-hand side) must be a node path and cannot be an XPath formula. For example, the expression `/table[floor(./a) > ceiling(./d)]` is not valid.

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

In EBX5, the foreign keys are grouped into a single field with the [osd:tableRef](#) declaration.

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

Example

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

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

See **primary key** [PrimaryKey]^{API} for more information.

- `/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 44

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.

Such information is defined by [EBX5 Directory](#).

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](#)).

Programmatic permission rules are created by developers (see section [Definition of permissions with programmatic rules](#)).

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](#).

Note. From Java environment, the class **SessionPermissions** [SessionPermissions]^{API} provides access to the resolved 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. Rights on services: indicates if a profile has the right to execute some home services. By default, all data space services are enabled. The administrator or the "super owner" of the target data space or a given user who is allowed to change permissions on the target data space can then modify these data space services permissions for a given profile.
11. 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).

Mode	Authorization
	<ul style="list-style-type: none"> 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.

Definition of permissions with programmatic rules

On a data set, access permissions can be defined with programmatic rules on target nodes. This can be done for all data set's nodes, for table's records, for a specific node and for a given complex node and its child nodes. To define programmatic rules, one should create a class that implements the Java interface **SchemaExtensions** [SchemaExtensions]^{API}.

It is also possible to define a permission for a service by means of a programmatic rule. In order to do so, one should create a class that implements the Java interface **ServicePermission** [ServicePermission]^{API}.

Hence, with these programmatic rules, a field can be updated according to the value of another field, for example.

Note. *Only one programmatic access permission can be defined for a given node, instance or record. Hence, a newly defined programmatic access permission replaces the old existing one.*

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.

We can also notice that programmatic permissions can be invoked as well. These programmatic permissions are always restrictive.

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 access rights defined with programmatic rules

For programmatic rules resolution, there are three levels of resolution: data set, record and node. Since only one programmatic access rule can be set for a given level, the last rule set is the one considered in the resolution procedure.

Rule resolution on data set

On a data set, the last rule set is considered as the resolved rule for the data set.

Rule resolution on record

On a record, the resolved rule is the minimum between the resolved rule on data set and the rule set on this record. See `setAccessRuleOnOccurrence` in **SchemaExtensionContext** [`SchemaExtensionsContext`]^{API} for more details.

Access rule resolution on node

If the node is a record child node, the resolved rule is the minimum between the resolved rule on record and the rule set on this node. If not, the resolved rule is the minimum between the resolved rule on

data set and the rule set on this node. See `setAccessRuleOnNode` in **SchemaExtensionContext** [SchemaExtensionsContext]^{API} for more details.

Display policy in tableRef with access rules on rows

If a row is hidden due to an access rule, it will not appear in tableRef drop-down.

Note. The resolved access right on an instance or an instance's node is the minimum between the resolved access rights defined with EBX5 and the resolved programmatic rules if they exist.

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

Users	Actions list applied
	Duplicate a record

CHAPTER 45

Labeling and localization

Overview

EBX5 offers the ability to handle the labeling and the internationalization of a particular data model.

Localization of the user interaction

In EBX5, language preferences can be set on two scopes:

1. Session: The user can select a default locale on his profile page.
2. Data model: If a data model has been localized to other languages than those natively supported by EBX5, the user can also select one of those languages for that data model. See [Extending EBX5 internationalization](#) for details.

The languages supported by a particular data model must be specified in its [module declaration](#) .

Textual information

In EBX5, most master data entities can have a label, a description, or can correspond to a user message. For example:

- data spaces, snapshots and data sets can have their own label and description (the label is distinct from the identifier, so that it remains localizable and modifiable);
- any node in the model can have a static label and a description;
- values can have a static label when they are enumerated;
- validation messages can be customized and permission restrictions can provide a textual reason;
- each record is dynamically displayed according to its content and the context where it is displayed (in hierarchies, as foreign keys, etc.);

Obviously, all this textual information can be localized according to the declared locales of the module (see previous section).

For more information, see the chapter [Labels and messages](#) , [Tables declaration](#) , [Foreign keys declaration](#) , and other entities' documentation.

Values formats

When a value is displayed to the user, it is formatted according to its type and the formatting policy of the current locale. For example, a date will be displayed in some languages as "dd/MM/yyyy".

The [next section](#) describes how to define and customize formatting policies.

Formatting policies

A formatting policy is used to define how to display values of the [simple types](#) .

For each locale declared on the module (see section above), its formatting policy is configured by means of a file whose path is: /WEB-INF/ebx/{locale}/frontEndFormattingPolicy.xml . For instance, to define the formatting policy for the greek (el), the engine will look for the following path in the module:

```
<code>/WEB-INF/ebx/el/frontEndFormattingPolicy.xml</code>
```

If the corresponding file does not exist, the formatting policy is looked up in the root module, ebx-root-1.0 . If not declared in root module, the formatting policy of en_US is adopted.

The content of the file frontEndFormattingPolicy.xml is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<formattingPolicy xmlns="urn:ebx-schemas:formattingPolicy_1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ebx-schemas:formattingPolicy_1.0 ../schema/ebx-reserved/
formattingPolicy_1.0.xsd">
  <date pattern="dd/MM" />
  <time pattern="HH:mm:ss" />
  <dateTime pattern="dd/MM/yyyy HH:mm" />
  <decimal pattern="00,00,00.000" />
  <int pattern="000,000" />
</formattingPolicy>
```

The elements date, dateTime and time are mandatory.

Syntax for locales

There are two ways to write a locale:

1. The XML recommendation follows the [IETF BCP 47](#) recommendation, which uses the hyphen '-' as a separator.
2. The Java Specification uses the underscore '_' instead of the hyphen.

In any XML file (XSD, formatting policy file, ...) read by EBX5, both syntaxes are allowed. For webpath (that is, path within the webapp), only the Java syntax is allowed. So, a formatting policy file must be located in a directory whose name is a locale respecting the Java syntax.

Extending EBX5 internationalization

EBX5 internationalization can be extended according the [documentation provided here](#).

CHAPTER 46

Extending EBX5 internationalization

Overview

EBX5 offers the following localization capabilities:

1. The ability to internationalize of labels and specific data models, as described in section [Labeling and Localization](#),
2. The extensibility of EBX5 user interface localization

EBX5 native localization

By default, EBX5 provides its built-in user interface and messages in:

1. English (en-US),
2. French (fr-FR).

The built-in localized contents are provided 'as-is', and cannot be changed.

Extending EBX5 user interface localization

EBX5 now supports the localization of its user interfaces into any compatible language and region.

Note: Currently, Latin & Cyrillic characters are supported. Locales that use other character sets may be usable, but are not supported.

Adding a new locale

In order to add a new locale, the following steps must be performed:

- Declare the new locale in the file `ebx.properties`. For example:
`ebx.locales.available=en-US, fr-FR, xx`
 - The first locale is always considered the default.
 - The built-in locales, en-US and fr-FR, may be removed if required.
 - Deploy the following files in the EBX5 class-path:
 - A `com.orchestranetworks.il8n.frontEndFormattingPolicy_xx.xml` file, named
 - A set of localized message files in a resource bundle.
- A complete template set of files is available [here](#).

Specific localization deployment

Specific localized files can be deployed as a JAR file alongside `ebx.jar`.

Known limitations

Default localization

The EBX5 'en-US' and 'fr-FR' localizations cannot be modified.

Non extendable materials

Localization of the following cannot be extended:

- EBX5 product documentation,
- EBX5 HTML editor and viewer,

CHAPTER 47

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*

Furthermore, *functions* can be defined to compute values.

Note that inheritance mechanisms that we describe in this chapter should not be confused with "structural inheritance" that usually applies to models and that is proposed for example in UML class diagrams.

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.

Computed values (functions)

It is also possible to specify in the data model that a node holds a *computed value*. In this case, the specified JavaBean function will be executed each time the value is requested.

The function is able to take into account the current context, for example values of the current record or computation from another table, and to request third-party systems.

For more information on functions, see section [Computed Values](#) .

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.

Data set inheritance declaration

Data set inheritance mechanism is declared as follows in a data model:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ebxbind="urn:ebx-schemas:binding_1.0">
  <xs:annotation>
    <xs:appinfo>
      <osd:inheritance>
        <dataSetInheritance>all</dataSetInheritance>
      </osd:inheritance>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

The element `osd:inheritance` defines the property `dataSetInheritance` for specifying the use of inheritance on data sets based on this data model. The following values can be specified:

- `all`, indicates that inheritance is enabled for all data sets based on the data model.
- `none`, indicates that inheritance is disabled for all data sets based on the data model.

If not specified, inheritance mechanism is disabled.

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

It can explicitly be `null`

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.

Functions are always resolved in the current record context and are not 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.

Field inheritance declaration

Formally speaking, the specific inheritance must be specified on terminal nodes in the underlying data model and is declared as follows:

```
<xs:element name="sampleInheritance" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:inheritance>
        <sourceRecord>
          /root/table1/fkTable2, /root/table2/fkTable3
        </sourceRecord>
        <sourceNode>color</sourceNode>
      </osd:inheritance>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

The element *sourceRecord* is an expression describing how to look up the record from which the value is inherited. It is a foreign key, or a sequence of foreign keys, that starts from the current element, up to the source table.

If this element is not defined in the data model, the fields to inherit from is fetched into the same record.

The element *sourceNode* is the path of the node to inherit from in the source record.

The following conditions must be verified for this kind of inheritance:

- Element *sourceNode* is mandatory.
- The expression to the source record must be a consistent path of foreign key(s), from the current element up to the source record. This expression must only involve 1-1 relations or 0-1 relations.
- An element of the *sourceRecord* cannot be an aggregated list.
- Each element of the *sourceRecord* must be a foreign key.
- If the inherited field is also a foreign key, the *sourceRecord* property cannot refer to itself for the path to the source record of the inherited value.
- Each element of the *sourceRecord* must exist.
- The source node must belong to the source table.
- The source node must be terminal.
- The source node must be writeable.
- The source node type must be compatible with the current node type
- The source node cardinalities must be compatible with the current node.
- The source node cannot be the same as the inherited field if the fields to inherit from is fetched into the same record.

Lookup mechanism for value

The lookup mechanism for inherited fields values is the following:

1. If the value is locally defined, returns it.

It can explicitly be `null`

2. Otherwise, looks up the source record and value to inherit from, according to the properties that are defined in the data model.
3. The process is recursive, if the source node does not locally define a value, it is 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 48

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.

CHAPTER 49

Performance guidelines

Basic performance check-list

While EBX5 is designed to support large volumes of data, several common factors can lead to poor performance. Addressing the key points discussed in this section will solve usual performance bottlenecks.

Expensive programmatic extensions

The table below details the programmatic extensions that can be implemented.

Use case	Programmatic extensions that can be involved
<i>Validation</i>	<ul style="list-style-type: none"> • programmatic constraints [Constraint]^{API} • computed values [ValueFunction]^{API}
<i>Table access</i>	<ul style="list-style-type: none"> • record-level permission rules [SchemaExtensionsContext]^{API} • programmatic filters [AdaptationFilter]^{API}
<i>EBX5 content display</i>	<ul style="list-style-type: none"> • computed values [ValueFunction]^{API} • UI Components [UIBeanEditor]^{API} • node-level permission rules [SchemaExtensionsContext]^{API}
<i>Data update</i>	<ul style="list-style-type: none"> • triggers [package-summary]^{API}

For large volume of data, cumbersome algorithms will have serious effects on performance. For example, a constraint algorithm's complexity is $O(n^2)$; if size is 100, the resulting cost is proportional to 10,000

(this generally produces an immediate result); but if size is 10,000, the resulting cost will be proportional to 10,000,000.

Another source of slowness is a call to external resources. Local caching usually solves this type of problem.

If one of the specific use cases above shows poor performance, it is advised to track the problem either through code analysis or by means of a Java profiling tool.

Insufficient memory

When a table is in semantic mode (default mode), the EBX5 Java memory cache is used. It ensures a much more efficient access to data when this data is already loaded in cache. If there is not enough space for working data, swaps between the Java heap space and the underlying database can heavily degrade overall performance.

These aspects are exposed in the section [Memory Management](#) below.

Relational mode

If a table holds too many records to fit into the available memory, it is advised to use the relational mode.

For more information, see the chapter [Relational mode](#).

Directory integration

Authentication and permissions management involve the [directory](#) .

If a specific directory implementation is deployed and accesses an external directory, it can be useful to ensure local caching. More particularly, one of the most frequently called methods is

Directory.isUserInRole [Directory]^{API} .

Aggregated lists

In a data model, when an element's cardinality constraint `maxOccurs` is greater than 1 and no `osd:table` is declared on this element, it is implemented as a Java List. We call this an [aggregated list](#) as opposed to a *table*.

It is important to consider that no particular optimizations are done to access aggregated lists (for example: iterations, GUI display, etc.). Additionally and outside performance concerns, the aggregated lists are limited regarding many functionalities that are supported by tables (see [tables introduction](#) for a list of these features).

Hence *aggregated lists should be used only for small volumes of simple data (one or two dozens of records), with no advanced requirements* for their identification, lookups, permissions, etc. For larger volumes of data (or more advanced functionalities), it is recommended to use `osd:table` declarations.

Reorganization of database tables

As with any database, inserting and deleting large volumes of data may lead to fragmented data, which can deteriorate performance over time. To resolve the issue, reorganizing the impacted database tables is necessary. See [Repository Administration](#).

A specificity of EBX5 is that the creation of data spaces and snapshots adds new entries to the table `{ebx.persistence.table.prefix}HTB` (see [Repository Administration](#)). It may therefore be necessary to schedule regular reorganizations of this table for large repositories in which many data spaces are created and deleted.

Memory Management

Loading strategy

The administrator can specify the loading strategy of a data space or snapshot in its information pane. The default strategy is to load and unload the resources on demand. For those which are heavily used, a *forced load* strategy is usually recommended.

Note: the loading strategy only affects the tables in semantic mode (relational tables are never loaded into cache anyway).

The following sections give details about the existing modes.

On-demand loading and unloading

In this default mode, each resource in a data space is loaded or built only when it is needed. Moreover, the resources of the home are "softly" referenced by means of the standard Java `SoftReference` class; this implies that each resource can be unloaded *"at the discretion of the garbage collector in response to memory demand"*.

The main advantage of this default mode is the ability to free the memory when needed. As a counterpart, this implies a load/build cost when an accessed resource has not yet been loaded since server startup, or if it has been unloaded since.

Forced loading

If the *Forced loading* strategy is enabled for a data space or snapshot, its resources are loaded asynchronously on server startup. Moreover, each resource of the home is maintained into memory until server is shut down or data space is closed.

This mode is particularly recommended for long-lived data space and/or those that are heavily used, namely any data space that serves as a reference.

Forced loading and prevalidation

This strategy is similar to the *Forced loading* one, except that the content of the loaded data space or snapshot will also be validated on server startup.

Monitoring

Indications of EBX5 load activity are provided by monitoring the underlying database, and also by the ['monitoring' log category](#).

If the numbers for *cleared* and *built* objects are high for a long time, this is an indication that EBX5 is swapping.

To facilitate the analysis of logs generated by EBX5, you can use the provided OpenOffice spreadsheet worksheet by right-clicking and saving this [link](#).

Tuning the memory

The maximum size of memory allocation pool is usually specified by the Java command-line option – `Xmx`. As is the case for any intensive process, it is important that the size specified by this option does not go beyond the available physical RAM, so that the Java process does not swap to disk at operating-system level.

Tuning the garbage collector can also benefit to the overall performance. This tuning should be adapted to the use cases and to the specific Java Runtime Environment used.

Validation

The internal incremental validation framework will optimize the work needed when some updates occur. The incremental validation process runs as follows:

- A first call to a validation report performs the full validation of a data set. Note that the [loading strategy](#) can also specify a data space to be prevalidated at server startup.
- Then, data updates will transparently and asynchronously maintain the validation report, in so far as the updated nodes specify explicit dependencies. Note: standard and static facets, foreign key constraints, dynamics facets, selection nodes specify explicit dependencies.
- If a mass-update is executed or if there are too many validation messages, the incremental validation process is stopped. Next call to the validation report will hence trigger a full validation.
- Also, if a transaction is cancelled, the validation state of the updated adaptation instances is reset. Next call to the validation report will trigger a full validation as well.

However, there is an incompressible part that is systematically revalidated (even if no updates have occurred since last validation): these are nodes with *unknown dependencies*. A node has unknown dependencies if:

- it possesses a **programmatic constraint** `[Constraint]API` in default *unknown dependencies* mode;
- it declares a **computed value** `[ValueFunction]API`;
- it declares a dynamic facet that depends on a node that is itself a **computed value** `[ValueFunction]API`;

Consequently, on large tables (beyond the order of 10^5), it is recommended to avoid nodes with unknown dependencies (or at least to minimize the number of such nodes). For programmatic constraints, the developer is able to specify two alternative modes that drastically reduce incremental validation cost: *local dependency* mode and *explicit dependencies*. For more information, see **Dependencies and Validation** [DependenciesDefinitionContext]^{API} section.

Note: it is possible for a user granted with an Administrator role to manually reset the validation report of an adaptation. This option is available from the validation report section in EBX5.

Massive updates

Massive updates can involve several hundreds of thousands of insertions, modifications or deletions. Those updates are normally not frequent (usually initial data imports), or performed in a non-interactive way (usually nightly batches); hence performance is less critical than for frequent and interactive operations. However, like classic batch processing, it has some specific issues.

Transaction boundaries

It is generally not advised to use a single transaction when the number of atomic updates in the transaction is beyond the order of 10^4 .

The main reason is that large transactions require a lot of resources (more particularly memory) from EBX5 and from the underlying database.

For reducing transactions' size, it is possible to:

- specify the property [ebx.manager.import.commit.threshold](#), however, this property is used only for archive imports done interactively, in the context of the EBX5 user interface;
- explicitly **specify commit threshold** [ProcedureContext]^{API} inside the batch procedure;
- structurally limit the transaction scope by implementing **Procedure** [Procedure]^{API} onto a part of the task and executing it as many times as needed.

On the other hand, specifying a very small transaction size will be also hinder performance because of the persistent tasks done for each commit.

Note: If intermediate commits are a problem because transactional atomicity is no longer guaranteed, it is recommended to execute the massive update inside a dedicated data space. This data space will be created just before the massive update. If update does not complete successfully, the data space must just be closed; if it succeeds, the data space can be merged safely.

Triggers

If needed, triggers can be deactivated by means of method **setTriggerActivation** [ProcedureContext]^{API}.

Access to tables

Functionalities

Tables are commonly accessed through EBX5 and also through the **Request API** [Request]^{API} and data services. This access involves a unique set of functions, including a *dynamic resolution* process, that we detail for a better understanding of their performance implications.

- *Inheritance*: Inheritance in adaptations tree implies to take into account records and values that are defined by the parent instance, through a recursive process. Also in a root instance, a record can inherit some of its values from the data model default values (`xs:default` attribute).
- *Value computation*: A node declared as `osd:function` is always **computed** [ValueFunction]^{API} on the fly, when the value is accessed.
- *Filtering*: A [XPath predicate](#), a **programmatic filter** [AdaptationFilter]^{API}, or a record-level **permission rule** [SchemaExtensionsContext]^{API} imply a selection on records.
- *Sort*: Last but not least, a sort on the result can be made.

Semantic mode: Architecture and design

In order to improve the speed of operations on tables, indexes are managed by the EBX5 engine.

EBX5 advanced features such as advanced life-cycle (snapshots and data spaces), data set inheritance and flexible XML Schema modeling, have led to a particular design on indexing mechanisms. This design can be summarized as follows:

- *Indexes* maintain an in-memory data structure on a full table, as it is defined in a data space or in a snapshot; however, it does not take into account data set inheritance.
- *Final access* to tables performs a *dynamic resolution* based on indexes.

An index is not persisted, and building it requires loading all of the table blocks from the database. This tradeoff is still beneficial, since the index can be retained in memory for longer than the corresponding table blocs.

Semantic mode: Performance considerations

The request optimizer favors the use of indexes when computing a request result. Important note: only XPath filters are considered for optimization on index.

The impacts on performance are the following, if indexes are already built:

1. If the request implies no filter, no programmatic rule, and no sort, accessing its first few rows (these fetched by a paged view) is almost immediate.
2. If the request can be resolved with no extra sort step (this is the case if it has no sort criteria, or if its sort criteria relate to that of the index used for computing the request), accessing the first few rows of a table should be quick. More precisely, it depends on the cost of the specific filtering algorithm that is executed when fetching at least 2,000 records.
3. Both cases above guarantee an access time that is independent from the size of the table, and provides a view sorted on the index used. If an extra sort is required, then the first access time depends on the table size according to a $N \log(N)$ function (where N is the number of records in the resolved view).

If indexes are not yet built, or have been unloaded, additional time will be needed: the build time is $O(N \log(N))$.

Access to the table data blocks is required when the request cannot be computed against a sole index (whether for resolving a rule, filter or sort), as well as for building the index; if the table blocs are not present in memory, additional time will be needed to fetch them from the database.

It is possible to get information through the [monitoring](#) and request log category.

Semantic mode: Other operations on tables

The creation of new records (or *records insert*) depends on the primary key index. Hence, a creation becomes almost immediate if this index is loaded.

Semantic mode: Conclusion about tables

Faster access to tables is ensured if indexes are ready and maintained in memory cache. As mentioned above, it is important for the Java Virtual Machine to have enough space allocated, so that it does not release indexes too quickly.

Relational mode: Performance considerations

In order to improve the speed of operations on tables, indexes may be declared on a table at data-model level. This will trigger the creation of an index on the corresponding table, in the database.

When computing a request result, the EBX5 engine delegates to the RDBMS:

- handling of all the request sort criteria: they are translated to an ORDER BY clause;
- whenever possible, handling of the request filters (they are translated to a WHERE clause). Important note: only XPath filters are considered for optimization on index. If the request includes non-optimizable filters, table rows will be fetched from the database, then filtered in Java memory by EBX5, until the requested page size is reached. This is not as efficient as filtering on the database side (especially regarding I/O).

It is possible to get information on the transmitted SQL request, through the *persistence* log category.

When designing an index aiming at improving the performance of a given request, the same rules apply as for traditional database index design.

Persistence

CHAPTER 50

Overview of persistence

This chapter describes how master data, history, and replicated tables are persisted. A given table can employ any combination of master data persistence mode, historization, and replication.

While all persisted information in EBX5 is ultimately stored as relational tables in the underlying database, whether it is in a form that is accessible outside of EBX5 depends on if it is in mapped mode.

Note

The term [mapped mode](#) refers to any tables that are stored as-is, and thus whose contents can be accessed directly in the database.

Persistence of managed master data

Data that is modeled in and governed by the EBX5 repository can be persisted in one of two modes, semantic (default) or relational, as specified in its underlying data model. Distinct tables defined in either mode can co-exist and collaborate within the same EBX5 repository.

Semantic mode

The semantic mode offers all EBX5 advanced features of master data management, in particular, data spaces, data set inheritance, and inherited fields.

Semantic mode is the default mode for persisting the data governed by the EBX5 repository. Data models are in semantic mode unless [relational mode](#) is explicitly specified.

Internally, the master data managed in semantic mode is represented as standard XML, which complies with the XML Schema Document of its data model. The XML representation is additionally compressed and segmented for storage into generic relational database tables. This mode provides efficient data storage and access, including for:

- Data spaces: no data is duplicated when creating a child data space, and
- Inheritance: no data is duplicated when creating an inherited instance.

Semantic mode also makes it possible to maintain an unlimited number of data sets for each data model, organized into an unlimited number of data spaces or snapshots. This can be done with no impact on the database schema.

As this mode only uses common, generic internal tables, modifications to the structure of the data model also never impact the database schema. Data model evolutions only impact the content of the generic database tables.

See also:

- [data spaces](#)
- [data set inheritance](#)
- [inherited fields](#)

Relational mode

Relational mode, which is a mapped mode, persists master data directly into the database. The primary function of relational mode is to be able to benefit from the performance and scalability capabilities of the underlying relational database. However, relational mode does not support the advanced governance features offered by semantic mode.

For some cases where the management advantages of semantic mode are not necessary, such as "current time" tables, or tables that are regularly updated by external systems, the performance gains offered by relational mode may be more valuable.

Generally, when a data set is in relational mode, every table in this data set has a corresponding table in the database and every field of its data model is mapped to a relational table column.

See also: [Relational mode](#)

Comparison of semantic mode and relational mode

This table summarizes the differences between the two persistence modes:

	Semantic mode	Relational mode
Data spaces	Yes	No
Data set inheritance	Yes	No
Inherited fields	Yes	No
Data model	All features are supported.	Some restrictions, see Data model restrictions for tables in relational mode .
Direct SQL reads	No	Yes, see SQL reads .
Direct SQL writes	No	Yes, but only under precise conditions, see SQL writes .
Data validation	Yes, enables tolerant mode.	Yes, some constraints become blocking, see Validation .
Transactions	See Concurrency and isolation levels .	See Concurrency and isolation levels .

Historization of tables

Master data tables can activate historization in order to track modifications to their data, regardless of whether they are persisted in semantic or relational mode, and whether they are replicated.

The history itself is in mapped mode, meaning that it can potentially be consulted directly in the underlying database.

See also: [History](#)

Replication and SQL access to master data

Replication enables direct SQL access to tables of master data, by making a copy of data in the repository to relational table replicas in the database. Replication can be enabled any table regardless of whether it is persisted in semantic or relational mode, and whether it has history activated.

The replica tables are persisted in mapped mode, as their primary purpose is to make master data accessible to direct queries outside of EBX5.

See also: [Replication](#)

Mapped mode

Overview of mapped mode

Mapped mode refers to cases where tables are persisted in the underlying relational database in a format that allows their data to be accessed directly, outside of EBX5. Master data modeled in relational mode, history, and replica tables are all examples of tables in mapped mode.

All cases of mapped mode involve automatic alterations of the database schema (the database tables, indexes, etc.) when necessary, by automatically executing required DDL statements in the background. Such procedures are always triggered at data model compilation time and the data model compilation report notifies of any resulting errors.

Another general consideration regarding mapped modes is that, in most cases, when a data model entity is removed, its corresponding database object is not deleted immediately. Instead, it is marked as disabled, which leaves the possibility of later re-enabling the object. It also avoids executing large operations at runtime. In order to definitively drop the object and its associated data and resources from the database, it must be marked for purge. The removal then takes place during the next global purge.

See also:

- [Purging database resources of mapped tables](#)
- [Data model evolutions](#)

Data model restrictions due to mapped mode

Due to the nature of persisting directly in the underlying database, some restrictions apply to all tables stored in mapped mode:

- Unlimited-length strings: All string fields, except foreign keys, of type `xs:string`, its derived types, and `xs:anyURI` must define a 'maxLength' or 'length' facet. Since a foreign key field is composed of the final primary key field(s) of its target table(s), this facet requirement applies to each of those final primary key fields instead of the foreign key field itself. Additionally, limitations of the underlying database concerning the maximum length of its character types apply, such as `VARCHAR` and `NVARCHAR2`.
- The attribute `type="osd:password"` is ignored.

More generally, tables in mapped mode are subject to any limitations of the underlying RDBMS. For example, the maximum number of columns in a table applies (1000 for Oracle 11g R2, 1600 for PostgreSQL).

Data model evolutions may also be constrained by the underlying RDBMS, depending on the existing data model.

See also: [Data model evolutions](#)

CHAPTER 51

Relational mode

Relational mode is a [mapped mode](#), whose primary function is to be able to benefit from the performance and scalability capabilities of the underlying relational database. It persists master data tables directly into the database. Relational mode does not support the advanced governance features offered by semantic mode.

Enabling relational mode for a data model

The data model declares that it is in relational mode. Due to the necessary restrictions of relational mode, such as not having data spaces or snapshots, a specific relational data space must be provided, to which the data model will be published. Relational data spaces do not allow creating sub-data spaces or snapshots.

Example of a relational mode declaration:

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <osd:relationalMode>
        <dataSpace>aDataSpaceKey</dataSpace>
        <dataSet>aDataSetReference</dataSet>
        <tablesPrefix>aPrefixForTablesInRDBMS</tablesPrefix>
      </osd:relationalMode>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

with the elements:

Element	Description	Required
<code>dataSpace</code>	Specifies the data space where the data model must be published. This data space must itself be in relational mode. No data space or snapshot can be created from a data space declared in such a mode.	Yes
<code>dataSet</code>	Specifies the data set where the data model must be published.	Yes
<code>tablesPrefix</code>	Specifies the common prefix used for naming the generated tables in the database.	Yes

Validation

This section details the impact of relational mode on data validation.

Structural constraints

Some EBX5 data model constraints will generate a "structural constraint" on the underlying RDBMS schema for relational mode and also if [table history is activated](#). This concerns the following facets:

- facets `xs:maxLength` and `xs:length` on string elements;
- facets `xs:totalDigits` and `xs:fractionDigits` on `xs:decimal` elements.

Databases do not support as tolerant a validation mode as EBX5. Hence, the above constraints become *blocking constraints*. A blocking constraint means that updates are rejected if they do not comply. Additionally, such constraints are no longer checked during validation process, except for foreign key constraints under some circumstances (see [Foreign key blocking mode](#)). When a transaction does not comply with a blocking constraint, it is cancelled and a **ConstraintViolationException** `[ConstraintViolationException]API` is thrown.

Foreign key blocking mode

The foreign key constraints defined on a table in relational mode and referencing a table in relational mode are also in blocking mode, so as to reduce validation time. For this constraint, blocking mode implies that attempting the following actions will result in a **ConstraintViolationException** `[ConstraintViolationException]API`:

- Deleting a record referenced by a foreign key constraint,
- Deleting an instance referenced by a foreign key constraint,
- Closing a data space referenced by a foreign key constraint.

However, in order to ensure the integrity of foreign key constraints after direct SQL writes that bypass the EBX5 governance framework, the foreign key constraints will be validated on the following cases:

- On the first explicit validation through the user interface or API,
- On the first explicit validation through the user interface or API after refreshing the schema,
- On the first explicit validation through the user interface or API after resetting the validation report of a data set in the user interface.

Constraints on the whole table

Programmatic **constraints** `[Constraint]API` are checked on each record of the table at validation time. If the table defines millions of records, this becomes a performance issue. It is then recommended to define a **table-level constraint** `[ConstraintOnTable]API`.

In the case where it is not possible to define such a table-level constraint, it is recommended to at least define a **local or explicit dependency** `[DependenciesDefinitionContext]API`, so as to reduce the cost of incremental validation.

See also: ***ConstraintOnTable interface*** `[ConstraintOnTable]API`

SQL access to data in relational mode

This section describes how to directly access the data in relational mode, through SQL.

See also: [SQL access to history](#)

Finding the table in the database

For every EBX5 table in relational mode, a corresponding table is generated in the RDBMS. Using the EBX5 user interface, you can find the name of this database table by clicking on the documentation pane of the table.

SQL reads

Direct SQL reads are possible in well-managed, preferably short-lived transactions. However, for such accesses, EBX5 permissions are not taken into account. As a result, applications given allowed to perform reads must be trusted through other authentication processes and permissions.

SQL writes

Direct SQL writes bypass the governance framework of EBX5. Therefore, they must be used with **extreme caution**. They may risk causing the following situations:

- failure to historize EBX5 tables;
- failure to execute EBX5 triggers;
- failure to verify EBX5 permissions and constraints;
- modifications missed by the incremental validation process;
- losing visibility on EBX5 semantic tables, which might be referenced by foreign keys.

Consequently, direct SQL writes are to be performed *if, and only if, all the following conditions are verified*:

- The written tables are not historized and have no EBX5 triggers.
- The application performing the writes can be fully trusted with the associated permissions, to ensure the integrity of data. Specifically, the integrity of foreign keys (`osd:tableRef`) must be preserved at all times. See [Foreign key blocking mode](#) for more information.
- The application server running EBX5 is shut down *whenever writes are performed*. This is to ensure that incremental validation does not become out-of-date, which would typically occur in a batch context.

Limitations of the relational mode

The relational mode feature is fully functional, but has some known limitations, which are listed below. If using relational mode, it is strongly recommended to read these limitations carefully and to contact Orchestra Networks support in case of questions.

See [Supported databases](#) for the databases on which relational mode is supported.

Data model restrictions for tables in relational mode

Some restrictions apply to data models in relational mode:

- [Data model restrictions due to mapped mode](#).
- [Aggregated lists](#) in tables.
- User-defined attributes on relational tables result in data model compilation errors.
- [Data set inheritance](#).
- [Inherited fields](#).
- Programmatic constraints, since the computation cost of validation would be too high. However, **constraints on tables** [`ConstraintOnTable`]^{API} remain available.

Schema evolutions may also be constrained by the underlying RDBMS, depending on the data already contained in the concerned tables.

See also: [Data model evolutions](#)

Other limitations

- Relational mode supports Unicode data. However, for Oracle, data loss may occur if the *database character set* is not 'UTF8' or 'AL32UTF8'. A workaround is to set the Java system property `oracle.jdbc.defaultNChar=true`.
- From a data space containing data sets in relational mode, it is not possible to create child data spaces and snapshots.
- For D3, it is not possible to broadcast a data space defined in relational mode.
- For very large volumes of data, the validation will show poor performance if the relational table declares any of these features: `osd:function`, `osd:select`, `osd:uiFilter`, `osd:tableRef/filter`. Additionally, a sort cannot be applied on a `osd:function` column.
- The specific distinction of `null` encounters some limitations: terminal complex types are supported, but at record-level, they cannot be globally set to `null`; on simple `xs:string` elements, Oracle database does not support the distinction between empty strings and null values (see section [Empty string management](#)).
- It is not possible to set the `AdaptationValue.INHERIT_VALUE` to a node belonging to a data model in relational mode.

CHAPTER 52

History

Overview

History is a feature allowing to track all data modifications on a table (records creation, update and deletion).

Some RDBMS are not supported yet (see [Limitations of the historized mode](#) for more details). [XML audit trail](#) can then be used as an alternative. XML audit trail is activated by default; it can be safely deactivated if your RDBMS is supported.

See also:

- [History](#)
- [Relational mode](#)
- [Replication](#)
- [Data model evolutions](#)

Configuring history

In order to activate historization for a table, a history profile has to be set for the table in the data model. This section describes history profiles and the way they are associated with tables.

Configuring history in the repository

A history profile specifies when the historization is to be created and the level of guarantee requested. In order to edit history profiles, select **Administration > History and logs**.

A history profile is identified by a name and defines the following information:

- An internationalized label.
- A list of data spaces (branches) for which history is activated. It is possible to specify whether direct children and/or all descendants should also be concerned.
- The attribute *Transaction fails if historization is not available*. It allows you to specify whether a transaction must fail when historization is requested but not available. Usually, it will be disabled in a development environment, but enabled in a production environment.

Some profiles are already created when installing the repository. These profiles can neither be deleted nor modified.

Profile Id	Description
ebx-referenceBranch-strict	This profile is activated only on the reference data space. Its attribute <i>Transaction fails if historization is not available</i> is activated.
ebx-referenceBranch-lax	This profile is activated only on the reference data space. Its attribute <i>Transaction fails if historization is not available</i> is deactivated.
ebx-allBranches-strict	This profile is activated on all data spaces. Its attribute <i>Transaction fails if historization is not available</i> is activated.
ebx-allBranches-lax	This profile is activated on all data spaces. Its attribute <i>Transaction fails if historization is not available</i> is deactivated.
ebx-instanceHeaders	This profile historizes data set headers. However, this profile will only be setup in a future version, given that the internal data model only defines data set nodes.
ebx-xxx	This profile historizes internal data space xxx, for example <i>ebx-dataSpace</i> .

Configuring history in the data model

Activating table history

History can be activated on a table either through the data model assistant, or by editing the underlying data model.

To activate history by editing the data model, a history profile should be declared on the table using the `historyProfile` element.

```
<osd:table>
  <primaryKeys>/key</primaryKeys>
  <historyProfile>historyProfileForProducts</historyProfile>
</osd:table>
```

The data model assistant allows you to view the historization profiles defined in the repository.

Historization must be activated for each table separately. See [model design](#) documentation for more details.

Disabling history on a specific field or group

For a historized table, the default behavior is to historize all its supported elements (see [Impacts and limitations of the historized mode](#)).

It is possible to disable history for a given field or group, either through the data model assistant, or by editing the underlying data model.

To disable the history of a field or group by editing the data model, use the element `osd:history` with the attribute `disable="true"`.

```
<xs:element name="longDescription" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:history disable="true" />
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

To disable the history of a field or group through the data model assistant, use the `History` property in the `Advanced properties` of the element.

When this property is defined on a group, history is disabled recursively for all its descendents. Once a group disables history, it is not possible to specifically re-enable history on a descendant.

Note

If the table containing the field or group is not historized, this property will not have any effect.

It is not possible to disable history for primary key fields.

Integrity

If problems are detected at data model compilation, warning messages or error messages will be added to the validation report associated with this data model. Furthermore, if any error is detected, each associated instance (data set) will be inaccessible. The most common error cases are the following:

- A table references a profile that is not defined in the repository.
- A history profile that is referenced in the data model mentions a non-defined or closed data space in the current repository.

Note: Deploying a data model on a repository that does not have the expected profiles requires the administrator to add them.

History views and permissions

Table history view

When the history has been activated on a table in the data model, it is possible to access the history view from various locations in the user interface: record, selection of records, table and data set.

The next section explains how permissions are resolved.

For more information, see [table history view](#) section. To access the table history view from Java, the method **AdaptationTable.getHistory()** [AdaptationTable]^{API} must be invoked.

Permissions for table history

Data permissions are also applied to data history. History permissions are resolved automatically as the most restricted permission between data permissions and *read-only* access right.

This is true for user-defined permission rules and also for programmatic permission rules.

For defining a programmatic rule, there is sometimes a need to distinguish between the functional data set context and the history view context. This can also be required, for instance if the permission rule accesses some data set's fields that are not historized (in such a case, they are not present in the history structure). The methods **Adaptation.isHistory()** [Adaptation]^{API} and **AdaptationTable.getHistory()** [AdaptationTable]^{API} can then be used in the programmatic rule in order to implement the specific behavior for the history.

Transaction history views

The transaction history view gives access to the executed transactions, independently of a table, a data set or a data model, directly from the user interface.

To see the 'Transaction history' table, navigate to the Administration area and select 'History and logs' using the down arrow menu in the navigation pane. Transaction history can also be accessed from the Data Spaces area by selecting a historized data space and using the **Actions** menu in the workspace.

For more information, see [transaction history view](#).

SQL access to history

This section describes how to directly access the history data by means of SQL.

See also: [SQL access to data in relational mode](#)

Access restrictions

The database tables must be accessed only in read-only mode. It is up to the database administrator to forbid write access except for the database user used by EBX5, as specified in the section [Rules for the access to the database and user privileges](#).

Relational schema overview

Here is a description of the history tables in the database.

The database schema contains (see also the diagram in the next section):

Common and generic tables	<p>The main table is <code>HV_TX</code>; each record of this table represents a transaction. Only transactions that involve at least one historized table are recorded.</p> <p>These common tables are all prefixed by "HV".</p>
Specific generated tables	<p>For each historized table, a specific history table is generated. This table contains the history of the data modifications on the table.</p> <p>In the EBX5 user interface, the name of this table in database can be obtained by clicking on the table documentation pane (advanced mode). All the specific history tables are prefixed with "HG".</p>

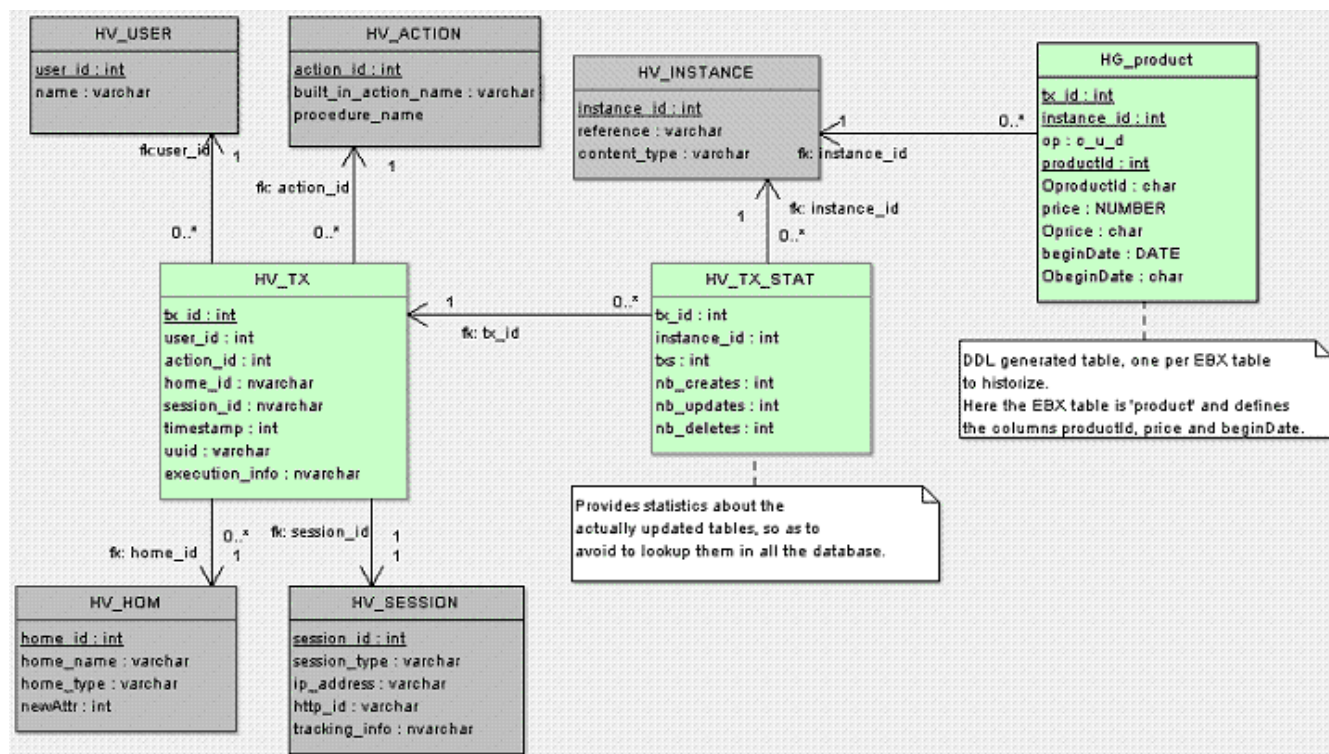
Example of a generated history table

In the following example, we are historizing a table called `product`. Let us assume this table declares three fields in EBX5 data model:

Product

- `productId`: int
- `price`: int
- `beginDate`: Date

The diagram below shows the resulting relational schema:



Activating history on this table generates the HG_product table shown in the history schema structure above. Here is the description of its different fields:

- tx_id: transaction ID.
- instance: instance ID.
- op: operation type - C (create), U (update) or D (delete).
- productid: productid field value.
- Oproductid: operation field for productid, see next section.
- price: price field value.
- Oprice: operation field for price, see next section.
- beginDate: date field value.
- ObeginDate: operation field for beginDate, see next section.

Operation field values

For each functional field, an additional operation field is defined, composed by the field name prefixed by the character O. This field specifies whether the functional field has been modified. It is set to one of the following values:

- `null`: if the functional field value has not been modified (and its value is not INHERIT).
- `M`: if the functional field value has been modified (not to INHERIT).
- `D`: if record has been deleted.

If [inheritance](#) is enabled, the operation field can have three additional values:

- `T`: if the functional field value has not been modified and its value is INHERIT.
- `I`: if the functional field value has been set to INHERIT.
- `O`: if the record has been set to OCCULTING mode.

Impacts and limitations of the historized mode

The history feature has some impacts and known limitations, which are listed in this section. If using historized mode, it is strongly recommended to read these limitations carefully and to contact Orchestra Networks support in case of questions.

See [Supported databases](#) for the databases on which historized mode is supported.

Validation

Some EBX5 data model constraints become blocking constraints when table history is activated. For more information, see the section [Structural constraints](#).

Data model restrictions for historized tables

Some restrictions apply to data models containing historized tables:

- [Data model restrictions due to mapped mode](#).
- Limitations exist for two types of aggregated lists: embedded aggregated lists, and aggregated lists under a terminal complex type. Data models that contain such aggregated lists can be used, however these list fields will be ignored (not historized).
- Computed values are ignored.
- User-defined attributes on historized tables result in data model compilation errors.

Data model evolutions may also be constrained by the underlying RDBMS, depending on the data already contained in the concerned tables.

See also: [Data model evolutions](#)

Other limitations

- No data copy is performed when a table with existing data is activated for history.
- Global operations on data sets are not historized (create an instance and remove an instance), even if they declare an historized table.
- Default labels referencing a computed field value are not supported for historized tables. The workaround is to implement the `UILabelRenderer` interface and adapt the label computation for history.
- D3: the history can be enabled in the delivery data space of a master node, but in the delivery data space of the slave nodes, the historization features are always disabled.

CHAPTER 53

Replication

Overview

Data stored in the EBX5 repository can be mirrored to dedicated relational tables to enable direct access to the data by SQL requests and views.

Like history and relational mode, this data replication is transparent to end-users and client applications. Certain actions trigger automatic changes to the replica in the database:

- Activating replication at the model-level updates the database schema by automatically executing the necessary DDL statements.
- Data model evolutions that impact replicated tables, such as creating a new column, also automatically update the database schema using DDL statements.
- When using the 'on commit' refresh mode: updating data in the EBX5 repository triggers the associated inserts, updates, and deletions on the replica database tables.

See also:

- [Relational mode](#)
- [History](#)
- [Data model evolutions](#)
- [Repository administration](#)

Configuring replication

Enabling replication

To define a *replication unit* on a data model, use the element `osd:replication` under the elements `annotation/appinfo`. Each replication unit specifies tables in a single data set in a specific data space.

The nested elements are as follows:

Element	Description	Required
name	Name of the replication unit. This name identifies a replication unit in the current data model. It must be unique in this data model.	Yes
dataSpace	Specifies the data space relevant to this replication unit. It cannot be a snapshot or a relational data space.	Yes
dataSet	Specifies the data set relevant to this replication unit.	Yes
refresh	Specifies the data synchronization policy. The possible policies are: <ul style="list-style-type: none"> onCommit: The replica table content in the database is always up to date with respect to its source table. Every transaction that updates the EBX5 source table triggers the corresponding insert, update, and delete statements on the replica table. onDemand: The replication of specified tables is only done when an explicit refresh operation is performed. See Requesting an 'onDemand' replication refresh. 	Yes
table/path	Specifies the path of the table in the current data model that is to be replicated to the database.	Yes
table/nameInDatabase	Specifies the name of the table in the database to which the data will be replicated. This name must be unique amongst all replications units.	Yes

For example:

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo>
      <osd:replication>
        <name>ProductRef</name>
        <dataSpace>ProductReference</dataSpace>
        <dataSet>productCatalog</dataSet>
        <refresh>onCommit</refresh>
        <table>
          <path>/root/domain1/tableA</path>
          <nameInDatabase>PRODUCT_REF_A</nameInDatabase>
        </table>
        <table>
          <path>/root/domain1/tableB</path>
          <nameInDatabase>PRODUCT_REF_B</nameInDatabase>
        </table>
      </osd:replication>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

Notes:

- See [Data model restrictions for replicated tables](#)
- If, at data model compilation, the specified data set and/or data space does not exist in the current repository, a warning is reported, but the replica table is created in the database. Once the specified data space and data set are created, the replication becomes active.
- At data model compilation, if a table replication is removed, or if some of the above properties has changed, the replica table is dropped from the database, and then recreated with the new definition if needed.

Disabling replication on a specific field or group

For a replicated table, the default behavior is to replicate all its supported elements (see [Data model restrictions for replicated tables](#)).

It is possible to disable replication for a specific field or group, either through the data model assistant, or by editing the underlying data model.

To disable the replication of a field or group by editing the data model, use the element `osd:replication` with the attribute `disable="true"`.

```
<xs:element name="longDescription" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:replication disable="true" />
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

To disable the replication of a field or group through the data model assistant, use the `Replication` property in the `Advanced properties` of the element.

When this property is defined on a group, replication is disabled recursively for all its descendents. Once a group disables replication, it is not possible to specifically re-enable replication on a descendant.

Note

If the table containing the field or group is not replicated, this property will not have any effect.

It is not possible to disable replication for primary key fields.

Accessing a replica table using SQL

This section describes how to directly access a replica table using SQL.

See also: [SQL access to history](#)

Finding the replica table in the database

For every replicated EBX5 table, a corresponding table is generated in the RDBMS. Using the EBX5 user interface, you can find the name of this database table by clicking on the documentation pane of the table.

Access restrictions

The replica database tables must only be directly accessed in read-only mode. It is the responsibility of the database administrator to block write-access to all database users except the one that EBX5 uses.

See also: [Rules for the access to the database and user privileges](#)

SQL reads

Direct SQL reads are possible in well-managed, preferably short-lived transactions. However, for such accesses, EBX5 permissions are not taken into account. As a result, applications given the privilege to perform reads must be trusted through other authentication processes and permissions.

Requesting an 'onDemand' replication refresh

The 'onDemand' refresh policy requires an explicit request to refresh the replicated table data.

There are several ways to request a replication refresh:

- **User interface:** In the data set actions menu, use the action 'Refresh replicas' under the group 'Replication' to launch the replication refresh wizard.
- **Data services:** Use the replication refresh data services operation. See [Replication refresh](#) for data services for more information.
- **Java API:** Call the `performRefresh` `[ReplicationUnit]API` methods in the `ReplicationUnit` API to launch a refresh of the replication unit.

Impact and limitations of replication

The replication feature has some known limitations and side-effects, which are listed below. If using replication, it is strongly recommended to read this section carefully and to contact Orchestra Networks support in case of questions.

See [Supported databases](#) for the databases for which replication is supported.

Validation

Some EBX5 data model constraints become blocking constraints when replication is enabled. For more information, see [Structural constraints](#).

Data model restrictions for replicated tables

Some restrictions apply to data models containing tables that are replicated:

- [Data model restrictions due to mapped mode](#).
- Data set inheritance is not supported for the 'onCommit' refresh policy if the specified data set is not a root data set or has not yet been created. See [data set inheritance](#) for more information.
- Field inheritance is also only supported for the 'onDemand' refresh policy. This means that, at data model compilation, an error is reported if the refresh mode is 'onCommit' and the table to be replicated has an inherited field. See [inherited fields](#) for more information.
- User-defined attributes are only allowed when using the 'onDemand' replication refresh policy.
- Computed values are ignored.
- Aggregated lists are ignored in this version; they will be supported in a future version.
- It is currently not supported to include the same table in several replication units. In a future version, it will be possible to include a table in at most two units, one unit having the refresh policy 'onDemand' and the other having 'onCommit'.

Data model evolutions may also be constrained by the underlying RDBMS, depending on the data already contained in the concerned tables.

See also: [Data model evolutions](#)

Database configuration

The refresh operation is optimized to transmit only the rows of the source table that have been modified (with respect to creation and deletion) since the last refresh. However, depending on the volume of data exchanged, this can be an intensive operation, requiring large transactions. In particular, the first refresh operation can concern a large number of rows. It is necessary for the database to be configured properly to allow such transactions to run under optimal conditions.

For instance, with Oracle:

- It is mandatory for the whole replica table to fit into the 'UNDO' tablespace.
- It is recommended to provide enough space in the buffer cache to allow those transactions to run with minimal disk access.
- It is recommended to provision 'REDO' log groups big enough to avoid those transactions to wait for the 'db_writer' process.

Other limitations

- Replication supports Unicode data. However, for Oracle, data loss may occur if the *database character set* is not 'UTF8' or 'AL32UTF8'. A workaround is to set the Java system property `oracle.jdbc.defaultNChar=true`.
- The distinction of `null` values encounters certain limitations. Terminal complex types are supported, however at record-level, they cannot be globally set to `null`. On simple `xs:string` elements, Oracle does not support the distinction between empty strings and `null` values. See [Empty string management](#) for more information.
- For inheritance, a replica record field cannot hold the "inherit value" flag (`AdaptationValue.INHERIT_VALUE`). It only holds the inherited value in such cases. More generally, it is not possible to distinguish inheriting state from overwriting state.

CHAPTER 54

Data model evolutions

This chapter describes the modifications that are possible on data models, as well as potential limitations. Most limitations apply to the mapped modes.

Note

Certain types of data model evolutions cannot be performed directly in the user interface, and thus the data model must be exported, modified in XSD format, then re-imported. For changes to a data model that impact its configuration, not just its structure, the XSD must be imported into EBX5 from a module. Otherwise, the configuration modifications are not taken into account.

See also: [Mapped mode](#)

Types of permitted evolutions

This section describes the possible modifications to data models after their creation.

Model-level evolutions

The following modifications can be made to existing data models:

- A data model in semantic mode can be declared to be in relational mode. Data should be manually migrated using an XML or archive export, then a re-import.
- Relational mode can be disabled on the data model. Data should be manually migrated using an XML or archive export, then a re-import.
- Replication units can be added or removed from the data model.
- The data model can be deleted. If it is in relational mode or if it declares replication units, this change marks the associated mapped tables as disabled and marks them for purge. See [Purging database resources of mapped tables](#) for the actual removal of associated database objects.

Table-level evolutions

The following modifications can be made to a data model at the table-level:

- A new table can be added. Upon creation, the table can also declare one or more mapped modes.
- An existing table, which may declare one or more mapped modes, can be deleted. If it declares at least one mapped mode, this change marks the mapped table as disabled. See [Purging database resources of mapped tables](#) for the actual removal of associated database objects.
- An existing table in semantic mode can be declared to be in relational mode. Data should be manually migrated using an XML or archive export, then a re-import.
- History can be enabled or disabled on a table. History will not take into account the operations performed while it is disabled.
- A table can be renamed or moved. Data should be manually migrated using an XML or archive export, then a re-import, because these changes are considered to be a combination of deletion and creation.

Field-level evolutions

The following modifications can be made to a data model at the field-level:

- A new field can be added.
- An existing field can be deleted. In mapped mode, the field is marked as disabled. In semantic mode, the data of the deleted field is removed from each record upon its next update.
- A field can be renamed or moved. Data should be manually migrated using an XML or archive export, then a re-import, because these changes are considered to be a combination of deletion and creation.
- The facets of a field can be modified, except for the facets listed under [Limitations/restrictions](#).
- For a specific field within a table that activates history or replication, the mode can be disabled using the attribute `disable="true"`. The field in the corresponding mapped table is automatically removed. See [Disabling history on a specific field or group](#) and [Disabling replication on a specific field or group](#).

Index-level evolutions

- An index can be added or renamed.
- An index can be modified, by changing or reordering its fields. In mapped mode, the existing index is marked as disabled and a new one is created.
- An index can be deleted. In mapped mode, a deleted index is marked as disabled.

Limitations/restrictions

Note

All limitations listed in this section that affect mapped mode can be worked around by purging the mapped table database resources. For the procedure to purge mapped table database resources, see [Purging database resources of mapped tables](#).

Limitations related to primary key evolutions

When a primary key definition is modified:

- In semantic mode, the existing records are only loaded into the cache if they:
- Respect the uniqueness constraint of the primary key,
- Comply with the new structure of the primary key.
- In mapped mode, the underlying RDBMS only accepts a primary key modification if all table records respect its uniqueness and non-nullity constraints.

Limitations related to foreign key evolutions

- When the declaration of a facet `osd:tableRef` is added or modified, or the primary key of the target table of a facet `osd:tableRef` is modified:
- In semantic mode, the existing values for this field are only loaded into the cache if they comply with the new structure of the target primary key.
- In mapped mode, the foreign key is mapped against columns that correspond to the target primary key. A type or path change for any of these columns is equivalent to the combination of a field deletion and creation. Thus, the corresponding data should be migrated manually.

Limitations related to field-level evolutions

- In mapped mode, when a `maxLength`, `length`, `totalDigits` or `fractionDigits` facet is modified:

Whether or not this modification is accepted depends on the underlying DBMS, as well as the field type and the contents of the table.

For example, Oracle will accept changing a `VARCHAR(20)` to a `VARCHAR(50)`, but will only change a `VARCHAR(50)` to a `VARCHAR(20)` if the table does not contain any values over 20 characters long.

PostgreSQL has the same limitations, but additionally, any modification of a field type (including modifications of its length) will invalidate all related prepared statements, and require restarting the application server.

- When a cardinality of an element is modified:
- In semantic mode, this change is supported. However, two cases are distinguished:
- When changing a single element to an aggregated list, the previous single value is preserved and added to the new aggregated list.
- When changing an aggregated list to a single element, only the last value of the aggregated list is preserved in the single element. The other values are lost.
- In relational mode, aggregated lists are not supported. An error message is added to the compilation report of the data model if an element is changed to an aggregated list.
- In historized mode, when changing a single element to an aggregated list, the modification is taken into account, but the previous single value is lost.

Administration Guide

Installation & configuration

CHAPTER 55

Supported environments

Browsing environment

Supported web browsers

EBX5 web tool supports the following browsers:

- Microsoft Internet Explorer 8
Compatibility mode is not supported
- Microsoft Internet Explorer 9
Compatibility mode is not supported
- Mozilla Firefox 3.6 and above
- Google Chrome

As Google Chrome is updated frequently and it is not possible to deactivate automatic updates, Orchestra Networks only tests and makes a best effort to support the latest version available.

Note

As of version 5.4.0, Microsoft Internet Explorer 6 is no longer a supported browser.

Screen resolution

The minimum screen resolution for EBX5 is 1024x768.

Refreshing pages

Browser page refreshes are not supported by EBX5. When a page refresh is performed, the last user action is re-executed, thereby resulting in potential issues. It is thus imperative to use the action buttons and links offered by EBX5 instead of refreshing the page.

Browser configuration

The following features must be activated in your browser configuration:

- JavaScript,
- Ajax,
- pop-ups.

Attention

Avoid using any browser extensions or plug-ins, as they could interfere with the proper functioning of EBX5.

Supported application servers

EBX5 supports the following configurations:

- Java Runtime Environment: JRE 1.5 or higher version, obviously with the limits specified by the Java Virtual Machine implementation vendor. Note for example that, for JRE and JDK 1.5, Oracle states that "they are not updated with the latest security patches and are not recommended for use in production" (see [Oracle Java Archive site](#)).
- Any Java application server that complies with Servlet 2.3 (or higher), for example Tomcat 5.0, Tomcat 5.5, WebSphere Application Server 6 or higher, WebLogic Server 8.1 or higher. See [Java EE deployment overview](#).
- The application server must use the UTF-8 encoding for HTTP query strings from EBX5. This can be set at the application server level.

For example, on Tomcat, you can set the server to always use the UTF-8 encoding, by setting `URIEncoding` to 'UTF-8' on the `<Connector>` in the `server.xml` configuration file. Alternatively, you can set the server to use the encoding of the request's body, by setting the parameter `useBodyEncodingForURI` to 'true' in `server.xml`.

Supported databases

EBX5 repository supports the following Relational Database Management Systems with suitable JDBC drivers. It is important to follow database vendor recommendations and update policies regarding the database itself, as well as the JDBC driver:

- IBM DB2 UDB v8.2 or higher; however mapped table modes are not supported ([History](#), [Relational mode](#) and [Replication](#)).
- Oracle 10g R2 or higher; however [Relational mode](#) is only supported for Oracle 11g R2 or higher.
- PostgreSQL 8.4 or higher.
- Microsoft SQL Server 2008 or higher.
- H2 v1.2 or higher.

For other relational databases, contact [Orchestra Networks technical support](#).

Attention

In order to guarantee the integrity of the EBX5 repository, it is strictly forbidden to perform direct modifications to the database (for example, using direct SQL writes), except in the specific use cases described in the section [SQL access to data in relational mode](#).

See also:

- [Repository administration](#)
- [Data source of the EBX5 repository](#)
- [Configuring the EBX5 repository](#)

CHAPTER 56

Java EE deployment

Software components

EBX5 uses the following components:

- Library `ebx.jar`
- [Third-party Java libraries](#)
- [EBX5 built-in web applications](#) and optional [custom web applications](#)
- [EBX5 main configuration file](#)
- [EBX5 repository](#)
- [Default user and roles directory](#), integrated within the EBX5 repository, or a third-party system (LDAP, RDBMS) for user authentication

See also: [Supported environments](#)

Third-party libraries

EBX5 requires several third party Java libraries. These libraries must be deployed and accessible from the class-loader of `ebx.jar`. Depending on the application server being used, these libraries may already be present or need to be added manually.

Database drivers

EBX5 repository requires a database. Generally, the required driver is configured along with a data source, if using one. Depending on the database configured in the main configuration file, one of the following drivers is required:

H2	Version 1.2 validated. http://www.h2database.com/
Oracle JDBC	Oracle 10g validated. JAR files to include: <code>ojdbc14.jar</code> and <code>orai18n.jar</code>
DB2 JDBC	DB2 UDB V8.2 validated. JAR files to include: <code>db2jcc_license_cu.jar</code> and <code>db2jcc.jar</code>
SQL Server JDBC	SQL Server 2005 and SQL Server 2008 validated. JAR file to include: <code>sqljdbc.jar</code>
PostgreSQL	PostgreSQL 8.3 validated. Contact Orchestra Networks Professional Services if considering PostgreSQL 8.4. JAR file to include: <code>postgresql-8.3-604.jdbc3.jar</code>

See also:

- [Data source of the EBX5 repository](#)
- [Configuring the EBX5 repository](#)

SMTP and emails

The libraries for JavaMail 1.2 email management and the JavaBean Activation Framework are required.

The following libraries are used by email features in EBX5. See [Activating and configuring SMTP and e-mails](#) for the details of the configuration.

- `mail.jar`, version 1.2, from December 5, 2000
- `smtp.jar`, version 1.2, from December 5, 2000
- `pop3.jar`, version 1.2, from December 5, 2000
- `activation.jar`, version 1.0.1, from May 21, 1999, or the maintenance release version 1.0.2, from August 28, 2002

See also:

- [JavaMail](#)

- [JavaBeans Activation Framework](#)

Secure Socket Layer (SSL)

These libraries are required if your web applications use SSL features.

- `jsse.jar`: <http://www.oracle.com/technetwork/java/download-141865.html>
- `ibmjsse.jar`: <http://www.ibm.com/developerworks/java/jdk/security/>

See also: [EBX5 main configuration file](#)

Java Message Service (JMS)

When using JMS, a version 1.1 or higher is required.

Depending on whether a Java EE application server or a Servlet/Java Server Pages (JSP) implementation is being used, the library required is as follows:

- For an application server based on Java EE (Java Platform Enterprise Edition), the required JMS provider library is available by default. See <http://www.oracle.com/technetwork/java/javaee/overview> for more information.
- For a Servlet/Java Server Pages (JSP) implementation using Java SE (Java Platform Standard Edition), for example Apache Tomcat, a JMS provider library such as [Apache ActiveMQ](#) may need to be added. See <http://www.oracle.com/technetwork/java/javase/overview> for more information.

See also: [EBX5 main configuration file](#)

Web applications

EBX5 provides pre-packaged EARs that you may deploy directly if your enterprise has no custom web applications to add as EBX5 modules. If you are deploying custom web applications as EBX5 modules, it is recommended that you rebuild an EAR containing your custom modules packaged at the same level as the built-in web applications.

For more information, see the note on [repackaging the EBX5 EAR](#) at the end of this chapter.

EBX5 built-in web applications

EBX5 includes the following built-in web applications.

ebx	EBX5 entry point, which handles the initialization upon start up. See Deployment details for more information.
ebx-root-1.0	EBX5 root web application. Any application that uses EBX5 requires the root web application to be deployed.
ebx-manager	EBX5 user interface web application.
ebx-dma	EBX5 data model assistant, which helps with the creation of data models through the user interface. Note: EBX5 modeling tool requires the deployment of <code>ebx-manager</code> user interface web application.
ebx-dataservices	EBX5 data services web application. Data services allow external interactions with data spaces, data workflows, and the user and roles directory in the EBX5 repository using the SOAP and Web Services Description Language (WSDL) standards. Note: The EBX5 web service generator requires the deployment of the <code>ebx-manager</code> user interface web application.

Custom web applications

Every custom web application that is defined as an [EBX5 module](#) must be registered with the `ModulesRegister.registerWebApp()` API. Registration of modules is explained in the [EBX5 modules](#) chapter.

See also:

- [Registration](#)
- [Module registration](#)

Deployment details

Introduction

This section describes the various options that are offered for the deployment of `ebx` web application. These options are available in its deployment descriptor (the file `WEB-INF/web.xml`) and are complemented by the various properties defined in the main configuration file.

Attention

For JBoss application servers, any unused resources must be removed from the `WEB-INF/web.xml` deployment descriptor.

See also:

- [EBX5 main configuration file](#)
- [Supported application servers](#)

User interface and web access

The web application 'ebx' (packaged as `ebx.war`) contains the servlet `FrontServlet`, which handles initialization and serves as the single user interface entry point for the EBX5 web tools.

Configuring the deployment descriptor for 'FrontServlet'

In the file `WEB-INF/web.xml` of the web application 'ebx', the following elements must be configured for `FrontServlet`:

<code>/web-app/servlet/load-on-startup</code>	To ensure that <code>FrontServlet</code> initializes upon EBX5 start up, the <code>web.xml</code> deployment descriptor must specify the element <code><load-on-startup>1</load-on-startup></code> .
<code>/web-app/servlet-mapping/url-pattern</code>	<code>FrontServlet</code> must be mapped to the path <code>'/'</code> .

Configuring the application server for 'FrontServlet'

- `FrontServlet` must be authorized to access other contexts, such as `ServletContext`.

For example, on Tomcat, this configuration is done using the attribute `crossContext` in the configuration file `server.xml`, as follows:

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

- When several EBX5 web components are to be displayed on the same HTML page, for instance using `iFrames`, it may be required to disable the management of cookies due to limitations present in certain Internet browsers.

For example, on Tomcat, this configuration is provided by the attribute `cookies` in the configuration file `server.xml`, as follows:

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

Data source of the EBX5 repository

Note

If the EBX5 main configuration specifies the property `ebx.persistence.url`, then the environment entry below will be ignored by EBX5 runtime. See [Configuring the EBX5 repository](#) for more information on this property.

The JDBC data source for EBX5 is specified in the deployment descriptor `WEB-INF/web.xml` of the `ebx` web application as follows:

Reserved resource name	Description
<code>jdbc/EBX_REPOSITORY</code>	JDBC data source for EBX5 Repository. Java type: <code>javax.sql.DataSource</code>

See also:

- [Configuring the EBX5 repository](#)
- [Rules for the access to the database and user privileges](#)
- *[see HTML documentation] Oracle data source configuration on WebSphere Application Server 6*
- *[see HTML documentation] SQL Server data source configuration on WebSphere Application Server 6*

Mail sessions

Note

If the EBX5 main configuration does not set `ebx.mail.activate` to 'true', or if it specifies the property `ebx.mail.smtp.host`, then the environment entry below will be ignored by EBX5 runtime. See [SMTP](#) in the EBX5 main configuration properties for more information on these properties.

SMTP and email is declared in the deployment descriptor `WEB-INF/web.xml` of the `ebx` web application as follows:

Reserved resource name	Description
<code>mail/EBX_MAIL_SESSION</code>	Java Mail session used to send e-mails from EBX5. Java type: <code>javax.mail.Session</code>

JMS connection factory

Note

If the EBX5 main configuration does not activate JMS through the property `ebx.jms.activate`, then environment entry below will be ignored by the EBX5 runtime. See [JMS](#) in the EBX5 main configuration properties for more information on this property.

The JMS connection factory is declared in the deployment descriptor `WEB-INF/web.xml` of the `ebx` web application as follows:

Reserved resource name	Description	Required
<code>jms/EBX_JMSConnectionFactory</code>	<p>JMS connection factory used by EBX5 to create connections with the JMS provider configured in the operational environment of the application server.</p> <p>Java type: <code>javax.jms.QueueConnectionFactory</code></p>	Yes

Note

For deployment on JBoss and WebLogic application servers with JNDI capabilities, it is necessary to update `EBX5.ear` or `EBX5ForWebLogic.ear` respectively for additional mappings of all required resource names to JNDI names.

JMS for data services

To configure data services to use JMS instead of the default HTTP, you must configure the [JMS connection factory](#) and the following queues, declared in the `WEB-INF/web.xml` deployment descriptor of the `ebx` web application. This is the only method for configuring JMS for data services.

See [JMS](#) for more information on the associated EBX5 main configuration properties.

Note

If the EBX5 main configuration does not activate JMS through the property `ebx.jms.activate`, then the environment entries below will be ignored by EBX5 runtime. See [JMS](#) in the EBX5 main configuration properties for more information on this property.

Reserved resource name	Description	Required
<code>jms/EBX_QueueIn</code>	JMS queue for incoming messages sent to EBX5 by other applications. Java type: <code>javax.jms.Queue</code>	Yes
<code>jms/EBX_QueueFailure</code>	JMS queue for failures. It is used for incoming messages for which an error has occurred. This allows replaying these messages if necessary. Java type: <code>javax.jms.Queue</code> Note: For this property to be read, the main configuration must also activate the queue for failures through the property <code>ebx.jms.activate.queueFailure</code> . See JMS in the EBX5 main configuration properties for more information on these properties.	No

JMS for distributed data delivery (D3)

To configure D3 to use JMS instead of the default HTTP and TCP protocols, you must configure the [JMS connection factory](#) and the following queues, declared in the `WEB-INF/web.xml` deployment descriptor of the `ebx` web application.

Note

If the EBX5 main configuration does not activate JMS and D3 (slave, hub or master node) through the properties `ebx.d3.mode`, `ebx.jms.activate` and `ebx.jms.d3.activate` then the environment entries below will be ignored by EBX5 runtime. See [JMS](#) and [Distributed data delivery \(D3\)](#) in the EBX5 main configuration properties for more information on these properties.

Common declarations for master and slave nodes

Reserved resource name	Description	Master node	Slave node
<code>jms/EBX_D3MasterQueue</code>	D3 master JMS queue (only for D3 mode 'slave' or 'hub'). It specifies the name of the communication queue with master D3 node. Java type: <code>javax.jms.Queue</code>	N/A	Required
<code>jms/EBX_D3ReplyQueue</code>	D3 Reply JMS queue (for all D3 modes except 'single' mode). It specifies the name of the reply queue for D3 node conversation. Java type: <code>javax.jms.Queue</code>	Required	Required
<code>jms/EBX_D3SlaveArchiveQueue</code>	D3 slave Archive JMS queue (only for D3 mode 'slave' or 'hub'). It specifies the name of slave transfer archive queue used by master D3 node. Java type: <code>javax.jms.Queue</code>	N/A	Required
<code>jms/EBX_D3SlaveCommunicationQueue</code>	D3 slave Communication JMS queue (only for D3 mode 'slave' or 'hub'). It specifies the name of the slave communication queue used by master D3 node. If this property is not specified, EBX5 uses <code>jms/EBX_QueueIn</code> by default. Java type: <code>javax.jms.Queue</code>	N/A	Optional

Additional declarations on the master node

If D3 enables JMS, then the deployment descriptor of the master node must specify the specific queues associated with each slave. The following table provides an example of the additional entries that are needed for communicating with a slave node "SL1" (the next section describes the environment on which these settings are declared).

Resource name for slave "SL1"	Description	Required
<code>SL1_QueueIn</code> (custom name, see note below)	Queue corresponding to <code>jms/EBX_QueueIn</code> of slave node. Java type: <code>javax.jms.Queue</code>	One of either this queue or the queue 'SL1_D3SlaveCommunicationQueue' must be defined
<code>SL1_D3SlaveArchiveQueue</code> (custom name, see note below)	Queue corresponding to <code>jms/EBX_D3SlaveArchiveQueue</code> of slave node. Java type: <code>javax.jms.Queue</code>	Yes
<code>SL1_D3SlaveCommunicationQueue</code> (custom name, see note below)	Queue corresponding to <code>jms/EBX_D3SlaveCommunicationQueue</code> of slave node. Java type: <code>javax.jms.Queue</code>	One of either this queue or the queue 'SL1_QueueIn' must be defined

Note

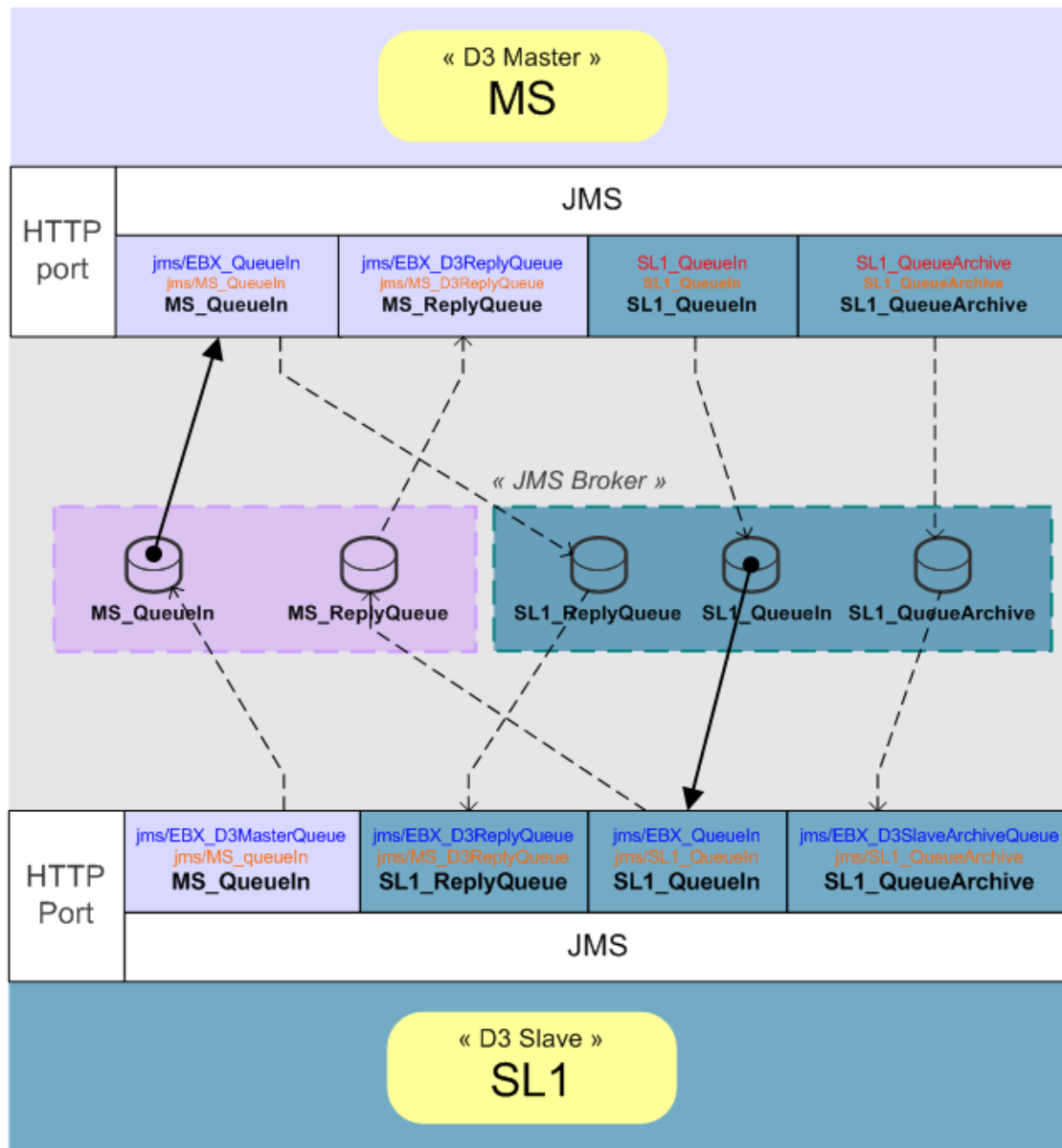
Due to restrictions on the master node, the resource name of the slave communication queue must be the same as the physical name of the queue.

Note

For deployment on JBoss and WebLogic application servers with JNDI capabilities, it is necessary to update `EBX5.ear` or `EBX5ForWebLogic.ear` respectively for additional mappings of all required resource names to JNDI names.

Example of a simple JMS configuration between a master node and a slave node

Message Queueing (D3 / JMS)



Reserved or custom resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

JNDI resource name (declared in 'WEB-INF/weblogic.xml' or 'WEB-INF/jboss-web.xml' of 'ebx.war')

Physical name (resource adapter / messaging)

← ● Consumer

← - - - Volatile exchange

Java EE deployment examples

EBX5 can be deployed on any Java EE application server supporting at least Servlet 2.3. The following documentation on Java EE deployment and installation notes are available:

- *[see HTML documentation]* Installation on Tomcat 5
- *[see HTML documentation]* Installation on WebSphere 6
- *[see HTML documentation]* Installation on WebLogic 8.1

Attention

- The EBX5 installation notes on Java EE application servers do not replace the native documentation of each application server.
- They are *not* general installation recommendations, as the installation process is determined by architectural decisions, such as the technical environment, application mutualization, delivery process, and organizational decisions.
- In these examples, no additional EBX5 modules are deployed. To deploy additional modules, the best practice is to rebuild an EAR with the module as a web application at the same level as the other EBX5 modules. The web application must declare its class path dependency as specified by the Java™ 2 Platform Enterprise Edition Specification, v1.4:

J2EE.8.2 Optional Package Support

(...)

A JAR format file (such as a JAR file, WAR file, or RAR file) can reference a JAR file by naming the referenced JAR file in a Class-Path header in the Manifest file of the referencing JAR file. The referenced JAR file is named using a URL relative to the URL of the referencing JAR file. The Manifest file is named META-INF/MANIFEST.MF in the JAR file. The Class-Path entry in the Manifest file is of the form:

Class-Path: list-of-jar-files-separated-by-spaces

In an "industrialized" process, it is strongly recommended to develop a script that automatically builds the EAR, with the custom EBX5 modules, the EBX5 web applications, as well as all the required shared libraries.

- In order to avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar` and other libraries in the class-loading system.

Limitations

Clustering

EBX5 does not include a cache synchronization mechanism, thus it cannot be deployed into a cluster of active instances.

See [Technical Architecture](#) for more information.

CHAPTER 57

EBX5 main configuration file

Overview of the EBX5 main configuration file

The main configuration file, by default named `ebx.properties`, contains most of the basic parameters for running EBX5. This is a Java properties file that uses the [standard simple line-oriented format](#).

The main configuration file complements the [Java EE deployment descriptor](#). Additional configuration can also be done by the administrator through the user interface, which is then stored in the EBX5 repository.

See also:

- [Deployment details](#)
- [Front end administration](#)

Location of the file

The access path to the main configuration file can be specified in several ways. In order of descending priority:

1. By defining the Java system property 'ebx.properties'. For example, this property can be set by adding the option `-Debx.properties=<filePath>` in the `java` command-line. See [Java documentation](#).
2. By defining the servlet initialization parameter 'ebx.properties'.

This standard Java EE setting must be specified in the `web.xml` file of the web application 'ebx'. EBX5 accesses this parameter by calling the method [ServletConfig.getInitParameter\("ebx.properties"\)](#) in the servlet `FrontServlet`.

3. By default, if nothing is specified, the main configuration file is located at `WEB-INF/ebx.properties` of the web application 'ebx'.

Note

In addition to specifying properties in the main configuration file, you may also set the values of properties directly in the system properties (`-D` argument of the `java` command).

Custom properties and variable substitution

The value of every property can include one or more variables using the syntax `${propertyKey}`, where 'propertyKey' is either a system property, or a property defined in the main configuration file.

For example, the default configuration file provided with EBX5 uses the custom property `ebx.home` for setting a default common directory.

Setting EBX5 license key

See also: [Installing a repository using the Configuration Assistant](#)

```
#####
## EBX5 License number
## (as specified by your license agreement)
#####
ebx.license=paste_here_your_license_key
```

Setting the EBX5 root directory

The EBX5 root directory contains the archives, the XML audit trail and, when the repository is persisted on H2 standalone mode, the H2 database files.

```
#####
## Path for EBX5 XML repository
#####
ebx.repository.directory=${ebx.home}/ebxRepository
```

Configuring the EBX5 repository

Before configuring the access to EBX5 repository, carefully read the section [Technical architecture](#) in the chapter 'Repository administration'.

The required library (driver) for each supported database is described in the chapter [Database drivers](#).

See also:

- [Repository administration](#)
- [Rules for the access to the database and user privileges](#)
- [Supported databases](#)
- [Data source of the EBX5 repository](#)
- [Database drivers](#)

```
#####
## The maximum time to set up the database connection,
## in milliseconds.
#####
ebx.persistence.timeout=10000
#####
```



```

## The prefix to add to all table names of persistence system.
## This may be useful for supporting multiple repositories in the relational database.
## Default value is 'EBX_'.
#####
ebx.persistence.table.prefix=

#####
## Case EBX5 persistence system is H2 'standalone'.
#####
ebx.persistence.factory=h2.standalone
ebx.persistence.user=sa
ebx.persistence.password=

#####
## Case EBX5 persistence system is H2 'server mode',
#####
#ebx.persistence.factory=h2.server

## Specific properties to be set only only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url=jdbc:h2:tcp://127.0.0.1/ebxdb
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyyy

#####
## Case EBX5 persistence system is Oracle database.
#####
#ebx.persistence.factory=oracle

## Specific properties to be set only only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url=jdbc:oracle:thin:@127.0.0.1:1521:ebxDatabase
#ebx.persistence.driver=oracle.jdbc.OracleDriver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyyy

#####
## Case EBX5 persistence system is IBM DB2.
#####
#ebx.persistence.factory=db2

## Specific properties to be set only only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url=jdbc:db2://127.0.0.1:50000/ebxDatabase
#ebx.persistence.driver=com.ibm.db2.jcc.DB2Driver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyyy

#####
## Case EBX5 persistence system is Microsoft SQL Server.
#####
#ebx.persistence.factory=sqlserver

## Specific properties to be set only only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url= \

```

```
#jdbc:sqlserver://127.0.0.1:1036;databasename=ebxDatabase
#ebx.persistence.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyyy

#####
## Case EBX5 persistence system is PostgreSQL.
#####
#ebx.persistence.factory=postgresql

## Specific properties to be set only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url=jdbc:postgresql://127.0.0.1:5432/ebxDatabase
#ebx.persistence.driver=org.postgresql.Driver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyyy
```

Configuring the user and roles directory

This parameter specifies the Java directory factory class name. It must be defined only when not using the default EBX5 directory.

See also:

- [Users and roles directory](#)
- **DirectoryFactory** [*DirectoryFactory*]^{API}

```
#####
## Specifies the Java directory factory class name.
## Value must be the fully qualified name of the Java class.
## The class must extend com.orchestranetworks.service.directory.DirectoryFactory.
#####
#ebx.directory.factory=xxx.yyy.DirectoryFactoryImpl
```

It is also possible to disable the built-in role ADMINISTRATOR.

```
#####
## Specifies whether the built-in role ADMINISTRATOR is disabled.
## Default value is false.
#####
#ebx.directory.disableBuiltInAdministrator=true
```

Setting temporary files directories

Temporary files are stored as follows:

```
#####
## Directories for temporary resources.
#####
# The property ebx.temp.directory allows to specify a directory for temporary files.
# Default value is java.io.tmpdir
#
#ebx.temp.directory = ${java.io.tmpdir}
```

```
#ebx.temp.directory = /tmp/java

# The property ebx.temp.cache.directory allows to specify the directory containing
temporary files for cache.
# Default value is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.cache.directory = ${ebx.temp.directory}/ebx.platform

# The property ebx.temp.import.directory allows to specify the directory containing
temporary files for import.
# Default value is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.import.directory = ${ebx.temp.directory}/ebx.platform
```

Activating the XML audit trail

By default, the XML audit trail is activated. It can be deactivated using the following variable:

```
#####
# The XML history has been replaced by an SQL history.
# This old XML history can be deactivated using the following variable.
# Default is true.
#####
ebx.history.xmlaudittrail.activated = true
```

Configuring the EBX5 logs

The most important logging categories are:

- The category `ebx.log4j.category.log.kernel` displays EBX5 main features, processes and exceptions.
- The category `ebx.log4j.category.log.workflow` displays main features, warnings and exceptions about workflow.
- The category `ebx.log4j.category.log.setup` displays validation and compilation results of all EBX5 objects.
- The category `ebx.log4j.category.log.mail` displays the activity related to the emails sent by the server (see [Activating and configuring SMTP and e-mails](#)). Note that this category shall not use the [Specific SMTP appender](#) in order to prevent infinite loops...
- The category `ebx.log4j.category.log.d3` displays D3 events on EBX5.
- The category `ebx.log4j.category.log.dataservices` displays Data Services events on EBX5.
- The category `ebx.log4j.category.log.monitoring` displays raw logs for [monitoring](#).
- The category `ebx.log4j.category.log.request` displays the optimization strategy for every **request** [Request]^{API} issued on a semantic table in the EBX5 repository.

Some of these categories can also be written by specific developments through the **LoggingCategory** [LoggingCategory]^{API} interface.

```
#####
## Log4J properties:
##
## We have some specific syntax extensions:
## - Appender ebxFile:<aFileName>
## Defines a file appender with default settings (threshold=DEBUG)
##
## - property log.defaultConversionPattern is set by Java
#####
#ebx.log4j.debug=true
#ebx.log4j.disableOverride=
#ebx.log4j.disable=
ebx.log4j.rootCategory= INFO
ebx.log4j.category.log.kernel= INFO, Console, ebxFile:kernel, kernelMail
ebx.log4j.category.log.workflow= INFO, ebxFile:workflow
ebx.log4j.category.log.persistence= INFO, ebxFile:persistence
ebx.log4j.category.log.setup= INFO, Console, ebxFile:kernel
ebx.log4j.category.log.mail= INFO, Console, ebxFile:mail
ebx.log4j.category.log.frontEnd= INFO, Console, ebxFile:kernel
ebx.log4j.category.log.frontEnd.incomingRequest= INFO
ebx.log4j.category.log.frontEnd.requestHistory= INFO
ebx.log4j.category.log.frontEnd.UIComponentInput= INFO
ebx.log4j.category.log.fsm= INFO, Console, ebxFile:fsm
ebx.log4j.category.log.fsm.dispatch= INFO
ebx.log4j.category.log.fsm.pageHistory= INFO
ebx.log4j.category.log.wbp= FATAL, Console
#-----
ebx.log4j.appender.Console.Threshold = INFO
ebx.log4j.appender.Console=com.onwbp.org.apache.log4j.ConsoleAppender
ebx.log4j.appender.Console.layout=com.onwbp.org.apache.log4j.PatternLayout
ebx.log4j.appender.Console.layout.ConversionPattern=${log.defaultConversionPattern}
#-----
ebx.log4j.appender.kernelMail.Threshold = ERROR
ebx.log4j.appender.kernelMail = com.onwbp.org.apache.log4j.net.SMTPAppender
ebx.log4j.appender.kernelMail.To = admin@domain.com
ebx.log4j.appender.kernelMail.From = admin${ebx.site.name}
ebx.log4j.appender.kernelMail.Subject = EBX5 Error on Site ${ebx.site.name} (VM
${ebx.vm.id})
ebx.log4j.appender.kernelMail.layout.ConversionPattern=**Site ${ebx.site.name} (VM
${ebx.vm.id})*%n${log.defaultConversionPattern}
ebx.log4j.appender.kernelMail.layout = com.onwbp.org.apache.log4j.PatternLayout

#-----
ebx.log4j.category.log.monitoring= INFO, ebxFile:monitoring
ebx.log4j.category.log.dataServices = INFO, ebxFile:dataServices
ebx.log4j.category.log.d3= INFO, ebxFile:d3
ebx.log4j.category.log.request= INFO, ebxFile:request
```

Specific appender 'ebxFile'

The token `ebxFile:` can be used as a shortcut to define a daily rolling file appender with default settings. It must be followed by a file name. It then activates an appender that writes to a file located in the directory `ebx.logs.directory`, with a threshold set to `DEBUG`.

The property `ebx.log4j.appender.ebxFile.backup.Threshold` allows the definition of the maximum number of backup files for daily rollover.

```
#####
## Directory of log files 'ebxFile:'
## This property is used by special appender prefixed
## by 'ebxFile:' (see log section below)
#####
ebx.logs.directory=${ebx.home}/ebxLog

#####
# Daily rollover threshold of log files 'ebxFile:'
# Specifies the maximum number of backup files for daily rollover of 'ebxFile:' appenders.
# When set to a negative value, backup log files are never purged.
# Default value is -1.
#####
ebx.log4j.appender.ebxFile.backup.Threshold=-1
```

Specific SMTP appender

The appender `com.onwbp.org.apache.log4j.net.SMTPAppender` provides an asynchronous email sender.

See also: [Activating and configuring SMTP and e-mails](#)

Activating and configuring SMTP and e-mails

The internal mail manager sends emails asynchronously. It is used by the workflow engine and by the specific log4J emails appender `com.onwbp.org.apache.log4j.net.SMTPAppender`.

See also: [Mail sessions](#)

```
#####
## SMTP and e-mails
#####

## Activate e-mails (true or false, default is false).
## If activated, the deployer must ensure that the entry 'mail/EBX_MAIL_SESSION' is bound
## in the operational environment of the application server (except if a specific email
## configuration is used by setting the property ebx.mail.smtp.host below).
#ebx.mail.activate=false

## Polling interval is in seconds (default is 10).
#ebx.mail.polling.interval=10

## Specific properties to be set only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.mail.smtp.host = smtp.domain.com
## SMTP port default is 25.
#ebx.mail.smtp.port= 25
#ebx.mail.smtp.login=
#ebx.mail.smtp.password=
## Activate SSL (true or false, default is false).
## If SSL is activated, a SSL factory and a SSL provider are required.
```

```
#ebx.mail.smtp.ssl.activate=true
#ebx.mail.smtp.ssl.provider=com.sun.net.ssl.internal.ssl.Provider
#ebx.mail.smtp.ssl.factory=javax.net.ssl.SSLSocketFactory
```

Configuring data services

```
#####
## Data services
#####

# Specifies the default value of the data services parameter
'disableRedirectionToLastBroadcast'.
# Default is false.
#ebx.dataservices.disableRedirectionToLastBroadcast.default=false

# Specifies the default value for deletion at the end of close and merge operations.
# If the parameter is set in the request operation, it overrides this default setting.
# If unspecified, default is false.
#ebx.dataservices.dataDeletionOnCloseOrMerge.default=false
#ebx.dataservices.historyDeletionOnCloseOrMerge.default=false
```

Activating and configuring JMS

See also: [JMS for data services](#)

```
#####
## JMS configuration for Data Services
#####

## Activates JMS (true or false, default is false).
## If activated, the deployer must ensure that the entries 'jms/EBX_QueueIn' and 'jms/
EBX_JMSConnectionFactory'
## are bound in the operational environment of the application server.
#ebx.jms.activate=false

## Activates JMS queue for failures (true or false, default is false).
## If activated, the deployer must ensure that the entry 'jms/EBX_QueueFailure' is bound
## in the operational environment of the application server.
#ebx.jms.activate.queueFailure=false

## Number of concurrent listener(s)
## Default is 3.
## Property is used if ebx.jms.activate is set to true.
#ebx.jms.listeners.count=3
```

Configuring distributed data delivery (D3)

See [Configuring D3 nodes](#) for the main configuration file properties pertaining to D3.

See also:

- [JMS for distributed data delivery \(D3\)](#)
- [Introduction to D3](#)

Configuring web access from end user browsers

URLs computing

By default, EBX5 runs in "standalone" mode, where external resources (images, JavaScript, etc.) are provided by the application server.

By default, URL-related parameters in the main configuration file do not have to be set.

In this case, the server name and the port are obtained from the initial request sent to EBX5.

```
#####
## EBX5 FrontServlet: default properties for computing servlet address
##
## {useLocalUrl}:
## If set to true, servlet address is a "local absolute" URL.
## (that is, a relative URL consisting of an absolute path: "/path")
## See RFC 2396, http://www.ietf.org/rfc/rfc2396.txt).
## This property is defined once for HTTP and HTTPS.
## Default value is false.
##
## {host}:
## If neither defined nor adapted, retrieves initial request host
## {port}:
## If neither defined nor adapted, retrieves initial request host
## {path}:
## Mandatory, may be empty
##
## Resulting address will be:
## protocol://{host}:{port}/{path}
##
## Each property for HTTP (except {port}) may be inherited from HTTPS property,
## and reciprocally.
#####

#ebx.servlet.useLocalUrl=true
#ebx.servlet.http.host=
#ebx.servlet.http.port=
#ebx.servlet.http.path=ebx/
#ebx.servlet.https.host=
#ebx.servlet.https.port=
#ebx.servlet.https.path=

#####
## External resources: default properties for computing external resources address
##
## The same rules apply as EBX5 FrontServlet properties (see comments).
##
## Each property may be inherited from EBX5 FrontServlet.
#####

#ebx.externalResources.useLocalUrl=true
#ebx.externalResources.http.host=
#ebx.externalResources.http.port=
#ebx.externalResources.http.path=
#ebx.externalResources.https.host=
#ebx.externalResources.https.port=
```

```
#ebx.externalResources.https.path=
```

Proxy mode

Proxy mode allows using a front-end HTTP server to provide static resources (images, CSS, JavaScript, etc.). This architecture reduces the load on the application server for static HTTP requests. In addition, this configuration allows using SSL security on the front-end server.

The web server sends requests to the application server according to a path in the URL.

This 'servletAlias' path is specified in the main configuration file.

The web server provides all external resources. Those resources are stored in a dedicated directory, accessible using the path 'resourcesAlias'.

EBX5 must also be able to access external resources from the file system. To do so, the property `ebx.webapps.directory.externalResources` must be specified.

The main configuration file is configured as follows:

```
#####
## Path for external resources if they are not
## delivered within web applications
## This field is mandatory if in proxy mode.
#####
ebx.webapps.directory.externalResources=
D:/http/resourcesFolder

#####

#ebx.servlet.useLocalUrl=true
#ebx.servlet.http.host=
#ebx.servlet.http.port=
ebx.servlet.http.path= servletAlias
#ebx.servlet.https.host=
#ebx.servlet.https.port=
ebx.servlet.https.path= servletAlias

#####

#ebx.externalResources.useLocalUrl=true
#ebx.externalResources.http.host=
#ebx.externalResources.http.port=
ebx.externalResources.http.path= resourcesAlias
#ebx.externalResources.https.host=
#ebx.externalResources.https.port=
ebx.externalResources.https.path= resourcesAlias
```

Reverse proxy mode

URLs generated by EBX5 for requests and external resources must contain a server name, a port number and a specific path prefix.

The main configuration file is configured as follows:

```
#####
ebx.servlet.http.host= reverseDomain
#ebx.servlet.http.port=
ebx.servlet.http.path=ebx/
ebx.servlet.https.host= reverseDomain
#ebx.servlet.https.port=
ebx.servlet.https.path=ebx/
#####
## Web parameters (for external resources)
## if nothing is set, values are taken from servlet.
#####
ebx.externalResources.http.host= reverseDomain
#ebx.externalResources.http.port=
#ebx.externalResources.http.path=ebx/
ebx.externalResources.https.host= reverseDomain
#ebx.externalResources.https.port=
ebx.externalResources.https.path=ebx/
```

URL parameters encoding

URLs generated by EBX5 may contain sensitive characters according to some security policies. This can be avoided by enforcing strong url encoding.

The main configuration file is configured as follows:

```
#####
## URL parameter strong encoding
##
## If true, sensitive parameters are strongly encoded,
## by removing special characters (./'=', etc.) from URLs.
##
## Default value is false.
#####
ebx.urlParameters.strongEncoding=true
```

Tuning the EBX5 repository

Some options can be set so as to optimize memory usage.

The main configuration file is configured as follows:

```
#####
## Technical parameters for memory and performance tuning
#####
# Import commit threshold allows to specify the commit threshold
# exclusively for the archive import launched directly from Manager.
#
# For more details about the commit threshold,
# see the JavaDoc ProcedureContext.setCommitThreshold().
# Default value is 0.
#
ebx.manager.import.commit.threshold=100
# Validation messages threshold allows to specify the maximum number of
```

```
# messages to consider when performing a validation.
# This threshold is considered for each severity in each validation report.
# When the threshold is reached:
# - for severities 'error' or 'fatal', the validation is stopped.
# - for severities 'info' or 'warning', the validation continues without
# registering messages beyond the threshold. However the number of messages
# is still counted and messages of other severities can still be added.
#
# When set to 0 or a negative value the threshold is not considered.
# Default value is 0.
#
ebx.validation.messages.threshold=100
```

Miscellaneous

Activating data workflows

This parameter specifies whether the data workflow is activated. This parameter is not taken into account on the fly. The server must be restarted whenever the value changes.

```
#####
## Workflow activation.
## Default is false.
#####
ebx.workflow.activation = true
```

Deployment site identification

This parameter allows you to specify the address for technical log emails.

```
#####
## Unique Site Name
## --> used by monitoring mails and by repository
#####
ebx.site.name= name@domain.com
```

Dynamically reloading the main configuration

Some parameters can be dynamically reloaded, without restarting EBX5. The parameter 'thisfile.checks.intervalInSeconds' indicates the time interval between each check of the main configuration file.

```
#####
### Checks if this file has been updated
### If value <= 0, no more checks will be done
#####
thisfile.checks.intervalInSeconds=1
```

In development mode, this parameter can be set as low as one second. On production systems, where changes are expected to be less frequent, the value can be greater, or set to '0' to disable hot reloading entirely.

This property is not always supported when the module is deployed as a WAR, as it depends on the application server.

Applications that use the EBX5 navigation engine

Application template reloading:

```
#####
## Reload templates when it is updated
## (default value for all EBX5 modules).
## (value can be overridden by each EBX5 module.
#####
templates.checksIfUpdated=true
```

Debug mode in the EBX5 modules web pages (use only when developing):

```
#####
## End-User Debug Mode
## (default for all EBX5 modules ).
## Debug information appears on end-user web page.
#####
frontEnd.debugMode=false
```

Module registration

When the application server is started, all web applications declared as EBX5 modules have to register. Concurrently, depending on the loading strategy of data spaces, the deployment of the web application `ebx` may require the compilation of data models and their modules. Since these modules may not yet be registered (it depends on the order of web application deployment at server startup), a wait loop is implemented. This gives the module time to be registered.

See also: [Packaging EBX5 modules](#)

```
#####
## When the application server is started, all web applications declared as
## EBX5 modules have to register. This property
## specifies the estimated time in seconds taken by the application server
## to deploy all its web applications at startup. Beyond this time,
## if a required module has not yet registered, it is considered to be absent
## and an error is reported to 'kernel' log.
## Default is 30 seconds.
#####
#ebx.module.timeInSecondsForModuleRegistration=30
```

Running mode

This property defines how EBX5 runs. Three modes are available: *development*, *integration* and *production*.

When running in *development* mode, the [development tools](#) are activated in EBX5 and more technical information is displayed.

Note

The administrator always has access to this information regardless of the mode used.

The technical information is as follows:

Documentation pane	In the case of a computed value, the Java class name is displayed. The button to get the path to a node is also displayed.
Data model assistant	Data model configuration and additional options, such as Services, Business Objects and Rules, Java Bindings, and some advanced properties.
Validation report	The <i>Validation</i> tab is displayed.
Log	The logs include additional technical information intended for the developer. For example, a warning is written to logs if a drop-down list is defined on a node which is not an enumeration in a UI Bean.

```
#####
## Server Mode
## Value must be one of: development, integration, production
## Default is production.
#####
backend.mode=integration
```

Note

There is no difference between the modes *integration* and *production*.

Resource filtering

This property allows the filtering of certain files and directories in the resource directory contents (resource type node, with an associated facet that indicates the directory that contains usable resources).

```
#####
## list (separated by comma) of regexps excluding resource
## the regexp must be of type "m:[pattern]:[options]".
## the list can be void
#####
ebx.resource.exclude=m:CVS/*:
```

CHAPTER 58

Installing a repository using the Configuration Assistant

The EBX5 Configuration Assistant guides you through the initial configuration of the EBX5 repository. If EBX5 does not have a repository installed upon start up, the configuration assistant launches automatically.

Before starting the configuration of the repository, ensure that EBX5 is correctly deployed on the application server. For further information about EBX5 deployment, see [Java EE deployment overview](#).

Note

The main configuration file `ebx.properties` must also be correctly configured. See [EBX5 configuration properties](#) for more information.

License key

When you launch EBX5, the license key page displays automatically if no valid license key is available, that is, if there is no license key entered in the main configuration file `ebx.properties`, or if the current license key has expired.

If you do not have a license key, you may obtain a trial license key at <http://www.orchestranetworks.com/support>.

Configuration steps

The EBX5 configuration assistant guides you through the following steps:

1. Validating the license agreement.
2. Configuring the repository
3. Defining users in the default user and roles directory (if a custom directory is not defined)
4. Validating the information entered
5. Installing the EBX5 repository

Validating the license agreement

The page of the configuration assistant displays the product license agreement. In order to proceed with the configuration, you must read and accept the license agreement.

Configuring the repository

This page displays some of the properties defined in the EBX5 main configuration file `ebx.properties`. You also define several basic properties of the repository in this step.

Id of the repository (<code>repositoryId</code>)	Must uniquely identify the repository (in the scope of the enterprise). The identifier is 48 bits (6 bytes) long and is usually represented as 12 hexadecimal digits. This information is used for generating the Universally Unique Identifiers (UUIDs) of entities created in the repository, and also of transactions logged in the history. This identifier acts as the "UUID node", as specified by RFC 4122.
Repository label	Defines a user-friendly label that indicates the purpose and context of the repository.

Defining users in the default directory

If a custom user and roles directory is not defined in the main configuration file `ebx.properties`, the EBX5 configuration assistant allows you to define default users for the default user and roles directory. For more information about the default user and roles directory, see the [User and roles directory](#).

An administrator user must be defined. You may optionally create a second user.

Validating the information entered

Before proceeding with the installation of the repository, you can review the configuration of the repository and the information you have entered on the 'Configuration Summary' page. If you need to modify information, you can return to the previous pages using the configuration assistant's **< Back** buttons.

Once you have verified the configuration, click the button **Install the repository** > to proceed with the installation.

Installing the EBX5 repository

The repository installation is performed using the information that you provided. When the installation is complete, you are redirected to the repository login page.

CHAPTER **59**

Deploying and registering EBX5 add-ons

Note

Refer to the documentation of each add-on for additional installation and configuration information in conjunction with that which is found here.

Deploying an add-on module

Note

Each EBX5 add-on build is intended to run with a specific version of EBX5. Ensure that you have the correct build of the add-on for the version of EBX5 on which it will run.

The web application deployment descriptor for the add-on module must specify that class definitions and resources from the web application are to be loaded in preference to classes from the parent and server classloaders.

For example, on WebSphere Application Server, this can be done by setting `<context-priority-classloader>true</context-priority-classloader>` in the `web-app` element of the deployment descriptor.

On WebLogic, include `<prefer-web-inf-classes>True</prefer-web-inf-classes>` in `weblogic.xml`.

See the documentation on class loading of your application server for more information.

Registering an add-on module

To register a new EBX5 add-on in the repository:

1. Navigate to the 'Administration' area.
2. Click the down-arrow in the navigation pane and select the 'Add-ons' entry.
3. On the **Add-ons > Registered Add-ons** page, click the **+** button to create a new entry.
4. Select the add-on you are registering, and enter its license key.

Note

If the EBX5 repository is under a trial license, no license key is required for the add-on. The add-on will be subject to the same trial period as the EBX5 repository itself.

5. Click **Submit**.

Technical administration

CHAPTER 60

Repository administration

Technical architecture

Overview

The main principles of the EBX5 technical architecture are as follows:

- A Java process (JVM) that runs EBX5 is limited to a single EBX5 repository. This repository is physically persisted in a [supported relational database instance](#) accessed through a [configured data source](#).
- A repository cannot be shared by multiple JVMs at any given time. However, a failover architecture may be used. These aspects are detailed in the subsequent sections [Single JVM per repository](#) and [Failover with hot-standby](#). Furthermore, to achieve horizontal scalability, an alternative is to deploy a [distributed data delivery \(D3\)](#) environment.
- A single relational database instance can support multiple EBX5 repositories (used by distinct JVMs). This is done by specifying distinct table prefixes using the property `ebx.persistence.table.prefix`.

See also:

- [Configuring the EBX5 repository](#)
- [Supported databases](#)
- [Data source of the EBX5 repository](#)

Rules for the access to the database and user privileges

Attention

In order to guarantee the integrity of persisted master data, **it is strictly forbidden to perform direct SQL writes to the database**, except in the specific use cases described in the section [SQL access to data in relational mode](#).

It is recommended for the database user specified by the [configured data source](#) to have 'create/alter' privileges on tables, indexes and sequences. This allows [automatic repository installation and upgrades](#).

If this is not the case, it is still possible to perform [manual repository installation and upgrades](#), however, [Relational mode](#) and [History](#) will be inconvenient to maintain since modification of concerned models will require manual intervention.

See also:

- [SQL access to history](#)
- [Accessing a replica table using SQL](#)
- [SQL access to data in relational mode](#)
- [Data source of the EBX5 repository](#)

Single JVM per repository

A repository cannot be shared by multiple JVMs. If such a situation were to occur, it would lead to unpredictable behavior and perhaps even to corruption of data in the repository.

EBX5 performs checks to enforce this restriction. Before the repository becomes available, the repository must first acquire initial exclusive ownership on the relational database. After starting the repository, the JVM periodically checks that it still holds ownership of the repository.

These checks are executed by repeatedly tagging a technical table in the relational database. A shutdown command for the application server ensures that the tag on this technical table is removed. If the server shuts down abruptly, the tag may be left in the table. If this occurs, the server has to wait a few seconds upon restart, to ensure that the table is not being updated by another live process.

Attention

To avoid a wait period at the next start up, it is recommended to always cleanly shut down the application server.

Failover with hot-standby

The exclusion mechanism described above is compatible with failover architectures, where only one server is active at any given time (active/passive cluster). To ensure this is the case, the main server must declare the property `ebx.repository.ownership.type=failovermain`. The main server thus claims the repository database, as in the case of a single server.

A backup server can start, but it will not have access to the repository. It must declare the property `ebx.repository.ownership.type=failoverstandby` to act as the backup server. Once started, the backup server is registered in the connection log. Its status can be retrieved using the Java API or by an HTTP request, as described in the section [Repository status information and logs](#) below.

In order to activate the backup server and transfer exclusive ownership of the repository to it, a specific request must be issued by an HTTP request or using the Java API:

- Using HTTP, the request must include the parameter `activationKeyFromStandbyMode`, and the value of this parameter must be equal to the value declared for the entry `ebx.repository.ownership.activationkey` in the [EBX5 main configuration](#).
- Using the Java API, call the method **wakeFromStandby** [`RepositoryStatus`]^{API}.

The backup server then requests a clean shutdown for the main server, allowing any running transactions to finish. Only after the main server has yielded ownership may the backup use the repository.

Repository status information and logs

A log of all attempted Java process connections to the repository is available in the Administration area under 'History and logs' > 'Repository connection log'.

The status of the repository may be retrieved using the methods in the **RepositoryStatus** [RepositoryStatus]^{API} API.

It is also possible to get repository status information using a HTTP request that includes the parameter `repositoryInformationRequest` with one of following values:

state	<p>The state of the repository regarding the registration of ownership.</p> <ul style="list-style-type: none"> • D: Java process has been stopped. • O: Java process has taken exclusive ownership of the database. • S: Java process has been started in failover standby mode, but is not yet allowed to interact with the repository. • N: Java process has failed to take ownership of the database because another process was holding it.
heart_beat_count	<p>The number of times that the repository has made contact since associating with the database.</p>
info	<p>Detailed information for the end-user about the repository's registration status. The format of this information may be subject to modifications in future without explicit warning.</p>

Repository installation and upgrade

Automatic installation and upgrades

If the specified user for EBX5 data source has the rights to create and alter tables, indexes and sequences in the relational database, then the repository installation or upgrade is done automatically. Otherwise, you must see the procedure in the next section [Manual installation or upgrades](#).

Note

Concerning relational mode and history, declaring a table in relational mode or history mode implies the creation of a dedicated table in the database. Similarly, in these modes, modifying the structure of an existing table implies altering its declaration in the database (note that this action may need to take place at any time, independent of repository creation). The EBX5 applicative user must have 'create/alter' privileges on tables and indexes, since these actions must be carried out from EBX5.

Manual installation or upgrades

If the specified user for EBX5 data source does *not* have the rights to create or alter tables, indexes or sequences in the relational database, then the administrator must execute the SQL scripts located in the `ebx.software/files/ddl/xxx` folder, where `xxx` is the name of the target RDBMS.

If manually upgrading an existing repository, the administrator must execute the required subset of these SQL scripts; the file name of each script indicates the EBX5 version to which the patch applies.

Note:

- If a specific table prefix is specified by the property `ebx.persistence.table.prefix`, the default `EBX_` prefix must be renamed accordingly in the provided scripts.

See also: [Configuring the EBX5 repository](#)

Repository backup

Global backup of EBX5 repository must be delegated to the underlying RDBMS. The database administrator must use the standard backup procedures of the underlying database.

Archives directory

Archives are stored in a sub-directory named `archives` into `ebx.repository.directory` (see [configuration](#)). This directory is automatically created during the first export.

When manually creating this directory, be careful of the access rights: the EBX5 process must have read-and-write access to it. Furthermore, cleaning the directory is not done by EBX5 and is the responsibility of the administrator.

Note

The file transferred between two EBX5 environments must be done externally from the product. Tools such as FTP or simple file copies by network sharing can be used.

Repository attributes

A repository has the following attributes:

- *repositoryId*. It uniquely identifies a repository (at least in the scope of the enterprise): it is 48 bits (6 bytes), and it is usually represented as 12 hexadecimal digits. This information is used for generating UUID (Universally Unique Identifier) of entities created in the repository and also of transactions logged in the history tables or the XML audit trail (it plays the role of the 'UUID node' part, as specified by *RFC 4122*).
- *repository label*. It provides a label for the user so that he knows the purpose and context of the repository (for example, «production environment»).
- *store format*. It identifies the underlying persistence system, including the current version of its structure.

Monitoring and cleanup of relational database

Some entities are accumulated during the execution of EBX5. It is the *administrator's responsibility* to monitor and to cleanup these entities.

General considerations when using data spaces

[Data spaces](#) are an invaluable tool for managing complex data life cycles. While this feature brings great flexibility, it also implies a certain overhead cost, which should be taken into consideration for optimizing usage patterns.

As an extreme example, consider the case where every transaction triggers the following actions: a data space is created, and the transaction modifies some data; the data space is merged, closed, then deleted. In this case, no future references to the data space are needed, so using it to make isolated data modifications is unnecessary. In this case, using **Procedure** [Procedure]^{API} already provides sufficient isolation to avoid conflicts from concurrent operations.

For a developer-friendly analogy, this is like using a source-code management tool (CVS, SVN, etc.): when you need to perform a simple modification impacting only a few files, you probably do it directly on the main branch. In fact, it would be neither practical nor sustainable, with regard to file tagging, if every file modification involved branching the whole project, modifying the files, then merging the dedicated branch.

On larger repositories, creating data spaces can have a significant impact on the size of the repository persisted and the amount of maintenance required. As such, avoid creating data spaces unnecessarily.

Monitoring and reorganization

The persistence data source of the repository must be monitored through RDBMS monitoring.

The EBX5 tables specified in the [default semantic mode](#) have their content persisted in a set of generic database tables. They are the following:

- The table `{ebx.persistence.table.prefix}HOM` in which each record represents a data space or a snapshot (its name is `EBX_HOM`, if the property `ebx.persistence.table.prefix` is unset).
- The table `{ebx.persistence.table.prefix}BRV` where the data of EBX5 tables in semantic mode are segmented into blocks of at most 100 EBX5 records (its name is `EBX_BRV`, if the property `ebx.persistence.table.prefix` is unset).
- The table `{ebx.persistence.table.prefix}HTB`, which defines which blocks belong to a given EBX5 table in a given data space or snapshot (its name is `EBX_HTB`, if the property `ebx.persistence.table.prefix` is unset).

Note

For repositories having large volumes of data in semantic mode and on which frequent or heavy updates occur, it may be necessary to schedule a regular reorganization of the above database tables `...HTB` and `...BRV`, and their indexes. This is especially true for large repositories where many data spaces are created and deleted.

Cleaning up data spaces, snapshots, and history

A full clean-up of data spaces, snapshots, and history from the repository involves several stages:

1. Closing unused data spaces and snapshots to keep the cache to a minimal size.
2. Deleting data spaces, snapshots, and history.
3. Purging the remaining entities associated with the deleted data spaces, snapshots, and history from the repository.

Closing unused data spaces and snapshots

In order to keep the cache and the repository to a reasonable size, it is recommended to close any data spaces and snapshots that are no longer required. This can be done in the following ways:

- Through the user interface, in the Data Spaces area.
- From the 'Data spaces / Snapshots' table under 'Data spaces' in the Administration area, using the **Actions** menu in the workspace. The action can be used on a filtered view of the table.
- Through the Java API, using the method `closeHome [Repository]API`.
- Using the data service close data space and close snapshot operations. See [Closing a data space or snapshot](#) for more information.

Once the data spaces and snapshots have been closed, the data and associated history can be cleaned from the repository.

Note

Closed data spaces and snapshots can be reopened in the Administration area, under 'Data spaces'.

Deleting data spaces, snapshots, and history

Data spaces, their associated history, and snapshots can be permanently deleted from the repository. After you have deleted a data space or snapshot, some entities will remain until a repository-wide purge of all obsolete data is performed. Thus, both stages, the deletion and the repository-wide purge, must be performed in order to completely remove the data and history. This process has been separated into two steps for performance considerations. As the total clean-up of the repository can be time-intensive, this allows the purge execution to be initiated during off-peak periods on the server.

Note

The process of deleting the history of a data space takes into account all history transactions recorded up until the point that the deletion is submitted. Any subsequent historized operations will not be included when you run the purge operation. If you want to delete these new transactions, you must delete the history of that data space again.

The deletion of data spaces, snapshots, and history can be performed in a number of different ways:

- Using the dedicated 'Create deletion requests' interface, accessible from the Administration **Actions** menu in the Administration area. This interface presents all items available for deletion using hierarchical views, categorized into the following entries in the navigation panel:

Closed data spaces	Displays all data spaces and snapshots that are currently closed, and thus can be deleted. You can choose to delete the history associated with data spaces at the same time.
Open data spaces	Displays all data spaces that are currently open, for which you can delete the associated history.
Deleted data spaces	Displays all data spaces that have already been deleted, but have associated history remaining in the repository. You may delete this remaining history.

- From the 'Data spaces / Snapshots' table under 'Data spaces' in the Administration area, using the **Actions** menu in the workspace. The action can be used on a filtered view of the table.
- Using the Java API, specifically, the methods **deleteHome** [Repository]^{API} and **markHomeForHistoryPurge** [RepositoryPurge]^{API}.
- At the end of data service close data space operation, using the parameters `deleteDataOnClose` and `deleteHistoryOnClose`, or at the end of the merge data space operation, using the parameters `deleteDataOnMerge` and `deleteHistoryOnMerge`.

Purging remaining entities after data space, snapshot, and history deletion

Once items have been deleted, a purge can be executed to clean up the remaining data from *all* deletions performed up until that point. A purge can be initiated in the following ways:

- Through the user interface, by selecting Administration **Actions** > **Execute purge** in the navigation pane.
- Using the Java API, specifically the method **purgeAll** [RepositoryPurge]^{API}.
- Using the task scheduler. See [Task scheduler](#) for more information.

The purge process is logged into the directory `${ebx.repository.directory}/db.purge/`.

Cleaning up other repository entities

It is the *administrator's responsibility* to monitor and to regularly cleanup the following entities.

Cleaning up database resources of mapped tables

EBX5 gives the ability to purge mapped tables wherever it detects a mapped table in the database that is no longer used. This means that mapped mode must be deactivated before the mapped table database resources can be purged.

To deactivate mapped mode for a table, follow the procedure below.

For history mode:

- Deactivate historization of the table in the data model, or
- Remove the table from the data model

For relational mode:

- Remove the table from the data model, or
- Deactivate the relational mode on the data model. As data models in semantic mode cannot be used for relational data spaces, it is thus necessary to create a data set on a semantic data space using this modified data model. EBX5 will then detect that relational mode was previously used, and therefore propose the relational table database resources for purge.

Once mapped mode has been deactivated, you can perform a clean-up procedure similar to the process described in [Deleting data spaces, snapshots, and history](#). To select the tables to clean up, open the 'Create deletion requests' interface, accessible from the Administration **Actions** menu in the Administration area. Select 'Database tables' in the navigation pane.

A purge can then be executed to clean up the remaining data from *all* deletions, that is, deleted data spaces, snapshots, history, and database resources, performed up until that point. A purge can be initiated by selecting Administration **Actions** > **Execute purge** in the navigation pane.

Task scheduler execution reports

Task scheduler execution reports are persisted in the 'executions report' table, in the *Task scheduler* section of the Administration area. This table is constantly added to as scheduled tasks are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

User interactions

User interactions are used by the EBX5 component as a reliable means for an application to initiate and get the result of a service execution. They are persisted in the *ebx-interactions* administration feature. It is recommended to regularly monitor the user interactions table, as well as to clean it, if needed.

Workflow history

The workflow events are persisted in the workflow history table, in the *Workflow* section of the Administration area. This table is constantly added to as data workflows are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

Monitoring and cleanup of file system

In order to guarantee the correct operation of the software, the disk usage and disk availability of the following directories must be supervised:

- XML audit trail: `${ebx.repository.directory}/History/`
- archives: `${ebx.repository.directory}/archives/`
- logs: [ebx.logs.directory](#)
- temporary directory: [ebx.temp.directory](#)

Attention

- The cleaning of the above directories is not ensured by EBX5: *this is the administrator's responsibility*.
- For XML audit trail, if large transactions are executed with full updates detail activated (the default setting), *the disk space needed can be quite large*. For more information, see [XML audit Trail](#) section.

See also [configuration / tuning](#) section.

CHAPTER 61

Front end administration

Several administrative tasks can be performed from the *Administration* section of EBX5.

Administration tools

EBX5 offers tools for its administration.

System information

This tool lists information about EBX5 configuration, repository information and operating system information.

Modules and data models

This tool lists all the registered modules as well as the existing data models.

User sessions

This tool lists the user sessions and allows the termination of active sessions.

Data spaces administration

Some data space administrative tasks can be performed from the *Administration* section of EBX5 by selecting the *Data spaces* administration feature.

Data spaces / Snapshots

This table lists all the existing data spaces and snapshots in the repository, whether open or closed. You can view and modify the information of data spaces included in this table.

See also: [Data space information](#)

It is also possible to close open data spaces, reopen closed data spaces, as well as delete and purge open or closed data spaces, associated history, and snapshots.

See also: [Cleaning up data spaces, snapshots, and history](#)

Data space permissions

This table lists all the existing permission rules defined on every data space in the repository. Each permission rule can be accessed and its information modified.

See also: [Permissions](#)

Repository history

The table *Deleted Data space / Snapshot* lists all the data spaces that have been purged from the repository.

Directory administration

See also: [Users and roles directory](#)

Policy

These properties configure the policies of the user and roles directory, for example, whether or not users can edit their own profiles.

Users table

This table lists all the user defined in the internal directory. New users can be added from there.

Roles table

This table lists all the user defined in the internal directory. New roles can be created from there.

Global permissions

Global permission rules can be created in EBX5.

These rules allow the restriction of access to sections of the software by hiding them from the interface. They can be accessed, created and/or modified by selecting the *Global permissions* Data Set.

Profile	Indicates the profile affected by the rule.
Restriction policy	Indicates if the permissions defined here restricts the ones defined for other profiles. See the Restriction policy notion for more information.
Modeling DMA	Defines permissions for accessing data modeling section.
Modeling Workflow	Defines permissions for accessing workflow modeling section.
Data Space	Defines permissions for accessing data space section..
Data Services	Defines permissions for accessing data services section.
Administration	Defines permissions for accessing administration section.

User interface administration

Some options are available to configure the web interface from the *Administration* section then selecting the *User interface* data set in the navigation pane.

Attention

Pay special attention to "URL Policy" configuration! If the web interface configuration results in a non usable application, use the "rescue mode" by setting `frontEnd.rescueMode.enable=true` in [EBX5 main configuration file](#), and using the following URL in your browser, with a built-in administrator user: `http://.../ebx/?onwbpID=iebx-manager-rescue`.

Session configuration

User session default locale	Default session localization.
Session time-out (in seconds)	Maximum time of inactivity of the end-user before he will be considered as inactive and its session will be terminated. A negative time indicates the session should never timeout.

Interface configuration

Entry policy

Describes the URL to access the application.

Login URL	If the user is not authenticated, the session is forwarded to this url.
------------------	---

It defines, for this configuration, EBX5 login page, replacing the default one.

If defined,

- it even replaces the authentication url that may have been defined by the mean of a specific user **Directory** [Directory]^{API},
- it is used to build the permalinks in the user interface,
- if the url is full (starting with http:// or https://), it replaces the url of the workflow email configuration.

URL policy

Describes URL and proxy policy. Both dynamic (servlet) and static (resources) URLs can be configured.

HTTP Servlet Policy	Header Content of Servlet HTTP request: <ul style="list-style-type: none"> • if a field is not set, the default value is chosen (in environment configuration), • if a default value is not set, the value carried by the initial request is chosen.
HTTPS Servlet Policy	Header Content of Servlet HTTPS request: <ul style="list-style-type: none"> • if a field is not set, the default value is chosen (in environment configuration), • if a default value is not set, the value carried by the initial request is chosen.
HTTP External resource Policy	Header Content of External resource URL in HTTP: <ul style="list-style-type: none"> • if a field is not set, the default value is chosen (in environment configuration), • if a default value is not set, the value carried by the initial request is chosen.
HTTPS External resource Policy	Header Content of External resource URL in HTTPS: <ul style="list-style-type: none"> • if a field is not set, the default value is chosen (in environment configuration), • if a default value is not set, the value carried by the initial request is chosen.

Exit policy

Describes how to exit the application.

Normal redirection	Specifies the URL redirection address used when exiting the session normally.
Error redirection	Specifies the URL redirection address used when exiting the session with error.

Graphical interface configuration

Application locking

EBX5 availability status:

Availability status	This application can be closed to users during maintenance (but still remain open to administrators). Takes effect immediately.
Unavailability message	Message displayed to users when access is restricted to administrator profiles.


Security policy

EBX5 access security policy. These parameters only apply on new HTTP sessions.

IP access restriction	Restricts access to designated IP addresses (see IP pattern below).
IP restriction pattern	Regular expression representation of IP addresses authorized to access EBX5. For example, <code>((127\.0\.0\.1) (192\.168\.*\.*))</code> grants access to the local machine and the network IP range <code>192.168.*.*</code> .
Unique session control	Specifies whether EBX5 should control the uniqueness of user sessions. When set to 'Yes', if a user does not log out before closing the browser, it will not be possible for that user to log in again until the previous session has timed out.

Ergonomics and layout

EBX5 ergonomics parameters:

Max table columns to display	According to network and browser performance, adjusts the maximum number of columns to display in a table. This property is not used when a custom view is applied on a table.
Maximum auto-width for table columns	Defines the maximum width to which a table column can auto-size. This is to prevent columns from being too wide, which could occur for very long values, such as URLs. Users will still be able to manually resize columns beyond this value.
Max expanded elements for a hierarchy	Defines the maximum number of elements that can be expanded in a hierarchy when using the action "Expand all". A value less than or equal to '0' disables this parameter.
Default table filter	Defines the default table filter to display in the filters list in tabular views. If modified, users must log out and log in for the new value to take effect.
Display the message box automatically	Defines the message severity threshold for displaying the messages pop-up.
IE compatibility mode	<p>Defines whether or not to compensate for Internet Explorer 8+ displaying EBX5 in compatibility mode.</p> <p>In order to prevent Internet Explorer browsers from using compatibility mode when displaying the repository user interface, the meta-tag <i>http-equiv="X-UA-Compatible" content="IE=EmulateIE8"</i> is added to the header of pages. However, in some local environments, this setting may conflict with existing policies, in which case this header must be omitted by setting the parameter to 'No'. The default value is 'Yes'.</p> <p>See Specifying Document Compatibility Modes  for more information.</p>
Forms: width of labels	The width of labels in forms.
Forms: width of inputs	The width of form input fields in forms.
Forms: width of HTML editor	The width of HTML editors in forms.
Forms: height of HTML Editor	The height of HTML editors in forms.
Forms: height of text areas	The height of text entry fields in forms.

Searchable list selection page size	Number of lines to show per page in searchable list selection dialogs (used for selecting foreign keys, enumerations, etc.).
--	--

Record form: rendering mode for nodes	Specifies how to display non-terminal nodes in record forms. This should be chosen according to network and browser performance. For impact on page loading, link mode is light, expanded and collapsed modes are heavier. If this property is modified, users are required to log out and log in for the new value to take effect.
--	---

Default option values

Defines default values for options in the user interface.

Import / Export

CSV file encoding:	Specifies the character encoding to use for CSV file import and export.
---------------------------	---

Import mode	Specifies the import mode.
--------------------	----------------------------

Missing XML values as 'null'	If 'Yes', when updating existing records, if a node is missing or empty in the imported file, the value is considered as 'null'. If 'No', the value is not modified.
-------------------------------------	--

Colors and themes

Customizes EBX5 colors and themes.

Web site icon URL (favicon)	Sets a custom favicon. The recommended format is ICO, which is compatible with Internet Explorer.
Logo URL (SVG)	Specifies the SVG image used for compatible browsers. Leave this field blank to use the PNG image, if specified. The user interface will attempt to use the specified PNG image if the browser is not SVG-compatible. If no PNG image is specified, the GIF/JPG image will be used. The logo must have a maximum height of 40px. If the height exceeds 40px, it will be cropped, not scaled, to a height of 40px. The width of the logo will determine the position of the buttons in the header.
Logo URL (PNG)	Specifies the PNG image used for compatible browsers. Leave both this field and the SVG image URL field blank to use the GIF/JPG image. The user interface will use the GIF/JPG image if the browser is not PNG-compatible. The logo must have a maximum height of 40px. If the height exceeds 40px, it will be cropped, not scaled, to a height of 40px. The width of the logo will determine the position of the buttons in the header.
Logo URL (GIF/JPG)	Specifies the GIF/JPG image to use when neither the PNG nor SVG are defined. The recommended formats are GIF and JPG. The logo must have a maximum height of 40px. If the height exceeds 40px, it will be cropped, not scaled, to a height of 40px. The width of the logo will determine the position of the buttons in the header.
Main	Main user interface theme color, used for selections and highlights.
Secondary	Secondary user interface theme color, used for some text.
Header	Background color of the user interface header. By default, set to the same value as the Main color.
Header highlighting	Color of accents in the header. By default, set to the same value as the Secondary color.
Workflow badge	Background and text/outline colors of new workflow task counters.
Login page (background)	Background color of the login page. By default, set to a lighter version of the Main color.

Buttons	Colors of buttons and the brightness of button label text and icon.
Button highlighting	Color of buttons highlighting and selected by default. By default, set to the same value as the Main color.
Tabs	Color of tabs in forms.
Tab highlighting	Color of selected tabs. By default, set to the same value as the Main color.
Navigation pane	Background color of the navigation pane.
Form bottom bar	Background color of form bottom bar.
Table history view: technical data	Background color of technical data cells in the table history view.
Table history view: creation	Background color of cells with the state 'creation' in the table history view.
Table history view: deletion	Background color of cells with the state 'deletion' in the table history view.
Table history view: update	Background color of cells with the state 'update' in the table history view.

Lineage administration

Three tables are accessible:

- **Authorized profiles:** a profile must be added to this table to be used for data lineage WSDL generation.
- **History:** lists the data lineage WSDL generated and their configuration.
- **JMS location:** lists the JMS URL locations.

Other technical tables

Several technical tables can be accessed through the *Administration* section of EBX5. These tables are for internal use and/or historizes uses and their content should not be edited, except for removing obsolete or erroneous data.

Auto-increments	This table lists the defined auto-increments field.
Interaction	This table lists the defined interaction.
User preferences	This table lists the preferences of each user.
Custom views	This table lists the custom views defined by users.
User filters	This table lists the filters defined by users.
Default views	This table lists the default view preferences set by users.
Workflows	This table lists the all data workflows and the information associated with each.
Tokens	This table lists all data workflow tokens.
Work items	This table lists all work items and the information associated with each.
History table	This table lists all data workflow history.

CHAPTER 62

Users and roles directory

Overview

EBX5 uses a directory for user authentication and user role definition.

A default directory is provided and integrated into EBX5 repository. It is also possible to integrate with any specific enterprise directory.

See also: [Configuring the user and roles directory](#)

Concepts

In EBX5, a user can participate in several roles, and a role can be shared by several users. Moreover, a role can be included in another role. The generic term "profile" describes either a user or a role.

In addition to the directory-defined roles, EBX5 provides the following *built-in roles*:

Role	Definition
Profile.ADMINISTRATOR	Built-in Administrator role. Administrator role allows performing general administration tasks.
Profile.READ_ONLY	Built-in read-only role. A user associated with role read-only has no rights for doing any modifications on EBX5 repository. He can only visualize the repository.
Profile.OWNER	Dynamic built-in owner role. This role is checked dynamically for the current instance. It is activated only if the user belongs to the profile defined as owner of this current instance.
Profile.EVERYONE	All users have this role.

Information related to profiles is mainly defined in the directory. However, an association between a user and one of the last two roles (*OWNER*, *EVERYONE*) must not be managed by the directory, since EBX5 is assigned to perform this task automatically. User permissions are managed independently of the directory (see chapter [Permissions](#)).

See also:

- [profile](#)
- [role](#)
- [user](#)
- [administrator](#)
- [user and roles directory](#)

Default directory

Directory content

The default directory is represented by the data set named *Directory*, accessible through the 'Administration' area.

This data set contains two tables: one for users and one for roles. By default, only the administrator is allowed to modify the directory. However, users can modify all information related to their own accounts, except for the associated roles.

Note

It is not possible to delete or duplicate the default directory.

Password recovery procedure

In the default directory, passwords are encrypted (by default with a SHA256-like algorithm), and stored in this state. Consequently, it is impossible to retrieve lost passwords. A new password must be generated and sent to the user.

There are two options for this procedure:

1. A notification email is sent to the administrator, the administrator manually changes the password and sends it to the user.
2. A procedure automatically generates a new password and sends it to the user.

By default, the first option is used. For activating the second option, the file [ebx.properties](#) must specify the property:

```
ebx.password.remind.auto=true.
```

Note

For security reasons, the password recovery procedure is not available for administrator profiles. If required, use administrator recovery procedure instead.

Administrator recovery procedure

If an administrator has lost a user name or password, a special procedure must be followed. A specific directory class redefines an administrator user with the login "admin" and the password "admin". To activate this procedure, the file [ebx.properties](#) must specify the following:

```
ebx.directory.factory=\ncom.orchestranetworks.service.directory.DirectoryDefaultRecoverFactory
```

Once EBX5 has been restarted, then stopped, this line must be set back to its default value.

Custom directory

As an alternative to the default directory, it is possible to integrate a specific enterprise directory. For example, an LDAP instance, a relational database or a specific directory model instantiated into EBX5.

For more information, see **DirectoryFactory** [DirectoryFactory]^{API} in the Java API.

CHAPTER 63

Audit Trail

Overview

XML audit trail is a feature that allows logging updates into XML files (an alternative history feature is available, see [History](#) documentation for more details).

Any persistent update performed on the EBX5 repository is logged to an audit trail XML file. This is also the case for any procedure execution (even if it does not perform an update, a procedure is always considered as a transaction). The following information is logged:

- the transaction type (data set creation, record modification, record deletion, specific procedure, etc.);
- the data space or snapshot on which the transaction is executed;
- the transaction source: if the action is initiated by EBX5, this source is described by the user identity, HTTP session identifier and client IP address; if it is initiated programmatically, only the user identity is shown;
- also concerning the session, the optional value "trackingInfo";
- the transaction date and time (in milliseconds);
- the transaction UUID (conforming to Leach-Salz variant, version 1);
- if the transaction has failed, information about the error;
- the detail of updates done (if there are any updates and if history detail is activated, see next section).

Updates detail and disk management

The audit trail is able to describe all updates made into the EBX5 repository, at the finest level. Hence, the files can be quite large and the audit trail directory must be supervised with caution. The following facts should be taken into account:

1. If an archive import is executed in non-interactive mode (without a change set), the audit trail does not detail the updates: it only specifies the archive that has been imported. In this case, if it is important to keep a fine trace of the import-replace, the used archive must be preserved.
2. If an archive import is executed in interactive mode (with change set), or if a data space is merged to its parent, the resulting log size will be nearly equal to the unzipped size of the archive multiplied by three. Furthermore, for consistency issues, each transaction is logged into a temporary file (in the audit trail directory) before being moved into the main file. So EBX5 requires *at least six times the unzipped size of the largest archive that can be imported*.
3. In the context of a specific procedure that performs many updates that need not be audited, it is still possible for the developer to disable the detailed history with the method **ProcedureContext.setHistoryActivation(boolean)** [ProcedureContext]^{API}.

See also: [EBX5 monitoring](#)

Files organization

All audit trail files are stored in the directory `${ebx.repository.directory}/History`.

"Closed" audit files

Each file is named as:

```
aaaa-mm-dd-partnn.xml
```

where *aaaa-mm-dd* is the file date and *nn* is the file index in this day.

Current audit files for writing

When an audit file is being written, the XML structure implies to work in an "open mode". XML elements of the modifications are added to a text file named:

```
aaaa-mm-dd-partnnContent.txt
```

The standard XML format is still available in an XML file that references the text file. This file is named:

```
aaaa-mm-dd-partnnRef.xml
```

Those two files are then re-aggregated in a "closed" XML file when the repository is cleanly shut down or EBX5 is restarted.

Example of an audit directory:

```
2004-04-05-part00.xml
2004-04-05-part01.xml
2004-04-06-part00.xml
2004-04-06-part01.xml
2004-04-06-part02.xml
2004-04-06-part03.xml
2004-04-07-part00.xml
2004-04-10-part00.xml
2004-04-11-part00Content.txt
2004-04-11-part00Ref.xml
```


CHAPTER 64

Data model administration

Administering publications and versions

Technical data about data model publications and versions can be accessed in the *Administration* section by an administrator.

Data Modeling contains the two following tables:

- *Publications*. Stores the publications available in the repository.
- *Versions*. Stores the versions of the data models available in the repository.

These tables are in read-only but it is however possible to delete manually a publication or a version.

Important: If a publication or a version is deleted then the content of associated data sets will become unavailable. So these technical data must be deleted with caution.

It is possible to spread these technical data to other EBX5 repositories exporting an archive from a EBX5 repository and importing it to another one. It may be useful for propagating the evolutions of data models to other repositories.

Migration of previous data models in repository

In versions before 5.2.0, published data models not depending on module were generated in the file system directory `${ebx.repository.directory}/schemas/`, with the name of the data model (*product.xsd* for example if the data model is named *Product*). Since the 5.2.0, this kind of data model is now fully managed inside EBX5 through *Publications*. That is, republishing an existing data model migrates it as a *Publication* and redirects linked data sets to the new embedded data model. The previous XML Schema Document located in `${ebx.repository.directory}/schemas/` is renamed and suffixed with *toDelete* meaning that the document is no more used and can be safely deleted.

CHAPTER 65

Data workflow administration

To define general parameters for data workflow execution, manage workflow publications, and oversee data workflows in progress, navigate to the Administration area. Click the down arrow in the navigation pane and select the entry 'Workflow'.

Execution of workflows and history

Several tables facilitate managing the data workflows currently in progress. They are accessible by navigating to **root > Execution of workflows and history** in the navigation pane.

Workflows table

The 'Workflows' table contains instances of all data workflows in the repository. A data workflow is a particular execution instance of a workflow model publication. This table provides access to the data context variables for all data workflows. It can be used to access the status of advancement of the data workflow in terms of current variable values, and in case of data workflow suspension, to modify the variable values.

Tokens table

The 'Tokens' table allows the management of data workflows advancement. Each token marks the current step of execution of a data workflow in progress, as well as the state of the data workflow.

See also: [token](#)

Work items table

The 'Work items' table contains all the work items associated with user tasks that currently exist. If necessary, you can manually allocate a work item to a user from this table in the case of a blockage in a data workflow through this table. It is preferable, however, to use the buttons in the workspace of the 'Data Workflows' area whenever possible to allocate, reallocate, and deallocate work items.

See also: [work item](#)

History table

The 'History' table contains all actions that have been performed during the execution of workflows. The table is especially useful to view the states of various objects related to workflows at a given moment. This includes actions on work items, variables in the data context, as well as tokens. In case of error, a technical log is available in addition to the contents of the history table.

From the 'Services' menu in the workspace, you can clear the history of completed data workflows or data workflows older than a certain date.

Note

In cases where unexpected inconsistencies have arisen between the workflow execution technical tables, data workflows may encounter errors. It may then be necessary to run the operation 'Clean up inconsistencies in workflow execution tables' from the **Services** menu in the navigation panel under Administration > Workflows.

Interactions

An *interaction* is generated automatically for every work item that is created. It is a local data context of a work item and is accessible from an EBX5 session. When a work item is executed, the user performs the assigned actions based upon its interaction, independent of the workflow engine. User tasks define mappings for their input and output parameters to link interactions with the overall data contexts of data workflows.

Interactions can be useful for monitoring the current parameters of work items. For example, an interaction can be updated manually by a trigger or a UI service.

Workflow publications

The 'Workflow publications' table is a technical table that contains all the workflow model publications in the repository. This table associates published workflow models with their snapshots. It is not recommended to directly modify this table, but rather to use the actions available in the 'Workflow Modeling' area to make changes to publications.

Configuration

The 'Workflow publications' table is a technical table that contains all the workflow model publications in the repository. This table associates published workflow models with their snapshots. It is not recommended to directly modify this table, but rather to use the actions available in the 'Workflow Modeling' area to make changes to publications.

Email configuration

In order for email notifications to be sent during data workflow execution, the following settings must be configured under 'Email configuration':

- 'Activate email notification' must be set to 'Yes'.
- The 'From email' field must be filled in with the email address from to mark as the sender of notification emails.

Priorities configuration

The property 'Default priority' defines how data workflows and their work items across the repository appear if they have not set a priority level. For example, if this property is set to the value 'Normal', any workflow and work item with no priority will appear to have the priority 'Normal'.

The table 'priorities' defines all priority levels available to data workflows in the repository. You may add as many integer priority levels as required, along with their labels, which will appear when users hover over the priority icon in work item tables. You may also select the icons that correspond to each priority level, either from the set provided by EBX5, or by specifying a URL to an icon image file.

Temporal tasks

Under 'Temporal tasks', you can set the polling interval for time-dependent tasks in the workflow, such as deadlines and reminders. If no interval value is set, the steps in progress are checked every hour.

CHAPTER 66

Task scheduler

Overview

EBX5 offers the ability to schedule programmatic tasks.

Note

In order to avoid conflicts and dead-locks, tasks are scheduled in a single queue.

Configuration from EBX5

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the Administration area:

- **Schedules:** define scheduling using "cron expressions".
- **Tasks:** configure tasks, including parameterizing task instances and user profiles their execution.
- **Scheduled tasks:** current schedule, including task scheduling activation/deactivation.
- **Executions report:** reports of each scheduled task run that appear immediately after the task is triggered. The reports include actions to interrupt, pause, or resume running tasks, when made available by the task definition.

Cron expression

(extract from [Quartz Scheduler](#) documentation)

The task scheduler uses "cron expressions", which are able to create firing schedules such as: "At 8:00am every Monday through Friday" or "At 1:30am every last Friday of the month".

Format

A cron expression is a string comprised of 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

Field Name	Mandatory	Allowed Values	Allowed Special Characters
Seconds	Yes	0-59	, - * /
Minutes	Yes	0-59	, - * /
Hours	Yes	0-23	, - * /
Day of month	Yes	0-31	, - * ? / L W
Month	Yes	1-12 or JAN-DEC	, - * /
Day of week	Yes	1-7 or SUN-SAT	, - * ? / L #
Year	No	0-59	empty, 1970-2099

A cron expression can be as simple as this: "*** * * * ? ***",

or more complex, like this: "**0/5 14,18,3-39,52 * ? JAN,MAR,SEP MON-FRI 2002-2010**".

Note

The legal characters and the names of months and days of the week are not case sensitive. MON is the same as mon.

Special characters

A cron expression is a string comprised of 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

- * ("all values") - used to select all values within a field. For example, "*" in the minute field means "every minute".
- ? ("no specific value") - useful when you need to specify something in one of the two fields in which the character is allowed, but not the other. For example, if I want my trigger to fire on a particular day of the month (say, the 10th), but don't care what day of the week that happens to be, I would put "10" in the day-of-month field, and "?" in the day-of-week field. See the examples below for clarification.
- - - used to specify ranges. For example, "10-12" in the hour field means "the hours 10, 11 and 12".
- , - used to specify additional values. For example, "MON,WED,FRI" in the day-of-week field means "the days Monday, Wednesday, and Friday".
- / - used to specify increments. For example, "0/15" in the seconds field means "the seconds 0, 15, 30, and 45". And "5/15" in the seconds field means "the seconds 5, 20, 35, and 50". You can also specify '/' after the " **character - in this case** " is equivalent to having '0' before the '/'. '1/3' in the day-of-month field means "fire every 3 days starting on the first day of the month".
- L ("last") - has different meaning in each of the two fields in which it is allowed. For example, the value "L" in the day-of-month field means "the last day of the month" - day 31 for January, day 28 for February on non-leap years. If used in the day-of-week field by itself, it simply means "7" or "SAT". But if used in the day-of-week field after another value, it means "the last xxx day of the month" - for example "6L" means "the last friday of the month". When using the 'L' option, it is important not to specify lists, or ranges of values, as you'll get confusing results.
- W ("weekday") - used to specify the weekday (Monday-Friday) nearest the given day. As an example, if you were to specify "15W" as the value for the day-of-month field, the meaning is: "the nearest weekday to the 15th of the month". So if the 15th is a Saturday, the trigger will fire on Friday the 14th. If the 15th is a Sunday, the trigger will fire on Monday the 16th. If the 15th is a Tuesday, then it will fire on Tuesday the 15th. However if you specify "1W" as the value for day-of-month, and the 1st is a Saturday, the trigger will fire on Monday the 3rd, as it will not 'jump' over the boundary of a month's days. The 'W' character can only be specified when the day-of-month is a single day, not a range or list of days.

Note

The 'L' and 'W' characters can also be combined in the day-of-month field to yield 'LW', which translates to "*"last weekday of the month"*.

- # - used to specify "the nth" XXX day of the month. For example, the value of "6#3" in the day-of-week field means "the third Friday of the month" (day 6 = Friday and "#3" = the 3rd one in the month). Other examples: "2#1" = the first Monday of the month and "4#5" = the fifth Wednesday of the month. Note that if you specify "#5" and there is not 5 of the given day-of-week in the month, then no firing will occur that month.

Examples

Expression	Meaning
0 0 12 * * ?	Fire at 12pm (noon) every day.
0 15 10 ? * *	Fire at 10:15am every day.
0 15 10 * * ?	Fire at 10:15am every day.
0 15 10 * * ? *	Fire at 10:15am every day.
0 15 10 * * ? 2005	Fire at 10:15am every day during the year 2005.
0 * 14 * * ?	Fire every minute starting at 2pm and ending at 2:59pm, every day.
0 0/5 14 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day.
0 0/5 14,18 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at 6pm and ending at 6:55pm, every day.
0 0-5 14 * * ?	Fire every minute starting at 2pm and ending at 2:05pm, every day.
0 10,44 14 ? 3 WED	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
0 15 10 ? * MON-FRI	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday.
0 15 10 15 * ?	Fire at 10:15am on the 15th day of every month.
0 15 10 L * ?	Fire at 10:15am on the last day of every month.
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month.
0 15 10 ? * 6L 2002-2005	Fire at 10:15am on every last friday of every month during the years 2002, 2003, 2004 and 2005.
0 15 10 ? * 6#3	Fire at 10:15am on the third Friday of every month.
0 0 12 1/5 * ?	Fire at 12pm (noon) every 5 days every month, starting on the first day of the month.
0 11 11 11 11 ?	Fire every November 11th at 11:11am.

Note

Pay attention to the effects of '?' and '*' in the day-of-week and day-of-month fields!

Support for specifying both a day-of-week and a day-of-month value is not complete (you must currently use the '?' character in one of these fields).

Be careful when setting fire times between the hours of the morning when "daylight savings" changes occur in your locale (for US locales, this would typically be the hour before and after 2:00 AM - because the time shift can cause a skip or a repeat depending on whether the time moves back or jumps forward).

Task definition

EBX5 scheduler comes with a predefined task: "Repository clean-up".

Custom scheduled tasks can be added by the means of **scheduler** [package-summary]^{API} Java API.

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the Administration area.

Task configuration

A user must be associated to a task definition; this user will be used to generate the **session** [Session]^{API} that will run the task.

Note

The user will not be authenticated, and no password is required. As a consequence, a user with no password set in the directory can only be used to run scheduled tasks.

A custom task can be parameterized by means of JavaBean specification (getter and setter).

Supported parameter types are:

- java.lang.boolean
- java.lang.int
- java.lang.Boolean
- java.lang.Integer
- java.math.BigDecimal
- java.lang.String
- java.lang.Date
- java.net.URI
- java.net.URL

Parameter values are set in XML format.

Distributed Data Delivery (D3)

CHAPTER 67

Introduction to D3

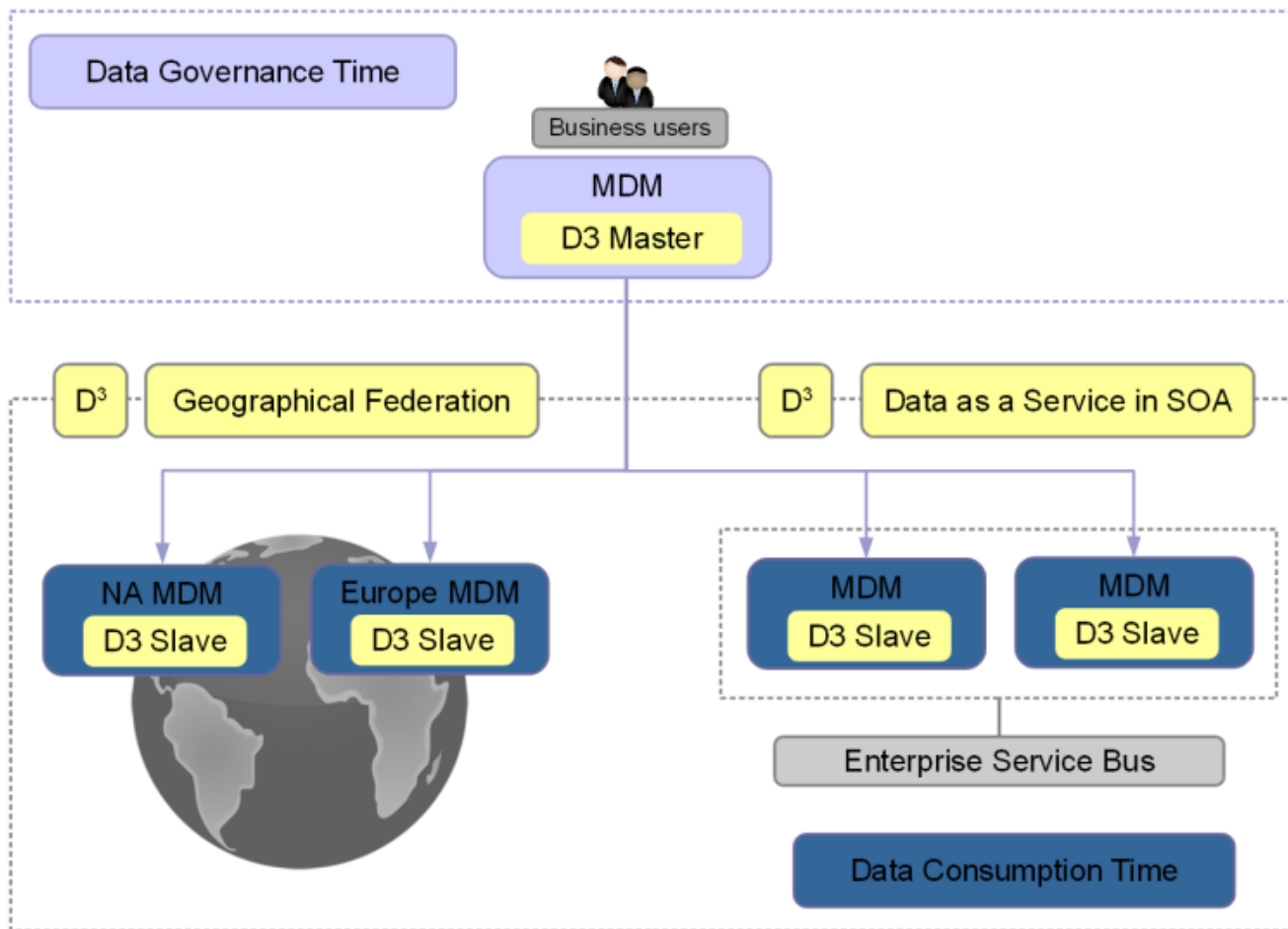
Overview

EBX5 offers the ability to replicate data from one EBX5 instance to several others. By means of the broadcast service, it also provides an additional layer of security and control to the other features of EBX5: it is particularly adapted when data governance requires the highest levels of data consistency, approval and/or capacity of rollback.

Architecture

A typical D3 installation consists of one master and several slaves. In the master, the Data Steward declares which data spaces must be replicated, and which user profile is allowed to broadcast them to the slaves. The Data Steward also defines delivery profiles, which are groups of one or more data spaces.

Each slave must define which delivery profile has to be replicated towards it.



Involvement of third-party systems

The features of D3 also allow access by third-party systems to data managed in EBX5 through Data Services. Essentially, when a system consumes the data of a delivery data space, the data is transparently redirected to the last Snapshot that has been broadcast. This ensures a more controlled and reliable view of the managed data.

Third-party systems can either access the master node or a slave node (hence a physical architecture involving a master node and no slave node is also an option).

Protocols

If JMS is activated, conversation between a master and a slave is based on SOAP over JMS, while archive transfer between a master and a slave is based on JMS binary messages.

If JMS is not activated, conversation between a master and a slave is based on SOAP over HTTP, while archive transfer (binary) between a master and a slave is based on TCP sockets.

See also: [JMS for distributed data delivery \(D3\)](#)

Glossary

Broadcast

The action broadcast is a way to "publish an official snapshot" of data. Underneath, it transparently and transactionally ensures that it is replicated to the slave nodes and becomes "the official truth" for all repositories (master and slaves).

Delivery data space

A delivery data space is a data space which can be broadcast to authenticated and allowed users, using a dedicated service.

By default, when a data service accesses a delivery data space (on any node), it is redirected to the last snapshot that has been broadcast. More information in the [Data Services](#) section.

Delivery profile

A delivery profile is a logical name that groups one or several delivery data spaces. The slaves subscribe to one or several delivery profiles.

Cluster delivery mode

Synchronization with subscribed slave nodes is performed in a two-phase commit transactional process. This delivery mode is designed to respond to a high volume of queries using load balancing and/or fault tolerance. It ensures the consistency of data in the cluster between slave nodes and their master delivery data spaces. Master and slave nodes use the same committed snapshots.

Federation delivery mode

Synchronization is performed in a single phase, and with each registered slave node independently. This delivery mode is designed to be used with geographically distributed and/or heterogeneous architectures where response time and network availability cannot be guaranteed. At any one time, slaves can be at different committed snapshot versions. The synchronization processes are thus independent of one another and replay individual slave nodes are performed for certain broadcast failures.

Master node

The master node is an instance of EBX5 that can define one or several delivery data spaces, and to which slave nodes can subscribe. Otherwise, the master node can also act as a regular EBX5 server.

Slave node

A slave node is an instance of EBX5 attached to a master node, in order to receive delivery data spaces replication. Except for update restrictions on the delivery/replicated data spaces, the slave node acts as a regular EBX5 server.

Hub node

A hub node is an instance of EBX5 acting as a master node and as a slave node.

Known limitations

General limitations

- Each slave node must have only one master.
- Embedded data models cannot be used in D3 data spaces. Therefore, it is not possible to create a data set based on a publication in a D3 data space.

Broadcast and delivery data space limitations

- Access rights on data spaces are not replicated (whereas access rights on data sets are).
- Data space information is not replicated.
- If a data space and its parent are replicated, this parent-child relationship will be lost in the slaves.
- If a data space is deleted from the master, it remains on the slaves.
- Re-broadcasting a snapshot where inherited data sets have been added or removed is not supported.
- Once a snapshot has been broadcast to a slave, subsequent broadcasts of *any* snapshot with the same name will result in restoring the originally broadcast version of that same name on the slave. That is, if the original snapshot on the master is purged and a new one is created with the same name and subsequently broadcast, then the content of the slave will be restored to that of the previously broadcast snapshot, and not the latest one of the same name.
- The data model on which the broadcast contents are based must be the same between the broadcast and the current version.

Administration limitations

- Technical data spaces cannot be replicated, thus the EBX5 default user directory cannot be synchronized through D3.

CHAPTER 68

D3 broadcasts and delivery data spaces

Broadcast procedure

Scope and contents of a broadcast

A D3 broadcast occurs at the data space or snapshot level. For data space broadcasts, D3 first creates a snapshot to capture the current state, then broadcasts this newly created snapshot.

A broadcast performs one of the following procedures depending on the situation:

- An update of the differences computed between the new snapshot broadcast and the current 'commit' version on the slave.
- A full synchronization containing all data sets, tables, records, and permissions. This is done on the first broadcast to a given slave node, or if the previous slave commit is not known to the master node.

Performing a broadcast

The broadcast can be done:

- By the end user, using the action **Broadcast** available in the delivery data space or a snapshot of the delivery data space,
- Using a custom code that uses **D3Engine** [D3Engine]^{API} in the Java API.

Conditions

To be able to broadcast, the following conditions must be fulfilled:

- The authenticated user profile is granted permission to broadcast.
- The data space or snapshot to be broadcast has no validation errors.
- The D3 master configuration has no validation errors.
- The data space or snapshot must not contain any tables in relational mode.
- There must be an associated delivery profile.
- If broadcasting a data space, the data space must not be locked.
- If broadcasting a snapshot, the snapshot must belong to a data space declared as delivery data space and must not already be the current broadcast snapshot (but a rollback to a previously broadcast snapshot is possible).

Note

Broadcasts are performed asynchronously. Therefore, no information is displayed in the user interface about the success or failure of a broadcast. Nevertheless, it is possible to monitor the slaves and the deliveries. See [D3 supervision](#) for more information.

Delivery data spaces

Data Services

By default, when a data service accesses a delivery data space, it is redirected to the current snapshot, which is the last one broadcast. However, this default behavior can be modified either at the request level or the global configuration level.

See also: [Common parameter 'disableRedirectionToLastBroadcast'](#)

Access restrictions

On the master node, a delivery data space can neither be merged nor closed. Other operations are available, for example, according to permissions, it can be modified directly, snapshot can be created independently of a broadcast, or a child data space can be created and merged.

On the slave node, no modification can be made on a delivery/replicated data space, by any means (end user, data services, Java program) except broadcast process. Furthermore, any data space-related operation (such as merge, close, etc) is forbidden on the slaves.

Transactional view

The current snapshot may change between two calls if a broadcast has been made meanwhile. If a fully stable view is required for several successive calls, these calls will need to directly specify the same expected Snapshot.

CHAPTER 69

D3 administration

Configuring D3 nodes

Runtime configuration of master or hub nodes through the user interface

The declaration of delivery data spaces and delivery profiles is done by selecting the '[D3] Master configuration' feature from the Administration area, where you will find the following tables:

Delivery data spaces	Declarations of the data spaces that can be broadcast.
Delivery profiles	Profiles to which slaves nodes can subscribe. The delivery mode must be defined for each delivery profile.
Delivery mapping	The association between delivery data spaces and delivery profiles.

Configuring master, hub and slave nodes

This section details how to configure a node in its EBX5 main configuration file.

See also: [Overview of the EBX5 main configuration file](#)

Master

In order to act as a *master* node, an instance of EBX5 must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=master` node:

```
#####
## D3 configuration
#####
#####
# Configuration for both master and slave
#####
# Optional property.
```

```
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=master
```

Hub

In order to act as a *hub* node (combination of master and slave configurations), an instance of EBX5 must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=hub` node:

```
#####
## D3 configuration
#####
#####
# Configuration for both master and slave
#####
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=hub

#####
# Configuration dedicated to hub or slave
#####
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

Slave

In order to act as a *slave* node, an instance of EBX5 must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=slave` node:

```
#####
## D3 configuration
#####
#####
# Configuration for both master and slave
#####
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=slave
```



```
#####
# Configuration dedicated to hub or slave
#####
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

Configuring the network protocol of a node

This section details how to configure the network protocol of a node in its EBX5 main configuration file.

See also: [Overview of the EBX5 main configuration file](#)

HTTP and socket TCP protocols

Sample configuration for ebx.d3.mode=hub or ebx.d3.mode=slave node with HTTP network protocol:

```
#####
# HTTP and TCP socket configuration for D3 slaves
#####
# URL to access the data services connector of the master
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not
  activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: http://[master_host]:[master_port]/ebx-dataservices/
  connector
ebx.d3.master.url=

# URL to access the data services connector of the slave
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not
  activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: http://[slave_host]:[slave_port]/ebx-dataservices/
  connector
ebx.d3.slave.url=

# Minimum port to use to transfer archives on TCP mode.
# Must be a positive integer above zero and below 65535.
# If not set, a random port will be used.
#ebx.d3.slave.socket.range.min=

# Max port to use on TCP mode to transfer archives.
# Must be a positive integer above ebx.d3.slave.socket.range.min and below 65535.
# Mandatory if ebx.d3.slave.socket.range.min is set.
#ebx.d3.slave.socket.range.max=
```

JMS protocol

If JMS is activated, the following properties can be defined in order to enable JMS functionalities for a D3 node.

Sample configuration for all D3 nodes with JMS network protocol:

```
#####
## JMS configuration for D3
#####
# Taken into account only if Data Services JMS is configured properly
#####
# Configuration for master, hub and slave
#####
# Default is false, activate JMS for D3
## If activated, the deployer must ensure that the entry 'jms/EBX_D3ReplyQueue'
## are bound in the operational environment of the application server.
ebx.jms.d3.activate=false

# Change default timeout when use reply queue, default is 10000 milliseconds
#ebx.jms.d3.reply.timeout=10000
```

See also: [JMS for distributed data delivery \(D3\)](#)

Services on master nodes

Services to manage a master node are available in the Administration area of the slave node under '[D3] Master configuration' and also on the tables 'Delivery data spaces' and 'Registered slaves'. The services are as follows:

Relaunch replays	Immediately relaunch all replays for waiting federation deliveries.
Delete data space	Delete the delivery branch on registered slaves and remove it from the configuration of the D3 master.
Launch slave subscription	Send a request to the selected slaves to subscribe them.
Deactivate slaves	Remove the selected slaves from the broadcast scope and switch their states to 'Unavailable'.
Unregister slaves	Completely remove the selected slaves from the master.

Services on slave nodes

Services to manage a slave node's subscription to the master node are available in the Administration area of the slave node under '[D3] Slave configuration', in the navigation pane. The services are as follows:

Register slave	Re-subscribes the slave to the master if it has been unregistered.
Unregister slave	Disconnects the slave from the master.

Supervision

Master node management console

Several tables compose the management console for the master node, located in the Administration area of the master node, under '[D3] Master configuration'. They are as follows:

Registered slaves	Slaves registered with the master node. From this table, several services are available on each record.
Broadcast history	History of broadcast operations that have taken place.
Slave registration log	History of initialization operations that have taken place.
Detailed history	History of archive deliveries that have taken place. The list of associated delivery archives can be accessed from the tables 'Broadcast history' and 'Initialization history' using selection nodes.

Master node supervision services

Available in the administration area of the master node under '[D3] Master configuration'. The services are as follows:

Display and refresh state of slaves	Lists the slaves and related information, such as the slave's state, associated delivery profiles, and delivered snapshots.
Clear history content	Deletes all records in all history tables, such as 'Broadcast history', 'Slave registration log' and 'Detailed history'.

Slave monitoring through the Java API

A slave monitoring class can be created to implement actions that are triggered when the slave's status switches to either 'Available' or 'Unavailable'. To do so, it must implement the interface

`NodeMonitoring`. This class must be outside of any EBX5 module and accessible from the class-loader of 'ebx.jar' and its full class name must be specified under '[D3] Slave configuration'.

See also: ***NodeMonitoring*** [*NodeMonitoring*]^{API}

Log supervision

The technical supervision can be done through the log category 'ebx.d3', declared in the EBX5 main configuration file. For example:

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

See also: [*Configuring the EBX5 logs*](#)

Developer Guide

CHAPTER 70

Notes to developers

Vocabulary evolution

EBX5 introduced a new vocabulary for users (see the [Glossary](#)) but this new vocabulary has not impacted the API. Hence the Java classes, data services WSDL and EBX5 components are still using the old vocabulary.

The typography rules applied below are representative of the ones applied in the documentation, but not of those used in the user interface.

The following table summarizes those evolutions:

Vocabulary in EBX5	Vocabulary prior to version 5
Data set	Adaptation instance
Child data set	Child adaptation instance
Data model	Data model
Data space	Branch
Snapshot	Version
Data space or snapshot	Home
Data Workflow	Workflow instance
Workflow model	Workflow definition
Workflow publication	Workflow
Data services	Data services
Field	Attribute
Inherited field	Inherited attribute
Record	Record/occurrence
Validation rule	Constraint
Simple/advanced control	Simple/advanced constraint

Model design

CHAPTER 71

Introduction

A data model is a structural definition of the data to be managed in the EBX5 repository. Globally, the goal is to guarantee the highest level of data consistency and to facilitate their management.

In concrete terms, the data model is a document that conforms to the XML Schema standard (W3C recommendation). The main standard features that are supported are the following:

- A rich library of well-defined basic [data types](#) (*simple types*: integer, boolean, decimal, date, time, ...);
- The ability to build more [simple types](#) (*simple types*); and [complex structures](#) (*complex types*);
- The ability to define simple lists of items ([aggregated lists](#));
- [Validation constraints](#) (*facets*): enumerations, uniqueness, minimum/maximum boundaries, ...

EBX5 also uses the extensibility features of XML Schema for other useful information, such as:

- Useful [predefined types](#) (locale, resource, html, ...);
- Definition of [tables](#) and [foreign key constraints](#) ;
- Mapping of data in EBX5 to Java beans;
- [Advanced validation constraints](#) (extended *facets*), like dynamic enumerations;
- Extensive [presentation information](#): label, description, error messages, ...

Note: EBX5 supports a subset of the W3C recommendation, some features actually being useless for Master Data.

Model Editor

The data model can be defined by using an XML Schema editor or the *Data Model Assistant* . The latter has the advantage of being integrated into EBX5 user interface and hiding the XML verbose language.

References

For an introduction to XML Schema, we recommend the [W3School](#) web site.

W3C specification documents: XML Schema [Part 0: Primer](#) , [Part 1: Structures](#) , [Part 2: Datatypes](#) .

Links between data sets and data models

Any root data set is associated with a single data model.

In order to create a root data set, you must have the rights granted. Being logged in EBX5, you have to position to a data space conten (click on the link "View or edit content"). On the left side of the screen, click on the "create..." link. Several creation modes are then available.

Pre-requisite for XML Schemas

In order to be accepted by EBX5, an XML Schema must include a global element declaration which has an attribute `osd:access="--"` .

```
<?xml version="1.0" encoding="UTF-8"?>
<!-->
<!-- {ebx.copyright.text} -->
<!-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:import namespace="urn:ebx-schemas:common_1.0"
    schemaLocation="http://schema.orchestranetworks.com/common_1.0.xsd"/>
  <xs:element name="root" osd:access="--">
    ...
  </xs:element>
</xs:schema>
```

Conventions

By convention, namespaces are always defined as below:

Prefix	Namespace
xs:	http://www.w3.org/2001/XMLSchema
osd:	urn:ebx-schemas:common_1.0
fmt:	urn:ebx-schemas:format_1.0

Schemas with reserved names

Several data model have reserved names in EBX5.

All references to other data models (attribute `schemaLocation` an import, includes or redefines) that end with one of the following strings are reserved:

- `common_1.0.xsd`
- `org_1.0.xsd`
- `coreModel_1.0.xsd`
- `session_1.0.xsd`

Those files correspond to the schemas provided for the module `ebx-root-1.0`, path `/WEB-INF/ebx/schemas`. The attribute `schemaLocation` can reference those files at this location or reference a copy, if the file name is the same. This is useful if you want to avoid the module dependency to `ebx-root-1.0`.

For security reasons, EBX5 uses an internal definition of those schemas (to avoid any modification).

CHAPTER 72

Packaging EBX5 modules

An EBX5 module allows the packaging of a data model with its resources: included XML Schema Documents, Java classes, etc.

On a Java EE application server, a module in the EBX5 repository is equivalent to a standard Java EE web application. This provides features such as class-loading isolation, WAR or EAR packaging, web resources exposure, hot-redeployment. In addition, if your user application is a web application, it is possible to merge the EBX5 module with your application, in order to simplify the deployment.

Structure

An EBX5 module contains the following files:

1. /WEB-INF/web.xml:

This is the standard Java EE deployment descriptor. It must ensure that the EBX5 module is registered when the application server is launched (see [Registration](#) section).

2. /WEB-INF/ebx/module.xml:

This mandatory document defines the main properties and services of the module.

3. /www/:

This optional directory contains all external resources (accessible by a public URL). This directory is optional, localized and structured by resource type (html, images, jscripts, stylesheet). External resources in this directory can be referenced in data models (type `osd:resource`).

Declaration

A module is declared by means of the document /WEB-INF/ebx/module.xml. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:ebx-schemas:module_2.3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ebx-schemas:module_2.3 http://schema.orchestranetworks.com/
module_2.3.xsd">
  <name>moduleTest</name>
```

```

<locales>
  <locale isDefault="true">it</locale>
  <locale>en-US</locale>
</locales>
</module>

```

The [associated schema](#) provides a documentation on each property. Here is a summary of the main ones:

Element	Description	Required
name	Defines the unique identifier of the module on the server instance. The module name usually corresponds to the web application's name (the name of the directory).	Yes.
publicPath	If a specific path (different from the above module's name) identifies the web application in the public URLs, defines this path. This path is added to the URL of an external resource of the module, in the case of absolute URL computing. If this field is not present, the public path is the module's <code>name</code> defined above.	No.
locales	Defines the locales supported by the data model(s) of the module. This list must contain all the locales that can be found in the schemas within the module, and that we want to expose to the end user (EBX5 will not be able to display labels and messages in a language that is not declared in this list). If the element is not present, the module supports the locales of EBX5.	No.
services	Declares the UI services. For more information, see UI services section .	No.
beans	Declares the reusable Java bean components. For more information, see workflow package [package-summary] ^{API} .	No.
ajaxComponents	Declares the Ajax components. For more information, see the section "Declaration in a module" in UIAjaxComponent [UIAjaxComponent] ^{API} in the Java API.	No.

Registration

In order to be identified by EBX5, a module must be registered at runtime, when the application server is launched. For a web application, every EBX5 module must:

1. contain a Servlet whose standard method `init` invokes `ModulesRegister.registerwebApp(...)`, see Java code example at the end of this section;
2. declare this servlet in the deployment descriptor `/WEB-INF/web.xml`, in the standard way;
3. ensure that this servlet will be launched at server startup by adding the following standard element to the deployment descriptor: `<load-on-startup>1</load-on-startup>`.

Example of a Java EE deployment descriptor (`/WEB-INF/web.xml` file):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>InitEbxServlet</servlet-name>
    <servlet-class>com.foo.RegisterServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
</web-app>
```

Notes:

- If Java classes are located in the web application (in `/WEB-INF/classes` or `/WEB-INF/lib`) and these classes are used as resources of the schemas in the module, it is mandatory that the specific servlet class also be located in the web application (the reason is that the servlet class is internally used as a hook to the application's class-loader).
- It is recommended that the servlet also implements the method `destroy()`, otherwise the "stop" of the web application will not work.

Once the method `ModulesRegister.unregisterWebApp()` has been executed, the schemas and the adaptations that depend on it become unavailable.

EBX5 supports hot-deployment and hot-redeployment operations that are implemented by Java EE application servers: deployment-start, stop-restart, stop-redeployment-restart, stop of a web application. However, these operations remain sensitive, more particularly concerning class-loading and potential complex dependencies, consequently these operations must be followed carefully.

- All modules' registrations and unregistrations are logged to 'log.kernel' category.
- If an exception occurs while loading a module, the cause of the error is written in the application server log.

Java code example of a servlet that registers/ unregisters the module to/from EBX5:

```
package com.foo;
import javax.servlet.*;
import javax.servlet.http.*;
import com.onwbp.base.repository.*;
/**
 * /
public class RegisterServlet extends HttpServlet
{
    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
        ModulesRegister.registerWebApp(this, config);
    }
    public void destroy()
    {
        ModulesRegister.unregisterWebApp(this, this.getServletConfig());
    }
}
```

See also: [Module registration](#)

CHAPTER 73

Types

This page presents the data types supported by EBX5. For the definition of tables, read the section [Tables & Filters](#).

XML Schema built-in simple types

The table below lists all simple types defined in XML Schema and supported by EBX5:

WXS type		Java class
string	primitive datatype	java.lang.String
boolean	primitive datatype	java.lang.Boolean
decimal	primitive datatype	java.math.BigDecimal
dateTime	primitive datatype	java.util.Date
time	primitive datatype	java.util.Date
date	primitive datatype	java.util.Date
anyURI	primitive datatype	java.net.URI
Name	string restriction	java.lang.String
int	decimal restriction	java.lang.Integer

The rightmost column shows the Java class that is instantiated in for each XML Schema type in EBX5. Correspondence rules between XML Schema types and Java types are detailed in the section [Mapping to Java](#).

XML Schema named simple types

Named simple types can be defined when designing a data model for redefining an existing built-in simple type. A *named simple type* can be reused in the data model.

Restrictions:

- In term of content model, only the tag `<restriction>` is allowed in a named simple type (in particular, tags `<list>` and `<union>` are not supported), then only derivation by restriction is supported.
- Facets definition is not cumulative. That is, if an element and its named type both define the same kind of facet then the one defined in the type is overwritten by the local definition. This restriction is however not performed on programmatic facets defined by the tag `osd:constraint`. In this case, if an element and its named type both define a programmatic facet with distinct Java classes then the definition of these facets will be cumulative. Contrary to the XML Schema Specification, EBX5 is not strict regarding to the definition of a facet of the same kind in an element and its named type. That is, the value of a same kind of facet defined in an element is not checked according to the one defined in the named type. However, in the case of static enumerations defined both in an element and its type the local enumeration will be replaced by the intersection between these enumerations.
- It is forbidden to define different kind of enumerations on both an element and its named type. For instance, it is forbidden to specify a static enumeration in an element and a dynamic enumeration in its named type.
- It is forbidden to define simultaneously a pattern facet in both an element and its named type.

XML Schema complex types

Complex types can be defined when designing a data model. A *named complex type* can be reused in the data model.

Restrictions:

- In term of content model, only the tag `<sequence>` is allowed (in particular, the attribute definition is not supported)
- Type extension is not supported in the current version of EBX5

Extended simple types defined by EBX5

EBX5 provides pre-defined simple data types:

WXS type		Java class
osd:text	string restriction	java.lang.String
osd:html	string restriction	java.lang.String
osd:email	string restriction	java.lang.String
osd:password	string restriction	java.lang.String
osd:resource	anyURI restriction	internal class
osd:locale	string restriction	java.util.Locale

Those types are defined by the reserved schema *common-1.0.xsd*. They are defined as follows:

- *osd:text*

This type represents a textual information. Regarding a basic `xs:string`, its default user interface in EBX5 consists of a dedicated editor with several lines for input and display.

```
<xs:simpleType name="text">
  <xs:restriction base="xs:string" />
</xs:simpleType>
```

- *osd:html*

This represents a character string with HTML format. A "wysiwyg" editor is provided in EBX5 to edit it.

```
<xs:simpleType name="html">
  <xs:restriction base="xs:string" />
</xs:simpleType>
```

- *osd:email*

This represents an email address as specified in RFC822.

```
<xs:simpleType name="email">
  <xs:restriction base="xs:string" />
</xs:simpleType>
```

- *osd:password*

This represents an encrypted password. A specific editor is provided in EBX5 to edit it.

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

The default editor performs an encryption using the SHA-256 algorithm (this encryption function is available from a Java client by means of the method **DirectoryDefault.encryptString** [DirectoryDefault]^{API}).

It is also possible that the default editor uses a different encryption by specifying a class implementing the interface **Encryption** [Encryption]^{API}:

```
<xs:element name="password" type="osd:password">
  <xs:annotation>
    <xs:appinfo>
      <osd:uiBean class="com.orchestranetworks.ui.UIPassword">
        <encryptionClass>package.EncryptionClassName</encryptionClass>
      </osd:uiBean>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

- *osd:resource*

This represents a resource in a module for EBX5. A resource is a file of type image, HTML, CSS or JavaScript, stored into a module within EBX5. It requires the definition of the facet [FacetOResource](#) .

```
<xs:simpleType name="resource">
  <xs:restriction base="xs:anyURI" />
</xs:simpleType>
```

- *osd:locale*

This represents a geographical, political or cultural specific location. The locale type is translated into Java by the class `java.util.Locale` .

```
<xs:simpleType name="locale">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ar" osd:label="Arabic" />
    <xs:enumeration value="ar_AE" osd:label="Arabic (United Arab Emirates)" />
    <xs:enumeration value="ar_BH" osd:label="Arabic (Bahrain)" />
    <xs:enumeration value="ar_DZ" osd:label="Arabic (Algeria)" />
    <xs:enumeration value="ar_EG" osd:label="Arabic (Egypt)" />
    <xs:enumeration value="ar_IQ" osd:label="Arabic (Iraq)" />
    ...
    <xs:enumeration value="vi_VN" osd:label="Vietnamese (Vietnam)" />
    <xs:enumeration value="zh" osd:label="Chinese" />
    <xs:enumeration value="zh_CN" osd:label="Chinese (China)" />
    <xs:enumeration value="zh_HK" osd:label="Chinese (Hong Kong)" />
    <xs:enumeration value="zh_TW" osd:label="Chinese (Taiwan)" />
  </xs:restriction>
</xs:simpleType>
```

Complex types defined by EBX5

EBX5 provides pre-defined complex data types:

WXS type	Description
UDA	User Defined Attribute: this type allows any user, according to his access rights, to define a value associated with an attribute defined in a dictionary called a UDA Catalog.
UDACatalog	Catalog of User Defined Attributes: this type consists of a table in which attributes can be specified. This catalog will be used by all <code>osd:UDA</code> declared in the same data model.

- *UDA*

A User Defined Attribute (UDA) supports both `minOccurs` and `maxOccurs` parameters and `osd:UDACatalogPath` to specify the path of the matching catalog.

```
<xs:element name="firstUDA" type="osd:UDA" minOccurs="0"
  maxOccurs="unbounded" osd:UDACatalogPath="//insuranceCatalog" />
<xs:element name="secondUDA" type="osd:UDA" minOccurs="1" maxOccurs="1"
  osd:UDACatalogPath="/root/userCatalog" />
<xs:element name="thirdUDA" type="osd:UDA" minOccurs="0" maxOccurs="1"
  osd:UDACatalogPath="//userCatalog" />
```

In the manager, when working with an UDA, the editor will adapt itself to the type of the selected attribute.

- *UDACatalog*

A catalog is a table. The parameters `minOccurs` and `maxOccurs` have to be specified.

Several catalogs can be defined in one data model.

```
<xs:element name="insuranceCatalog" type="osd:UDACatalog" minOccurs="0"
  maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation xml:lang="en-US">Insurance Catalog.</xs:documentation>
    <xs:documentation xml:lang="fr-FR">Catalog assurance.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="userCatalog" type="osd:UDACatalog" minOccurs="0"
  maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation xml:lang="en-US">User catalog.</xs:documentation>
    <xs:documentation xml:lang="fr-FR">Catalogue utilisateur.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Only the following types are available for creating new attributes:

- `xs:string`

- `xs:boolean`
- `xs:decimal`
- `xs:dateTime`
- `xs:time`
- `xs:date`
- `xs:anyURI`
- `xs:Name`
- `xs:int`
- `osd:html`
- `osd:email`
- `osd:password`
- `osd:locale`
- `osd:text`

Restrictions on User Defined Attributes and Catalogs

The following are unsupported on UDA elements:

- Facets
- Functions using the `osd:function` property
- UIBean editors using the `osd:uiBean` property
- `osd:checkNullInput` property
- Historization features
- Inheritance features using the `osd:inheritance` property

As UDA catalogs are considered to be tables, the restrictions that apply to tables also exist for `UDACatalog` elements.

Aggregated lists

In XML Schema, the maximum number of times an element may appear is determined by the value of a `maxOccurs` attribute in its declaration. If this value is strictly greater than 1 or is equal to `unbounded`, then the data can have multiple occurrences. If no `osd:table` declaration is added, this case is called *aggregated lists*. In Java, it is represented as an instance of class `java.util.List`.

Important note: the addition of a `osd:table` declaration to an element with `maxOccurs > 1` is a very important consideration that must be taken into account during the design process. An aggregated list is indeed severely limited regarding the many features that are supported by tables. As a reminder, here are some of them: lookups, filters and searches; sorting, custom view and display in hierarchies; identity constraints (primary keys and uniqueness constraints); detailed permissions for creation, modification, delete and particular permissions at record level; detailed comparison and merge; and last but not least performance and memory optimizations. Hence *aggregated lists should be used only for small volumes of simple data (one or two dozen of occurrences), with no advanced requirements* . For larger volumes of data or more advanced functionalities, it is strongly advised to use `osd:table` declarations.

For more information on table declarations, read the [chapter about tables](#) .

Below is an example of an aggregated list that defines the pricing of a loan product, depending on the amount borrowed.

```
<xs:element name="pricing" minOccurs="0" maxOccurs="unbounded"
  osd:access="RW">
  <xs:annotation>
    <xs:documentation>
      <osd:label>Pricing</osd:label>
      <osd:description>Pricing grid </osd:description>
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="amount" type="xs:int">
        <xs:annotation>
          <xs:documentation>
            <osd:label>Amount borrowed</osd:label>
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="monthly" type="xs:int">
        <xs:annotation>
          <xs:documentation>
            <osd:label>Monthly payment </osd:label>
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="cost" type="xs:int">
        <xs:annotation>
          <xs:documentation>
            <osd:label>Cost</osd:label>
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Aggregated lists have a dedicated editor in EBX5. This editor allows you to add occurrences or to delete occurrences.

Including external data models

Including another data model in your current model allows you to use the reusable types that are defined in that data model. You can therefore use the inclusion of external data models to share data types between multiple XML Schema Documents.

To include another XML Schema Document in your model, thereby including the data types that it defines, specify the element `xs:include` as follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:include schemaLocation="./schemaToInclude.xsd"/>
  ...
</xs:schema>
```

The attribute `schemaLocation` is mandatory and must specify either an absolute path or a relative path to the XML Schema Document to include.

The inclusion of XML Schema Documents is not namespace aware, thus all included data types must belong to the same namespace. As a consequence, including XML Schema Documents that define data types of the same name is not supported.

EBX5 includes extensions with specific URNs for including embedded data models and data models packaged in modules.

To include an embedded data model in a model, specify the URN `urn:ebx:publication:aPublicationName` for the attribute `schemaLocation`, where 'aPublicationName' is the name of a publication that was created using the data model assistant by publishing an embedded data model. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:include schemaLocation="urn:ebx:publication:myPublication"/>
  ...
</xs:schema>
```

To include a data model packaged in a module, specify the URN `urn:ebx:module:aModuleName:aPathInModule` for the attribute `schemaLocation`. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:include schemaLocation="urn:ebx:module:ebx-tutorial:/WEB-INF/ebx/schema/ebx-
tutorial.xsd"/>
  ...
</xs:schema>
```

Note

If the packaged data model uses Java resources, the class loader of the module containing the data model will be used at runtime for resolving these resources.

Also, as the above URNs are specific to EBX5, they will not be able to be resolved by external tools. Thus, when using an external editor to edit any XML Schema Documents that contain these URNs, the editor may report errors. To workaround this issue, you can manually substitute the physical locations of the targeted XML Schema documents for the URNs before beginning to edit the XML Schema document using an external editor.

CHAPTER 74

Tables, filters and selection nodes

EBX5 supports the features of relational databases tables with a high volume of records and primary key identification.

Tables bring many benefits over [aggregated lists](#) . Beyond pure relational aspects, here are some features that tables provide:

- filters and searches;
- sorting, custom views and custom hierarchies;
- identity constraints: primary keys, [foreign keys](#) and [uniqueness constraints](#) ;
- detailed permissions for creation, modification, delete;
- dynamic and contextual permissions at individual record level;
- detailed comparison and merge;
- inheritance capacity at record level (see [data set inheritance](#));
- performance and memory optimizations.

Table definition

An element declaration with *maxOccurs* > 1 is declared as a table by adding the following annotation:

```
<xs:annotation>
  <xs:appinfo>
    <osd:table>
      <primaryKeys>/pathToField1 /pathToField...n</primaryKeys>
    </osd:table>
  </xs:appinfo>
```

</xs:annotation>

Element	Description	Required
primaryKeys	<p>Specifies the primary key fields of the table. Each element in the primary key is denoted by its absolute XPath notation that starts just under the table root element. If there are several elements in the primary key, the list is white-space delimited.</p> <p>Note: A specific whitespaces management has been defined for primary keys of type <code>xs:string</code>.</p>	Yes.
defaultLabel	<p>Defines the end-user display of records. Multiple localized versions can be specified:</p> <ul style="list-style-type: none"> Default expression and localized expressions are specified using the <code>defaultLabel</code> element, for example: <pre> <defaultLabel>Product: \${./ProductID}</defaultLabel> <defaultLabel xml:lang="fr-FR">Produit : \${./ProductID}</defaultLabel> <defaultLabel xml:lang="en-US">Product : \${./ProductID}</defaultLabel> </pre> JavaBean that implements the interface UILabelRenderer ^{API} <code>[UILabelRenderer]</code> and/or the interface UILabelRendererForHierarchy ^{API} <code>[UILabelRendererForHierarchy]</code>. The JavaBean is specified by means of the attribute <code>osd:class</code>, for example: <pre> <defaultLabel osd:class="com.wombat.MyLabel"></defaultLabel> </pre> <p>Note: The priority of the tags at display is the following:</p> <ol style="list-style-type: none"> <code>defaultLabel</code> tags with a JavaBean (but it is not allowed to define several renderers of the same type); <code>defaultLabel</code> tags with a localized expression; <code>defaultLabel</code> tag with a default expression. 	No.
index	<p>Property used to index some fields; indexing speeds up table access for requests matching these fields. (see performances).</p> <p>Notes:</p> <ul style="list-style-type: none"> It is possible to define several indexes on a table. It is not allowed to define two indexes with the same name. It is not allowed to declare two indexes containing the same exact fields. An indexed field must be terminal. An indexed field cannot be a list or under a list. A field declared as an <i>inherited field</i> cannot be indexed. A field declared as a function cannot be indexed. 	No.

Element	Description	Required
	<p>For performance purposes, the following nodes are automatically indexed:</p> <ul style="list-style-type: none"> Primary keys nodes (see primary keys). Nodes defining a foreign key constraint (see foreign key constraint). Nodes declared as being unique (see uniqueness constraint). Auto-incremented nodes (see auto-incremented values). 	
mayCreateRoot	expression that specifies when a <i>root</i> record may be created (see Definition modes). The expression must follow the syntax below (see property <code>mayDelete</code>).	No, default is "always".
mayCreateOverwriting	expression that specifies when an <i>overwriting</i> record may be created (see Definition modes). The expression must follow the syntax below (see property <code>mayDelete</code>).	No, default is "always".
mayCreateOcculting	expression that specifies when an <i>occulting</i> record may be created (see Definition modes). The expression must follow the syntax below (see property <code>mayDelete</code>).	No, default is "always".
mayDuplicate	expression that specifies when a record may be <i>duplicated</i> . The expression must follow the syntax below (see property <code>mayDelete</code>).	No, default is "always".
mayDelete	<p>expression that specifies when a record may be <i>deleted</i> . The expression must follow the syntax:</p> <p>The <code>may...</code> expression specifies when the action is possible, however the user access rights may restrict this possibility independently. The expressions have the following syntax:</p> <pre> expression ::= always never <condition>* condition ::= [root:yes root:no] "always": the operation is "always" possible (but user rights may restrict this). "never": the operation is never possible. "root:yes": the operation is possible if the record is in a root instance. "root:no": the operation is not possible if the record is in a root instance. If the record does not verify any condition, then default is taken. </pre>	No, default is "always".

Below is an example of a product catalog as a table:

```

<xs:element name="product" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>
      <osd:label>Product Table </osd:label>
    
```

```

    <osd:description>List of products in Catalog </osd:description>
  </xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:annotation>
  <xs:appinfo>
    <osd:table>
      <primaryKeys>/productRange /productCode</primaryKeys>
      <index name="indexProductCode">/productCode</index>
    </osd:table>
  </xs:appinfo>
</xs:annotation>
<xs:sequence>
  <xs:element name="productRange" type="xs:string"/><!-- key -->
  <xs:element name="productCode" type="xs:string"/><!-- key -->
  <xs:element name="productLabel" type="xs:string"/>
  <xs:element name="productDescription" type="xs:string"/>
  <xs:element name="productWeight" type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>

```

See also:

- [Foreign keys](#)
- [Main actions](#)
- [Simple tabular view](#)
- [Hierarchies](#)
- [History](#)

Specifying filters on a table

By default, tables present all records in EBX5. However, the user has the ability to add filters and display only the records corresponding to given criteria.

In order to do that, you must follow the steps below:

- Define the filter in the data model (see example below),
- Implement the class in charge of the filtering, that extends `UITableFilter` ^{API}
[UITableFilter]

As a result, a filter option will appear in the user screen, giving the opportunity to refine your search.

```

<xs:element name="product" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:appinfo>
      <osd:table>
        ...
      </osd:table>
      <osd:uiFilter class="com.mycompany.product.filters.productCodeFilter">
        <label>Product code filter</label>
        <label xml:lang="en-US">Product code filter</label>
        <label xml:lang="fr-FR">Filtre code produit</label>
      </osd:uiFilter>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

```

<osd:uiFilter class="com.mycompany.product.filters.productLabelFilter">
  <label>Product label filter</label>
  <label xml:lang="en-US">Product label filter</label>
  <label xml:lang="fr-FR">Filtre nom produit</label>
</osd:uiFilter>
</xs:appinfo>
</xs:annotation>
...
</xs:element>

```

Element	Description	Required
label	Specifies a label for the filter. This label can be localized using the <code>xml:lang</code> attribute.	No.
class	Attribute declaring the Java class which extends <code>UITableFilter</code> . Full qualified Java name must be used.	Yes.

Selection nodes

An element declaration may define a dynamic and contextual XPath selection. In this case, the user interface provides a link so that the user can navigate to the pre-filtered table that corresponds to the selection.

A selection node is useful for creating an *association between two entities* in the user interface and also for validation purposes. For example, the tutorial library data model specifies that a book is written by an author (defined explicitly by a foreign key in the 'Book' complex type definition). The relation in the opposite direction, that an author has written certain books, cannot be easily expressed in XML Schema, unless the 'Author' complex type definition includes the following selection node:

```

<xs:element name="linkToAuthorTitles" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:select>
        <xpath>/root/Titles[au_id =${../au_id}]</xpath>
        <minOccurs>1</minOccurs>
        <minOccursValidationMessage>
          <severity>error</severity>
          <message>A default validation message.</message>
          <message xml:lang="en-US">A validation message in English.</message>
          <message xml:lang="fr-FR">Un message de validation en français.</message>
        </minOccursValidationMessage>
        <maxOccurs>10</maxOccurs>
        <maxOccursValidationMessage>
          <severity>error</severity>
          <message>A default validation message.</message>
          <message xml:lang="en-US">A validation message in English.</message>
          <message xml:lang="fr-FR">Un message de validation en français.</message>
        </maxOccursValidationMessage>
      </osd:select>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

The element `minOccurs` specifies that, for being valid, an author must have written at least one book.

The element `maxOccurs` specifies that, for being valid, an author must have written at most ten books.

Note

The "official" cardinality constraints (`minOccurs="0"` `maxOccurs="0"`) are required because from an external XML point of view (that is, an instance of XML Schema), the corresponding node is absent. In other words, a selection node is a "virtual" element regarding XML and XML Schema.

It is also possible to specify an additional constraint on the relation. In the following example, each relation between an author and a book is valid, if the date of the publication of the book is anterior to the death of the current author:

```
<osd:select>
  <xpath>/root/Titles[au_id =${../au_id}]/</xpath>
  <constraintPredicate>date-less-than(datePub, ${../death})</constraintPredicate>
  <constraintPredicateValidationMessage>
    <severity>error</severity>
    <message>A default validation message.</message>
    <message xml:lang="en-US">A validation message in English.</message>
    <message xml:lang="fr-FR">Un message de validation en français.</message>
  </constraintPredicateValidationMessage>
```


</osd:select>

Element	Description	Required
xpath	<p>Specifies the selection to be performed, relative to the current node.</p> <p>Examples: <code>/root/Titles[au_id = \${../au_id}]</code> or <code>//Titles[au_id = \${../au_id}]</code> or <code>../Titles[au_id = \${../au_id}]</code>.</p> <p>The path up to the predicate (for example <code>../Titles</code>) specifies the target table to be filtered. This part of the path is resolved relative to the current table root.</p> <p>If the selection depends on the local state, the XPath expression predicate must include references to the node on which it depends, with this notation: <code>\${ <relative-path> }</code> where <code>relative-path</code> is a path that locates the element relative to the node that holds the selection link.</p> <p>For XPath syntax, see EBX5 XPath supported syntax.</p>	Yes.
container	Reference of the instance that contains the target table.	No, default is current instance.
minOccurs	Specifies an additional validation constraint: the selection must be at least of the specified size.	No, default is 0.
minOccursValidationMessage	<p>Specifies additional localized messages that will be displayed if the selection does not comply with the <code>minOccurs</code> constraint.</p> <p>A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'.</p> <p>Each localized message variant is defined in a nested <code>message</code> element with its locale in an <code>xml:lang</code> attribute. To specify a default, non-localized message, define a <code>message</code> element with no <code>xml:lang</code> attribute.</p>	No.
maxOccurs	Specifies an additional validation constraint: the selection must be at most of the specified size.	No, by default the maximum is not restricted.
maxOccursValidationMessage	<p>Specifies an additional localized message that will be displayed if the selection does not comply with the <code>maxOccurs</code> constraint.</p> <p>A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'.</p> <p>Each localized message variant is defined in a nested <code>message</code> element with its locale in an <code>xml:lang</code> attribute. To specify a default, non-localized message, define a <code>message</code> element with no <code>xml:lang</code> attribute.</p>	No.
constraintPredicate	Specifies an additional validation constraint: each relation of the selection must satisfy the specified	No.

Element	Description	Required
	XPath predicate. The notation <code>\${<relative-path>}</code> has the same meaning as for the <code>xpath</code> element (see above).	
<code>constraintPredicateValidationMessage</code>	<p>Specifies a localized message to display when the constraint predicate is not satisfied.</p> <p>A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'.</p> <p>Each localized message variant is defined in a nested <code>message</code> element with its locale in an <code>xml:lang</code> attribute. To specify a default, non-localized message, define a <code>message</code> element with no <code>xml:lang</code> attribute.</p>	No.

Important: When creating a data set, you can create a data set that defines a selection node to a container that does not exist in the repository. However, the content of this data set will not be available after its creation. After the creation of the container, a data model refresh is required to make the data set available. When creating a data set that refers to a container that does not yet exist, the following limitations apply:

- Triggers defined at the data set level are not executed,
- Default values on fields outside tables are not initialized,
- During an archive import, it is not possible to create a data set that refers to a container that does not exist.

CHAPTER 75

Constraints, triggers and functions

Facets allow you to define constraints on the data in your data models. EBX5 supports XML Schema facets and provides extended and programmatic facets for advanced data controls.

XML Schema supported facets

The tables below show the facets that are supported by different data types.

Key:

- **X** - Supported
- **1** - The `whiteSpace` facet can be defined, but is not interpreted by EBX5
- **2** - In XML Schema, boundary facets are not allowed on the type `string`. Nevertheless, EBX5 allows those facets as extensions.
- **3** - The `osd:resource` type only supports the facet `FacetOResource`, which is required. See [Extended Facets](#).

	length	minLength	max Length	pattern	enumeration	white Space
string	X	X	X	X	X	1
boolean				X		1
decimal				X	X	1
dateTime				X	X	1
time				X	X	1
date				X	X	1
anyURI	X	X	X	X	X	1
Name	X	X	X	X	X	1
integer				X	X	1
osd:resource 3						

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
string			2	2	2	2
boolean						
decimal	X	X	X	X	X	X
dateTime			X	X	X	X
time			X	X	X	X
date			X	X	X	X
anyURI						
Name			2	2	2	2

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
integer	X	X	X	X	X	X
osd:resource 3						

Example:

```
<xs:element name="loanRate">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="4.5" />
      <xs:maxExclusive value="17.5" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Uniqueness constraint

It is possible to define a uniqueness constraint, by using the standard XML Schema element [xs:unique](#). This constraint indicates that a value or a set of values has to be unique inside a table.

In the example below, a uniqueness constraint is defined on the *publisher* table, for the target field *name* (this means that two records of the *publisher* table cannot have the same name):

```
<xs:element name="publisher">
  ...
  <xs:complexType>
    <xs:sequence>
      ...
      <xs:element name="name" type="xs:string" />
      ...
    </xs:sequence>
  </xs:complexType>
  <xs:unique name="uniqueName">
    <xs:annotation>
      <xs:appinfo>
        <osd:validationMessage>
          <severity>error</severity>
          <message>Name must be unique in table.</message>
          <message xml:lang="en-US">Name must be unique in table.</message>
          <message xml:lang="fr-FR">Le nom doit être unique dans la table.</message>
        </osd:validationMessage>
      </xs:appinfo>
    </xs:annotation>
    <xs:selector xpath="." />
    <xs:field xpath="name" />
  </xs:unique>
</xs:element>
```

A uniqueness constraint has to be defined inside a table and has the following properties:

Property	Description	Mandatory
name attribute	Identifies the constraint in the data model.	Yes
xs:selector element	Indicates the table where the uniqueness applies, using a restricted XPath expression ('..' is forbidden). It can also indicate an element inside the table (without changing the meaning of the constraint).	Yes
xs:field element	Indicates the field in the context whose values must be unique, using a restricted XPath expression. It is possible to indicate that a set of values must be unique by defining multiple xs:field elements.	Yes

Additional localized validation messages can be defined using the element `osd:validationMessage` under the elements `annotation/appinfo`. If no custom validation messages are defined then a built-in validation message will be used.

Limitations:

1. The target of the `xs:field` element has to be in a table.
2. Uniqueness constraint cannot apply if the field is inside an aggregated list.
3. Uniqueness constraint cannot apply if the field is computed.

Extended facets

EBX5 provides additional constraints that are not specified in XML Schema, but that are useful for managing master data.

In order to guarantee conformance to XML Schema, those Extended Facets are defined under the element `annotation/appinfo/otherFacets`.

Foreign keys

A reference to a [table](#) is defined using the extended facet `osd:tableRef`.

The node holding the `osd:tableRef` declaration must be of type `xs:string`. At instantiation, any value of the node identifies a record in the target table by mean of its **primary key syntax**

[PrimaryKey]^{API}. This extended facet is also interpreted as an enumeration whose values refer to the records in the target table.

Element	Description	Required
tablePath	XPath expression that specifies the target table.	Yes.
container	Reference of the data set that contains the target table.	Only if element 'branch' (i.e data space) is defined. Otherwise, default is current data set.
branch	Reference of the data space (former 'branch') that contains the data set container.	No, default is current data space or snapshot.
display	<p>Custom display for presenting the selected foreign key in the current record and the sorted list of possible keys. Two variants can be specified, either pattern-based expressions, or a JavaBean if the needs are very specific:</p> <ul style="list-style-type: none"> Default and localized expressions. They are specified by means of the <code>display</code> and <code>pattern</code> elements, for example: <pre> <display> <pattern>Product : \${./ProductID}</pattern> <pattern xml:lang="fr-FR">Produit : \${./ProductID}</pattern> <pattern xml:lang="en-US">Product : \${./ProductID}</pattern> </display> </pre> A JavaBean that implements the interface TableRefDisplay [TableRefDisplay]^{API}. It is specified by means of the attribute <code>osd:class</code>; for example: <pre> <display osd:class="com.wombat.MyLabel"></display> </pre> 	No, if the <code>display</code> property is absent, then the table's record rendering is used.
filter	<p>Specifies an additional constraint that filters the records of the target table. Two types of filters are available:</p> <ul style="list-style-type: none"> An XPath filter is an XPath predicate in the target table context. It is specified by means of the <code>predicate</code> element; for example: <pre> <filter><predicate>type = \${../refType}</predicate></filter> </pre> <p>A localized validation message can be specified using the element <code>validationMessage</code>, which will be displayed to the end user at validation time if a record is not accepted by the filter.</p> <p>A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'.</p> <p>Each localized message variant is defined in a nested <code>message</code> element with its locale in an <code>xml:lang</code> attribute.</p> 	No.

Element	Description	Required
	<p>To specify a default message for unsupported locales, define a <code>message</code> element with no <code>xml:lang</code> attribute.</p> <ul style="list-style-type: none"> A programmatic filter is a JavaBean that implements the interface TableRefFilter <code>[TableRefFilter]^{API}</code>. It is specified by means of the attribute <code>osd:class</code>; for example: <pre><filter osd:class="com.wombat.MyFilter"></filter></pre> <p>Additional validation messages can be specified during the setup of the programmatic filter using the dedicated methods of the interface TableRefFilterContext <code>[TableRefFilterContext]^{API}</code>.</p> <p>Notes:</p> <ul style="list-style-type: none"> Attribute <code>osd:class</code> and the property <code>predicate</code> cannot be set simultaneously. 	
<code>validationMessage</code>	<p>Specifies localized validation messages for the <code>osd:tableRef</code>.</p> <p>A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'.</p> <p>Each localized message variant is defined in a nested <code>message</code> element with its locale in an <code>xml:lang</code> attribute. To specify a default message for unsupported locales, define a <code>message</code> element with no <code>xml:lang</code> attribute.</p>	No.

Important: When creating a data set, you can create a data set that defines a foreign key to a container that does not exist in the repository. However, the content of this data set will not be available after its creation. After the creation of the container, a data model refresh is required to make the data set available. When creating a data set that refers to a container that does not yet exist, the following limitations apply:

- Triggers defined at the data set level are not executed,
- Default values on fields outside tables are not initialized,
- During an archive import, it is not possible to create a data set that refers to a container that does not exist.

The example below specifies a reference (foreign key) to a record of *catalog* element.

```
<xs:annotation>
  <xs:appinfo>
    <osd:otherFacets>
      <osd:tableRef>
        <tablePath>../catalog</tablePath>
        <display>
          <pattern xml:lang="en-US">Product: ${../ProductID}</pattern>
          <pattern xml:lang="fr-FR">Produit : ${../ProductID}</pattern>
        </display>
        <filter>
          <predicate>type = ${../refType} </predicate>
          <validationMessage>
```



```

    <severity>error</severity>
    <message>A default error message</message>
    <message xml:lang="en-US">A localized error message</message>
    <message xml:lang="fr-FR">Un message d'erreur localisé</message>
  </validationMessage>
</filter>
</osd:tableRef>
</osd:otherFacets>
</xs:appinfo>
</xs:annotation>

```

See also:

- [Table definition](#)
- **Primary key syntax** [*PrimaryKey*]^{API}
- [Extraction of foreign keys \(XPath predicate syntax\)](#)

Dynamic constraints

Those facets retain the XML Schema semantics, but the value attribute is replaced by a path attribute that allows fetching the value from another element in the adaptation:

- length
- minLength
- maxLength
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

By means of these facets, the data model may be modified dynamically, in the adaptations.

Example:

```

<xs:element name="amount">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:minInclusive path="/domain/Loan/Pricing/AmountMini/amount" />
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>

```

In this example, the boundary of the facet minInclusive is not statically defined. The value of the boundary comes from the node */domain/Loan/Pricing/AmountMini/amount*

FacetOResource constraint

This facet must be defined for each resource type. It has the following attributes:

- `moduleName`: indicates with an alias which EBX5 module contains the resource. If it is the module itself that contains the resource, the alias must be preceded by "wbp". Otherwise, the alias must be one of the values defined in the element `<dependencies>` of the file 'module.xml';
- `resourceType`: represents the resource type that is one of the following values: "ext-images", "ext-jscripts", "ext-stylesheets", "ext-html";
- `relativePath`: represents the local directory where the resource is located, just under the directory named with the value `resourceType`. In the example below, it is the directory `www/common/images/`. Hence, for this example, the resource is located at `www/common/images/promotion/` where the `www/` directory is at the same level as the `WEB-INF/` directory.

Furthermore, if a resource is defined in a localized directory (`www/en/` for instance), it will be considered only if another resource of the same name is defined in the directory `www/common/`.

This facet has the same behavior as an enumeration facet: the values are collected by recursively listing all the files in 'local path' from the specified directory 'resource type' in the specified 'module'.

Example:

```
<xs:element name="promotion" type="osd:resource">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:FacetOResource osd:moduleName="wbp"
          osd:resourceType="ext-images" osd:relativePath="promotion/" />
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

An overview of the standard directory structure of a module (Java EE web application) for EBX5 is detailed in the section [Modules - Structure](#).

excludeValue constraint

This facet verifies that an instance is different from a specified value.

Example:

```
<xs:element name="roleName">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:excludeValue value="">
          <osd:validationMessage>
            <severity>error</severity>
            <message>Please select address role(s).</message>
          </osd:validationMessage>
        </osd:excludeValue>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

```

    </osd:otherFacets>
  </xs:appinfo>
</xs:annotation>
<xs:simpleType type="xs:string" />
</xs:element>

```

In this example, the empty string is excluded from the possible values.

excludeSegment constraint

This facet verifies that an instance is not included in a segment of values. Boundaries are excluded.

Example:

```

<xs:element name="zipCode">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:excludeSegment minValue="20000" maxValue="20999">
          <osd:validationMessage>
            <severity>error</severity>
            <message>Postal code not valid.</message>
          </osd:validationMessage>
        </osd:excludeSegment>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  <xs:simpleType type="xs:string" />
</xs:element>

```

In this example, values between 20000 and 20999 are excluded.

Enumeration facet filled by another node

By default, an enumeration facet is described statically in XML Schema.

The content of an enumeration facet can be filled dynamically by another node of the data model.

Filling by a list

With the following information, we indicate that the content of an enumeration facet comes from the node *CountryList*.

```

<xs:annotation>
  <xs:appinfo>
    <osd:otherFacets>
      <osd:enumeration osd:path="../../CountryList" />
    </osd:otherFacets>
  </xs:appinfo>
</xs:annotation>

```

The node *CountryList* :

- Must be of the same type as the node that has the enumeration facet
- Must be an aggregated list (maxOccurs > 1).

Example:

```
<xs:element name="FacetEnumBasedOnList">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CountryList" maxOccurs="unbounded">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="DE" osd:label="Germany" />
            <xs:enumeration value="AT" osd:label="Austria" />
            <xs:enumeration value="BE" osd:label="Belgium" />
            <xs:enumeration value="JP" osd:label="Japan" />
            <xs:enumeration value="KR" osd:label="Korea" />
            <xs:enumeration value="CN" osd:label="China" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="CountryChoice" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <osd:otherFacets>
              <osd:enumeration osd:path="../../CountryList" />
            </osd:otherFacets>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Programmatic facets

A programmatic constraint can be added on any XML element declaration that refers to a simple type.

In order to guarantee the conformance to XML Schema, those constraints are specified under the element annotation/appinfo/otherFacets.

Programmatic constraints

A programmatic constraint is defined by a Java class that implements the interface **Constraint** [Constraint]^{API}.

Additional parameters can be defined. Consequently, the implemented Java class must conform to the JavaBean protocol. In the example below, the Java class must define the methods: *getParam1()*, *setParam1(String)*, ..., *getParamX()*, *setParamX(String)*, etc.

Example:

```
<xs:element name="amount">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:constraint class="com.foo.CheckAmount">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraint>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

See also: *JavaBeans Specifications* [*package-summary*]^{API}

Programmatic enumeration constraints

An enumeration constraint adds to a basic programmatic constraint the definition of an ordered list of values. Such a constraint is defined by a Java class that implements the interface **ConstraintEnumeration** [ConstraintEnumeration]^{API}. This facet allows the selection of a value from a list.

Example:

```
<xs:element name="amount">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:constraintEnumeration class="com.foo.CheckAmountInEnumeration">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraintEnumeration>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

Constraint on null value

Sometimes, a value is mandatory only if some conditions are verified (for example if another field has some specific value). In this case, the standard XML Schema attribute `minOccurs` will not help because it is static.

For checking if a value is mandatory or not, according to its context:

1. a programmatic constraint must be defined by a Java class (see above);
2. this class must also implement the interface **ConstraintOnNull** [ConstraintOnNull]^{API};
3. XML Schema cardinality attributes must specify that the element is optional (minOccurs="0" and maxOccurs="1").

Note

By default, constraint on null value is not checked on user input. In order to enable this check, the ['checkNullInput' property](#) must be set and if the element is terminal, the adaptation instance must also be activated.

Example:

```
<xs:element name="amount" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:constraint class="com.foo.CheckIfNull">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraint>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

Constraints on table

A constraint on table is defined by a Java class that implements the interface **ConstraintOnTable** [ConstraintOnTable]^{API}.

A constraint on table can only be defined on a table node.

Additional parameters can be defined. Consequently, the implemented Java class must conform to the JavaBean protocol. In the example below, the Java class must define the methods: **getParam1()**, **setParam1(String)**, ..., **getParamX()**, **setParamX(String)**, etc.

Example:

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:appinfo>
      <osd:table>
        <primaryKeys>/key</primaryKeys>
      </osd:table>
      <osd:otherFacets>
        <osd:constraint class="com.foo.checkTable">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraint>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

```

</osd:constraint>
</osd:otherFacets>
</xs:appinfo>
</xs:annotation>
</xs:element>

```

Important: Regarding the validation process, constraints on tables are only checked when calling a validation report of an adaptation instance/table. This means, for performance purposes, these constraints are not checked when updates (such as record insertion, deletion or modification) on tables occur. However the internal incremental validation framework will optimize the validation cost of these constraints if dependencies are defined. For more information, see [performance](#) section.

See also: *JavaBeans Specifications* [*package-summary*]^{API}

Triggers and functions

Computed values

By default, data is read and persisted in the XML repository. Nevertheless, they may be the result of a specific computation and/or access an external database (a RDBMS, a central system...).

EBX5 allows you to take into account other data in the current adaptation context.

This is made possible by defining *functions*.

A function is specified in the data model by using the `osd:function` element (see example below).

- The value of *class* attribute must be the qualified name of a Java class which implements the Java interface **ValueFunction** [*ValueFunction*]^{API}.
- Additional parameters may be specified at the data model level, in this case the JavaBeans convention is applied.

Example:

```

<xs:element name="computedValue">
  <xs:annotation>
    <xs:appinfo>
      <osd:function class="com.foo.ComputeValue">
        <param1>...</param1>
        <param...n>...</param...n>
      </osd:function>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>

```

Triggers

Instances or table records can be associated with methods that are automatically executed when some operations are performed: creation, update, delete, etc.

In the data model, these triggers must be declared under the `annotation/appinfo` element using the `osd:trigger` tag.

For adaptation instances triggers, we have to define, inside the `osd:trigger` tag, a Java class that extends the abstract class **InstanceTrigger** [InstanceTrigger]^{API}.

Note that in the case of adaptation instance triggers, it is advised to define `annotation/appinfo/osd:trigger` tags just under the root element of the considered data model.

Example:

```
<xs:element name="root" osd:access="--">
  ...
  <xs:annotation>
    <xs:appinfo>
      <osd:trigger class="com.foo.MyInstanceTrigger">
        <param1>...</param1>
        <param...n>...</param...n>
      </osd:trigger>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

For the definition of tables records triggers, we have to define, inside the `osd:trigger` tag, a Java class that extends the abstract class **TableTrigger** [TableTrigger]^{API}.

In the case of table records triggers, it is advised to define the `annotation/appinfo/osd:trigger` tags just under the element describing the considered table or table type.

Examples:

On a table element:

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:appinfo>
      <osd:table>
        <primaryKeys>/key</primaryKeys>
      </osd:table>
      <osd:trigger class="com.foo.MyTableTrigger" />
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

or on a table type element:

```
<xs:complexType name="MyTableType">
  ...
  <xs:annotation>
    <xs:appinfo>
      <osd:trigger class="com.foo.MyTableTrigger">
        <param1>...</param1>
        <param...n>...</param...n>
      </osd:trigger>
    </xs:appinfo>
  </xs:annotation>
</xs:complexType>
```



```

    </xs:appinfo>
  </xs:annotation>
  ...
</xs:complexType>

```

Note that additional parameters can be defined. Consequently, the Java class implemented must conform to the JavaBean protocol. In the example above, the Java class must define the methods: *getParam1()*, *setParam1(String)*, ..., *getParamX()*, *setParamX(String)*, etc.

Auto-incremented values

It is possible to define auto-incremented values. Auto-incremented values are only allowed inside a table, and they must be of type `xs:int` or `xs:integer`.

An auto-increment is specified in the data model by using the `osd:autoIncrement` element under the `annotation/appinfo` element tag.

Example:

```

<xs:element name="autoIncrementedValue" type="xs:int">
  <xs:annotation>
    <xs:appinfo>
      <osd:autoIncrement />
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

There are two optional elements `start` and `step`.

Example:

```

<xs:element name="autoIncrementedValue" type="xs:int">
  <xs:annotation>
    <xs:appinfo>
      <osd:autoIncrement>
        <start>100</start>
        <step>5</step>
      </osd:autoIncrement>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

The `start` attribute specifies the first value for this auto-increment. If this attribute is not specified, then the value 1 is set by default.

The `step` attribute specifies the step for the next value to be generated by the auto-increment. If this attribute is not specified, then the value 1 is set by default.

A field specifying an `osd:autoIncrement` has the following behavior:

- The computation and allocation of the field value are performed whenever a new record is inserted and the field value is still undefined.
- No allocation is performed if a programmatic insertion already specifies a non `null` value. For example, if an archive import or an XML import specifies the value, then it is preserved. Note also that, consequently, the allocation is not performed for a record insertion in occulting or overwriting modes.
- A newly allocated value is, whenever possible, unique in the scope of the repository. More precisely, the uniqueness of the allocation spans over all the data sets of the data model, and it also spans over all the data spaces. The latter case allows the merging of a data space to its parent with a reasonable guarantee that there will not be a conflict if the `osd:autoIncrement` is part of the records' primary key. This principle has a very specific limitation: when a massive update transaction specifying values is performed concurrently to a transaction allocating a value on the same field, it is possible that the latter transaction will allocate a value that will be set in the first transaction (there is no locking between several data spaces).

Internally, the auto-increment value is stored in the 'Auto-increments' table of the repository (in the user interface, it can be accessed from the Administration pane). This field is automatically updated, so that it defines the greatest value ever set on the associated `osd:autoIncrement` field, in any instance and any data space in the repository. This value is computed, taking into account the max value found in the table being updated.

In some particular case, for example when multiple environments have to be managed (development, test, production), each having different auto-increment ranges, it may be required to avoid this "max value" check. This particular behavior can be achieved, thanks to the `disableMaxTableCheck` property:

- locally, by setting a parameter element in auto-increment declaration:
`<disableMaxTableCheck>true</disableMaxTableCheck>` ,
- for the whole data model, by setting `<osd:disableMaxTableCheck="true">` in `xs:appinfo` of data model declaration,
- or globally, by setting property `ebx.autoIncrement.disableMaxTableCheck=true` in main configuration file (`ebx.properties`).

Note that it is generally not recommended to enable this property, as this could generate conflicts in the auto-increment values.

Note: when this option is enabled globally, it becomes possible to create records in the table of auto-increments by import XML or CSV for example. If this option is not selected, the creation of records in the table of auto-increments is prohibited to maintain the integrity of the repository.

User input control

Check null input

According to EBX5 default validation policy, in order to allow a temporary incomplete input, the control whether an element is defined when it is mandatory is not performed on user input, but on adaptation validation only. If the 'mandatory feature' must be checked immediately on user input, the element must additionally specify: `osd:checkNullInput="true"`.

Note: A value is mandatory if the data model specifies that the element is mandatory, either statically (`minOccurs="1"`) or dynamically (by means of a Constraint on null). For terminal elements mandatory values are checked only for an activated adaptation instance. For non-terminal elements the adaptation instance does not need to be activated.

Example:

```
<xs:element name="amount" osd:checkNullInput="true" minOccurs="1">
  ...
</xs:element>
```

See also:

- [constraint on null](#)
- [Whitespace management](#)
- [Empty string management](#)

EBX5 whitespace management for data types

According to XML Schema (cf. <http://www.w3.org/TR/xmlschema-2/#rf-whiteSpace>), whitespaces handling must follow one of *preserve*, *replace* or *collapse* procedures:

- *preserve*. No normalization is done, the value is not changed
- *replace*. All records of #x9 (tab), #xA (line feed) and #xD (carriage return) are replaced with #x20 (space)
- *collapse*. After the processing implied by replace, contiguous sequences of #x20's are collapsed to a single #x20, and leading and trailing #x20's are removed

EBX5 general management

EBX5 complies with XML Schema recommendation:

- For nodes of type `xs:string` (whether primary key element or not), whitespaces are always preserved and an empty string is never converted to `null`.
- For other nodes (non `xs:string` type), whitespaces are always collapsed and an empty string is converted to `null`.

Exceptions: for nodes of type `osd:html` or `osd:password`, whitespaces are always preserved and an empty string is converted to `null`. For nodes of type `xs:string` defining property `osd:checkNullInput="true"` an empty string is interpreted as `null` at user input in EBX5.

Note

Values that are composed of *only whitespaces* are considered to be empty.

Specific case for handling whitespaces on a primary key of type string

For primary key columns of type `xs:string`, a default EBX5 constraint is defined. This constraint forbids empty strings and non collapsed whitespaces values at validation level.

However, if the primary key node specifies its own `xs:pattern` facet, this last one overrides the default EBX5 constraint. For example, the specific pattern `.*` would accept any strings (note that we do not recommend this).

The default constraint allows the handling of some ambiguities. For example, it would be difficult for a user to distinguish between the following strings: "12 34" and "12 34". For usual values, it does not create conflicts but for primary keys, it could create errors.

Note: EBX5 provides a complete [Tables specification](#), that allows the organization and management of your data in EBX5 as in relational databases.

Empty string management

Default conversion

For nodes of type `xs:string` at user input no distinction is made between an empty string and a null value. That is, an empty string value is automatically converted to `null` at user input.

Distinction between empty string and null value

There are some cases where the distinction is still made between an empty string and a null value. The distinction is still performed in the following cases:

- A primary key defines a pattern allowing empty strings.
- An element defines a foreign key constraint and the target table has a single primary key defining a pattern allowing empty strings.
- An element defines a static enumeration containing an empty string.
- An element defines a dynamic enumeration to another one defining these specific distinction cases.

If the distinction is still made between an empty string and a null value this will imply the following behaviors:

- An empty string will not be converted to `null` at user input,
- Input field on nodes of type `xs:string` will display an additional button for setting to `null` the value of the node,
- At validation time an empty string will be considered as a compliant value regarding to a `minOccurs="1"` property.

Note: in relational mode, the Oracle database does not support the distinction between empty strings and null values, then these specific cases are not supported.

See also: [Relational mode](#)

CHAPTER 76

Labels and messages

EBX5 allows you to enrich your data models with labels and error messages displayed in the interface.

Label and description

A label and a description can be added to each node in an adaptation model.

In EBX5, each adaptation node is displayed with its label . If no label is defined, the name of the element is used instead.

Two notations are available:

- full notation: label and description are defined respectively by the elements `<osd:label>` and `<osd:description>`.
- simple notation: the label is extracted from the text content up to the first dot (' . ') and/or this extraction is not longer than 60 characters; the description is the rest of the text.

The description may also have an hyperlink, either a standard HTML `href` to an external document, or a link to another node of the adaptation within EBX5.

- `href` notation, and more generally any html notation must be escaped.
- EBX5 link notation is not escaped and it must specify the path of the target, for example:

```
<osd:link path="../../../misc1">link to another node in the adaptation</osd:link>
```

Example:

```
<xs:element name="misc1" type="xs:string">
  <xs:annotation>
    <xs:documentation>
      Miscellaneous 1. This is the description of miscellaneous element #1.
      Click <a href="http://www.orchestranetworks.com" target="_blank">here</a>
      to learn more.
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

```

<xs:element name="misc2" type="xs:string">
  <xs:annotation>
    <xs:documentation>
      <osd:label>
        Miscellaneous 2
      </osd:label>
      <osd:description>
        This is the miscellaneous element #2 and here is a
        <osd:link path="../misc1"> link to another node in the
        adaptation</osd:link>.
      </osd:description>
    </xs:documentation>
  </xs:annotation>
</xs:element>

```

If a node points to a named type, then the label of the node replaces the label of the named type. The same mechanism applies for the description of the node (`osd:description`).

Note

Regarding whitespace management, the label of a node is always *collapsed* at display (that is, contiguous sequences of blanks are collapsed to a single blank, and leading and trailing blanks are removed). In contrast, for descriptions, the whitespaces are always *preserved*.

Dynamic labels and descriptions

As an alternative to statically defining the localized labels and descriptions for each node, it is possible to specify a Java class that programmatically determines the labels and descriptions for the nodes of the data model. To define the class, include the element `osd:documentation`, with the attribute `class` in the data model. It is possible to pass JavaBean properties using nested parameter elements.

Example:

```

<xs:schema ...>
  <xs:annotation>
    <xs:appinfo>
      <osd:documentation class="com.foo.MySchemaDocumentation">
        <param1>...</param1>
        <param2>...</param2>
      </osd:documentation>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema ...>

```

The labels and descriptions provided programmatically take precedence over any labels and descriptions defined locally on individual nodes.

See also: [SchemaDocumentation](#) [*SchemaDocumentation*]^{API}

Enumeration labels

A simple, non-localized label can be added to each enumeration element, by means of the attribute `osd:label`.

Important: Labels defined by an enumeration element are always collapsed at display.

Example:

```
<xs:element name="Service" maxOccurs="unbounded">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="1" osd:label="Blue" />
      <xs:enumeration value="2" osd:label="Red" />
      <xs:enumeration value="3" osd:label="White" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

It is also possible to fully localize the labels by means of the standard `xs:documentation` element.

Example:

```
<xs:element name="access" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="readOnly">
        <xs:annotation>
          <xs:documentation xml:lang="en-US">
            read only
          </xs:documentation>
          <xs:documentation xml:lang="fr-FR">
            lecture seule
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="readWrite">
        <xs:annotation>
          <xs:documentation xml:lang="en-US">
            read/write
          </xs:documentation>
          <xs:documentation xml:lang="fr-FR">
            lecture écriture
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="hidden">
        <xs:annotation>
          <xs:documentation xml:lang="en-US">
            hidden
          </xs:documentation>
          <xs:documentation xml:lang="fr-FR">
            masqué
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Mandatory error message (osd:mandatoryErrorMessage)

If the node specifies the attribute `minOccurs="1"` (default behavior), then a required error message is displayed if the user does not fill the field. This error message can be specific to each node by using the element `osd:mandatoryErrorMessage`.

Example:

```
<xs:element name="birthDate" type="xs:date">
  <xs:annotation>
    <xs:documentation>
      <osd:mandatoryErrorMessage>
        Please give your birth date.
      </osd:mandatoryErrorMessage>
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

The mandatory error message can be localized:

```
<xs:documentation>
  <osd:mandatoryErrorMessage xml:lang="en-US">
    Name is mandatory
  </osd:mandatoryErrorMessage>
  <osd:mandatoryErrorMessage xml:lang="fr-FR">
    Nom est obligatoire
  </osd:mandatoryErrorMessage>
</xs:documentation>
```

Note

Regarding whitespaces management, the enumeration labels are always *collapsed* at display.

Conversion error message

For each predefined XML Schema element, it is possible to define a specific error message if the user entry has an incorrect format.

Example:

```
<xs:element name="email" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <fmt:string>
        <fmt:ConversionErrorMessage xml:lang="en-US">
          Please enter a valid e-mail address.
        </fmt:ConversionErrorMessage>
        <fmt:ConversionErrorMessage xml:lang="fr-FR">
          Saisissez un e-mail valide.
        </fmt:ConversionErrorMessage>
      </fmt:string>
    </xs:appinfo>
  </xs:annotation>
```

```
</xs:element>
```

Validation message with severity associated with a facet

The validation message that is displayed when the value of a field does not comply with a constraint can define a custom severity, a default non-localized message, and localized message variants. If no severity is specified, the default level is *error*. Blocking constraints *must* have the severity *error*.

XML Schema facet (*osd:validationMessage*)

The validation message is described by the element `osd:validationMessage` in `annotation/appinfo` under the definition of the facet.

Example:

```
<xs:element name="zipCode">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <!--facet is not localized, but validation message is localized-->
      <xs:minInclusive value="01000">
        <xs:annotation>
          <xs:appinfo>
            <osd:validationMessage>
              <severity>error</severity>
              <message>Non-localized message.</message>
              <message xml:lang="en-US">English error message.</message>
              <message xml:lang="fr-FR">Message d'erreur en français.</message>
            </osd:validationMessage>
          </xs:appinfo>
        </xs:annotation>
      </xs:minInclusive>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML Schema enumeration facet (*osd:enumerationValidationMessage*)

The validation message is described by the element `osd:enumerationValidationMessage` in `annotation/appinfo` under the definition of the field.

Example:

```
<xs:element name="Gender">
  <xs:annotation>
    <xs:appinfo>
      <osd:enumerationValidationMessage>
        <severity>error</severity>
        <message>Non-localized message.</message>
        <message xml:lang="en-US">English error message.</message>
        <message xml:lang="fr-FR">Message d'erreur en français.</message>
      </osd:enumerationValidationMessage>
    </xs:appinfo>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
```

```

<xs:enumeration value="0" osd:label="male" />
<xs:enumeration value="1" osd:label="female" />
</xs:restriction>
</xs:simpleType>
</xs:element>

```

EBX5 facet (osd:validationMessage)

The validation message is described by the element `osd:validationMessage` under the definition of the facet (which is defined in `annotation/appinfo/otherFacets`).

Example:

```

<xs:element name="price" type="xs:decimal">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:minInclusive path="../../priceMin">
          <osd:validationMessage>
            <severity>error</severity>
            <message>Non-localized message.</message>
            <message xml:lang="en-US">English error message.</message>
            <message xml:lang="fr-FR">Message d'erreur en français.</message>
          </osd:validationMessage>
        </osd:minInclusive>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

CHAPTER 77

Additional properties

Default value

It is possible to specify a default value for a field using the attribute `default`. This property is used for assigning a default value when no value is defined on a field.

The default value is displayed in the user input field at creation time. That is, the default value will be displayed when creating a new record or adding a new occurrence to an aggregated list.

Example:

```
<xs:element name="fieldWithDefaultValue" type="xs:string" default="aDefaultValue" />
```

In this example, the element specifies a default string value.

Access properties

The attribute `osd:access` defines the access mode, that is if the data of a particular data model node can be read and written. This attribute must have one of the following values: `RW`, `R-`, `CC` or `--` (see below).

For each node (XML Schema type), three adaptation types are possible:

1. *Adaptation terminal node*

This node is displayed with an associated value in EBX5. Access with the method `Adaptation.get()` uses the adaptation search algorithm.

2. *Adaptation non terminal node*

This node is a complex type that is displayed in EBX5 only if it has one child node that is an adaptation terminal node. It has no value. Access with the method `Adaptation.get()` returns `null`.

3. *Non adaptable node*

This node is not an adaptation terminal node and has no adaptation terminal node child. This node is not displayed in EBX5. Access with the method `Adaptation.get()` returns its default value if defined on `null`.

Access mode	Behavior
RW	<i>Adaptation terminal node</i> : value can be read and written in EBX5
R-	<i>Adaptation terminal node</i> : value can be only read in EBX5
CC	<i>Cut</i> : This is not an adaptation terminal node and no child is an adaptation terminal node. This "instruction" has priority on any child node whatever the value of their access attribute.
--	<p>If the node is a simple type, it is not adaptable. If the node is a complex type, it is not an adaptation terminal node and does not define any child node.</p> <p>The <code>root</code> node of a data model must specify this access mode.</p>
Default	<p>If the <code>access</code> attribute is not defined:</p> <ul style="list-style-type: none"> - If the node is a computed value, it is considered as R- - If the node is a simple type and its value is not computed, it is considered as RW - If the node is an aggregated list, it is then a terminal value and is considered as RW - Otherwise, it is not an adaptation terminal node and it does not define anything about its child nodes.

Example:

```
<xs:element name="proxyIpAddress" type="xs:string" osd:access="RW"/>
```

In this example, the element is adaptable because it is an adaptation terminal node.

Information

The element `osd:information` allows specifying additional information. This information can then be used by the integration code, for any specific purpose, by means of the method **SchemaNode.getInformation()** [SchemaNode]^{API}.

Example:

```
<xs:element name="misc" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:information>
        This is the text information of miscellaneous element.
      </osd:information>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Default view

Hiding a field in the default view

It is possible to specify that a field should be hidden by default in EBX5 using the element `osd:defaultView/hidden`. This property is used for hiding elements within the default view of a data set without defining specific access permissions. That is, if a field is configured as being hidden in the default view of a data set, then it will not be visible in all default forms and views (whether tabular or hierarchical) generated from the structure of the associated data model.

Important:

- If a field is configured to be hidden in the default view of a data set, then the access permissions associated with this field will not be evaluated.
- It is possible to display a field that is hidden in the default view of a data set by defining a custom view. Only in this case will the access permissions associated with this field will be evaluated to determine whether the field will be displayed or not.

Example:

```
<xs:element name="hiddenField" type="xs:string" minOccurs="0"/>
  <xs:annotation>
    <xs:appinfo>
      <osd:defaultView>
        <hidden>true</hidden>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the default view of a data set.

Hiding a field in the text search

To specify whether or not to hide an element in the text search tool, use the element `osd:defaultView/hiddenInSearch`. That is, if a field is configured as being hidden, then it will not be able to be selected in text search of a data set.

Important:

- If a complex field is configured as being hidden in the text search, then all the fields nested under this complex field will not be displayed in the text search.
- If a field is configured as being hidden in the default view of a data set but is not specifically hidden in the text search, then it still will not be available in the text search from the default view of a data set.

Example:

```
<xs:element name="hiddenFieldInSearch" type="xs:string" minOccurs="0"/>
  <xs:annotation>
    <xs:appinfo>
      <osd:defaultView>
        <hiddenInSeach>true</hiddenInSeach>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the text search tool of a data set.

Comparison mode

The attribute `osd:comparison` can be included on terminal node elements in order to set its comparison mode. This mode controls the how its differences are detected in a data set during comparison. The possible values for the attribute are:

default	Element is visible during comparisons of its data.
ignored	No changes will be detected when comparing two versions of modified content (records or data sets). During a merge, the data values of ignored elements are not merged on updated contents, even if the values are different. For new content, the values of ignored elements are merged. During an import of an archive, values of ignored elements are not imported when updating content. For new content, the values of ignored elements are imported.

Important:

- If a complex field is configured as being ignored during comparisons, then all the fields nested under this complex field will also be ignored.
- If a terminal field does not include the attribute `osd:comparison`, then it will be included by default during comparisons.

Restrictions:

- This property cannot be defined on non-terminal fields.
- This property cannot be defined on primary key fields.

Example:

```
<xs:element name="fieldExplicitlyIgnoredInComparison"
  type="xs:string" minOccurs="0" osd:comparison="ignored"/>
<xs:element name="fieldExplicitlyNotIgnoredInComparison"
  type="xs:string" minOccurs="0" osd:comparison="default"/>
```

In this example, the first element is explicitly ignored during comparison, the second element is explicitly included.

Categories

Categories allow the definition of "filters", which can be used to restrict the display of any data model elements that are located in tables.

To create a category, add the attribute `osd:category` to a table node in the data model XSD.

Filters on data

In the example below, the attribute `osd:category` is added to the node in order to create a category named *mycategory*.

```
<xs:element name="rebate" osd:category="mycategory"/>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="label" type="xs:string"/>
      <xs:element name="beginDate" type="xs:date"/>
      <xs:element name="endDate" type="xs:date"/>
      <xs:element name="rate" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

To activate a defined category filter on a data set in the user interface, select **Actions > Categories > *<category name>*** from the navigation pane.

Predefined categories

Two categories with localized labels are predefined:

- Hidden
- An instance node, including a table node itself, is hidden in the default view, but can be revealed by selecting **Actions > Categories > [hidden nodes]** from the navigation pane.
- A table record node is always hidden.
- Constraint (deprecated)

Restriction

Category specification does not apply on table record node, except for category Hidden.

Java reference

CHAPTER 78

Mapping to Java

How to access data from Java?

Read-access to data is possible from various generic Java classes, mainly **Adaptation** [Adaptation]^{API} and **ValueContext** [ValueContext]^{API}. The getter methods return objects according to the mapping rules described in the next section.

Any update on data is possible only in a well-managed context:

- in the context of a procedure execution, by calling the methods `setValue...` of interface **ValueContextForUpdate** [ValueContextForUpdate]^{API};
- during user input validation, by calling the method `setNewValue` of class **ValueContextForInputValidation** [ValueContextForInputValidation]^{API}.

According to the mapping that is described in the next section, some accessed Java objects are mutable objects: these are instances of `List`, `Date` or any `JavaBean`. Consequently, these objects can be locally modified by means of their own methods. However such modifications will remain local to the returned object unless one of the above setters is invoked (and the current transaction is successfully committed).

Concurrency and isolation levels

Highest isolation level

The highest isolation level in ANSI/ISO SQL is `SERIALIZABLE`. Three execution modes guarantee the `SERIALIZABLE` isolation level, within the scope of a data space:

- if client code is run inside a `Procedure` container; this is the case for every update, XML, CSV or archive export and for data services;
- if client code accesses a data space that has been explicitly locked (see Java class `LockSpec`);
- if client code accesses a data space snapshot.

Default isolation level

If the client code is run outside the contexts enabling `SERIALIZABLE`, its isolation level depends on the persistence mode:

- In semantic mode, the default isolation level is `READ UNCOMMITTED`.
- In relational mode, the default isolation level is the database default isolation level.

Java access specificities

From a Java application, a record is represented by an instance of the class `Adaptation`, this object being initially linked to the corresponding persisted record. Then, unless the client code is executed in a context enabling the [SERIALIZABLE isolation level](#), the object can become "disconnected" from the persisted record: if concurrent updates have occurred they will not be reflected by the `Adaptation` object.

So, it is important for the client code to either be in a `SERIALIZABLE` context, or to regularly look up or refresh the `Adaptation` object.

See also:

- **`AdaptationHome.findAdaptationOrNull()`** [`AdaptationHome`]^{API}
- **`AdaptationTable.lookupAdaptationByPrimaryKey()`** [`AdaptationTable`]^{API}
- **`Adaptation.getUpToDateInstance()`** [`Adaptation`]^{API}

Mapping of datatypes

Basic rules for simple datatypes

Each XML Schema simple type corresponds to a Java class. The mapping is summarized in the following table.

Schema Datatype	Java class	Comment
xs:string	<code>java.lang.String</code>	
xs:boolean	<code>java.lang.Boolean</code>	
xs:decimal	<code>java.math.BigDecimal</code>	A <code>BigDecimal</code> is converted to double at display. Some precision loss and/or rounding problem can appear if the <code>BigDecimal</code> is made up of more than 15 digits. Moreover, the erroneous displayed value will replace the correct one if the form is submitted.
xs:date	<code>java.util.Date</code>	The returned <code>Date</code> is the beginning moment of each day, that is '00:00:00'.
xs:dateTime	<code>java.util.Date</code>	
xs:time	<code>java.util.Date</code>	The returned <code>Date</code> is always set to the following day: 1970/01/01.
xs:anyURI	<code>java.net.URI</code>	
xs:Name	<code>java.lang.String</code>	
xs:int	<code>java.lang.Integer</code>	
xs:integer	<code>java.lang.Integer</code>	Note that this mapping does not comply with the XML Schema recommendation. Indeed XML Schema specification states that <code>xs:integer</code> have no value space limitations but this value space is restricted by the Java specifications of the <code>java.lang.Integer</code> object.

Multiple cardinality on a simple element

If the attribute `maxOccurs` is greater than 1, then the element is an aggregated list and the corresponding instance in Java is an instance of `java.util.List`.

Elements of the list are instances of the Java class deduced from the rules in the table above.

Complex type definition without `osd:class` declaration

By default, a terminal node on a complex type is instantiated with an EBX5 internal class. This class provides a generic JavaBean implementation, but when Java access is needed, it is recommended to use a specific JavaBean, by using the `osd:class` declaration described in the next section.

Mapping of complex types to specific JavaBeans

It is possible to map a XML Schema complex type to a specific Java class. This is achieved by adding the attribute `osd:class` in the complex node definition. It is also required to specify the attribute `osd:access` (see example), so that the node is considered as *terminal*, unless the element has `xs:maxOccurs > 1` (because in this latter case, it is already considered as terminal).

The Java class must conform to the JavaBean protocol. This means that each child of the complex type must correspond to a JavaBean property of the class. Additionally each JavaBean property must be a read-write property, and its implementation must ensure that the value set by the setter method is returned as-is by the getter method (more particularly, contextual computations are not allowed in these methods).

Example:

```
<xs:element name="customer" osd:access="RW">
  <xs:complexType name="subscriber" osd:class="com.carRental.Customer">
    <xs:sequence>
      <xs:element name="firstName" type="xs:string"/>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In this example, the class Java `com.carRental.Customer` must define the methods `getFirstName()` and `setFirstName(String)`.

A JavaBean may have a specific user interface within EBX5. This is achieved using a **UIBeanEditor** [UIBeanEditor]^{API}.

Multiple cardinality on a complex element

If the attribute `maxOccurs` is greater than 1, then the corresponding instance in Java is:

- either an instance of `java.util.List` for an aggregated list, and every element in the list is an instance of the Java class deduced from the rules in the table above;
- or an instance of **AdaptationTable** [AdaptationTable]^{API}, if the property `osd:table` is specified.

Java bindings

Java bindings allow to generate Java types that reflect the structure of the data model. The Java code generation can be done in EBX5.

Benefits

Ensuring the link between XML Schema structure and Java code provides a number of benefits:

- Development assistance:
automatic completion when writing access path to parameters (if supported by your IDE) ;
- Access code verification:
all accesses to parameters are verified at code compilation ;
- Impact verification:
each modification of the data model impacts the code compilation state ;
- Cross-references:
by using the reference tools of your IDE, it is easy to verify where a parameter is used

Consequently, we strongly recommend to use Java bindings.

XML Declaration

The specification of the Java types to be generated from the data model is included in the main schema.

Each binding element defines a generation target. It must be located at `xs:schema/xs:annotation/xs:appinfo/ebxbnd:binding` (XPath notation) where the prefix `ebxbnd` is a reference to the naming space identified by the URI `urn:ebx-schemas:binding_1.0`. Several binding elements can be defined if you have different generation targets.

The attribute `targetDirectory` of the element `ebxbnd:binding` defines the root directory for Java types generation (generally, it is the directory of the project source codes `"src"`). A relative path is interpreted relatively to the current runtime directory of the VM (not relative to the XML schema).

See the [bindings XML Schema](#).

Example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ebxbnd="urn:ebx-schemas:binding_1.0">
  <xs:annotation>
    <xs:appinfo>
      <!-- The bindings define how this schema will be represented in Java.
      Several <binding> elements may be defined, one for each target. -->
      <ebxbnd:binding
        targetDirectory="../_ebx-demos/src-creditOnLineStruts-1.0/">
        <javaPathConstants typeName="com.creditonline.RulesPaths">
          <nodes root="/rules" prefix="" />
        </javaPathConstants>
        <javaPathConstants typeName="com.creditonline.StylesheetConstants">
          <nodes root="/stylesheet" prefix="" />
        </javaPathConstants>
      </ebxbnd:binding>
```



```
</xs:appinfo>
</xs:annotation>
...
</xs:schema>
```

Therefore, we can define Java constants for XML schema paths. For that, we generate one or several interfaces from a schema node (that can be the root node `" / "`). Hence, in the previous example, we generate two interfaces of java paths constants, one from node `" / rules "` and the other from node `" / stylesheet "` of the considered schema. Interfaces names are described by the tags *javaPathConstants* with the attribute *typeName* and the associated node is described by the tag *nodes* with the attribute *root*.

CHAPTER 79

Tools for Java developers

EBX5 provides the Java Developer with tools that facilitate usage of the EBX5 API, and integration with development environments:

Activating the development tools

To activate development tools, you need to run EBX5 in *development* mode. It is specified in [ebx.properties](#) with the following property:

```
backend.mode=development
```

Data model refresh tool

EBX5 offers the possibility to refresh XML Schema. This tool can be accessed using the *Actions* button from the *Administration* section.

Typical scenario: the developer modifies his data model (for example, he adds new nodes) by editing the XML Schema file. He wants to see this change in EBX5, without restarting the application server.

Button 'Generate Java'

When accessing an adaptation instance or an data model, the *Generate Java* tool is accessible using the *Actions* button. This tool generates the Java types specified by the [Bindings](#).

Path to a node

The field *Data path* is displayed in the documentation pane. This field indicates the path to the node. It can be useful when writing XPath formula.

Note: this field is always available for an administrator.

