EBX5

Product Documentation

EBX5 Version 5.6.1 Fix A

EBX5 includes:

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

Table of contents

Reference Manual

Administration Guide

Developer Guide

User Guide

Introduction

CHAPTER **1**

How EBX5 works

This chapter contains the following topics:

1.1 Product overview

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

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

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

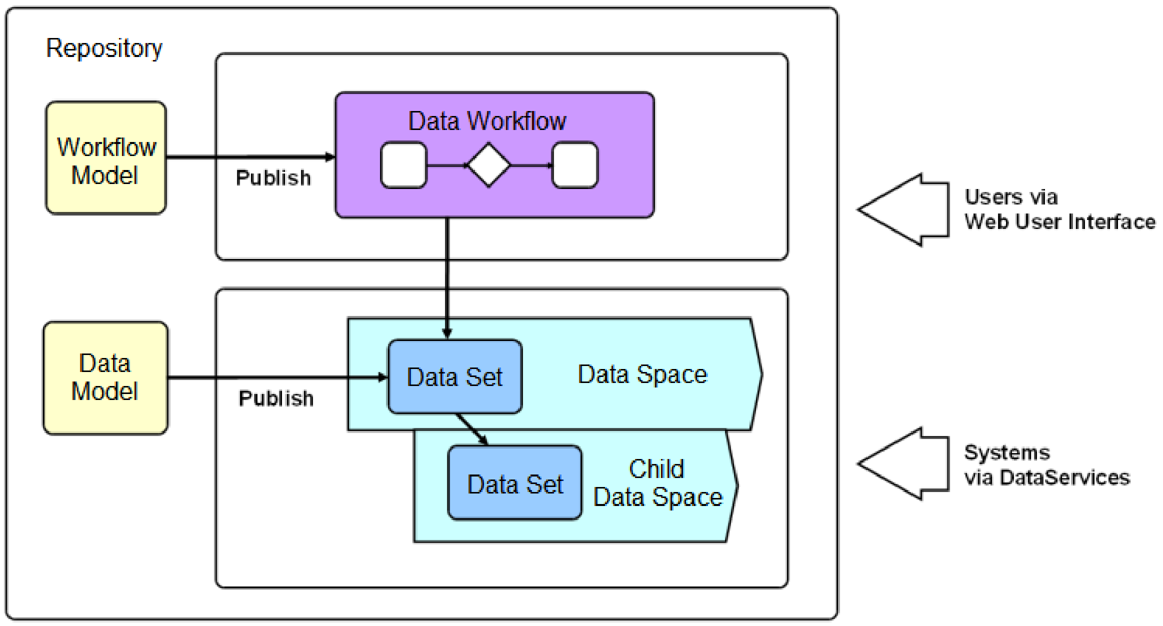
An MDM project using EBX5 starts with the creation of a *data model*. This is where the definition of tables, fields, links and business rules related to the master data takes place. Examples of modeled data include product catalogs, financial hierarchies, supplier lists or simple reference tables.

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

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

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

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

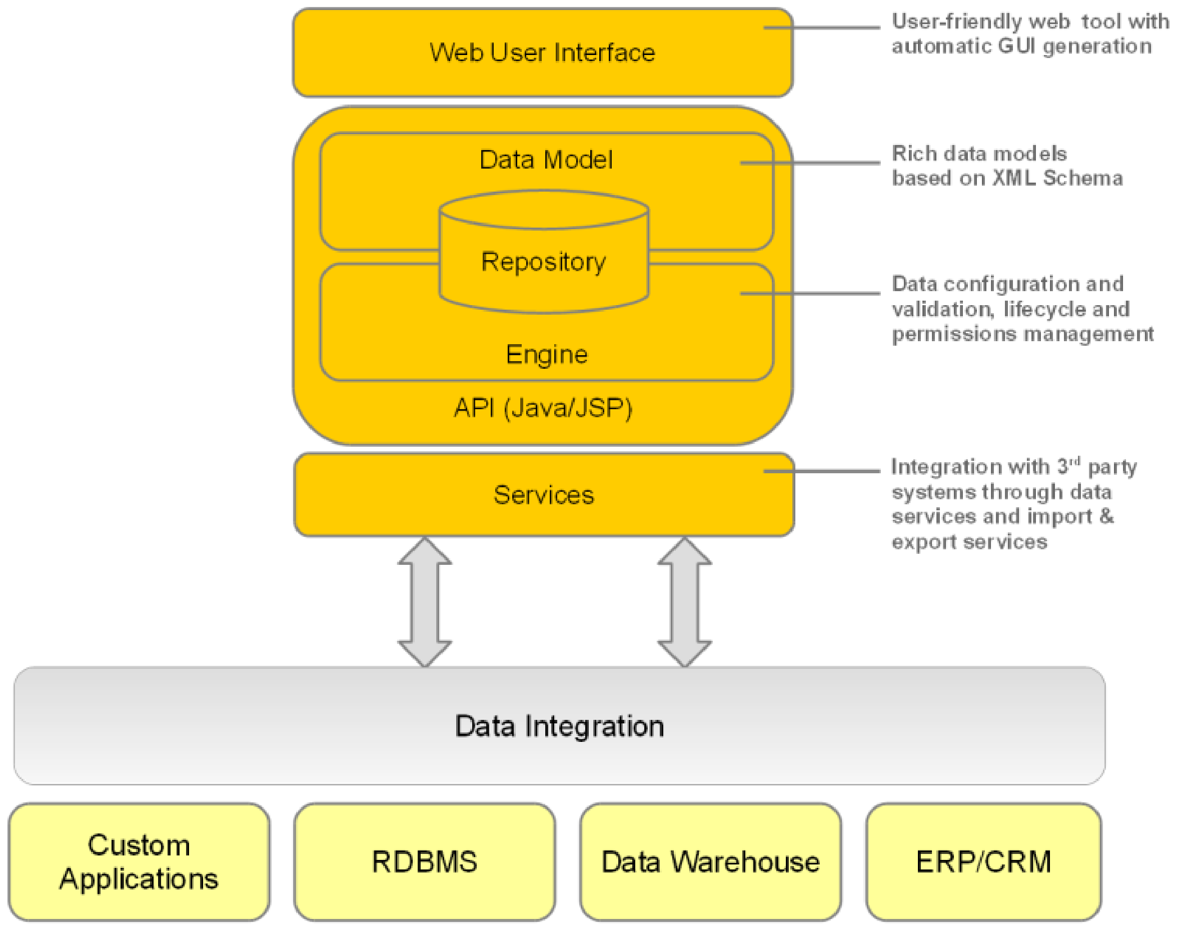


See also

- *Data modeling* [p 20]
- *Data sets* [p 22]
- *Data spaces* [p 23]
- *Workflow modeling* [p 25]
- *Data workflows* [p 26]
- *Data services* [p 27]

1.2 **EBX5 architecture**

The following diagram illustrates the EBX5 architecture.


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

CHAPTER 2

Using the EBX5 user interface

This chapter contains the following topics:

2.1 Overview

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

The advanced perspective is accessible by default.

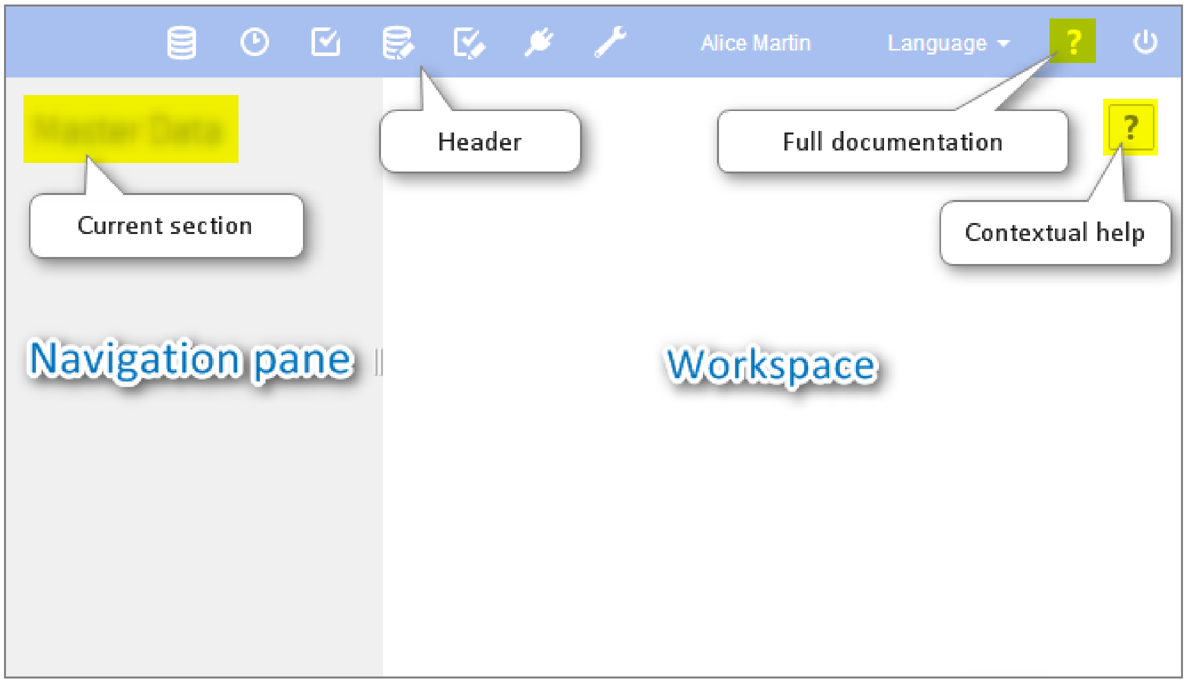
See also *Front end administration [p 377]*

2.2 Advanced perspective

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

- **Header:** Displays the user currently logged in, the user language preference selector (when more than one available), the perspective selector (when more than one available), a link to the product documentation, and a logout button.
- **Menu bar:** The functional categories accessible to the current user.
- **Navigation pane:** Displays context-dependent navigation options. For example: selecting a table in a data set, or a work item in a workflow.
- **Workspace:** Main context-dependent work area of the interface. For example, the table selected in the navigation pane is displayed in the workspace, or the current work item is executed in the workspace.

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



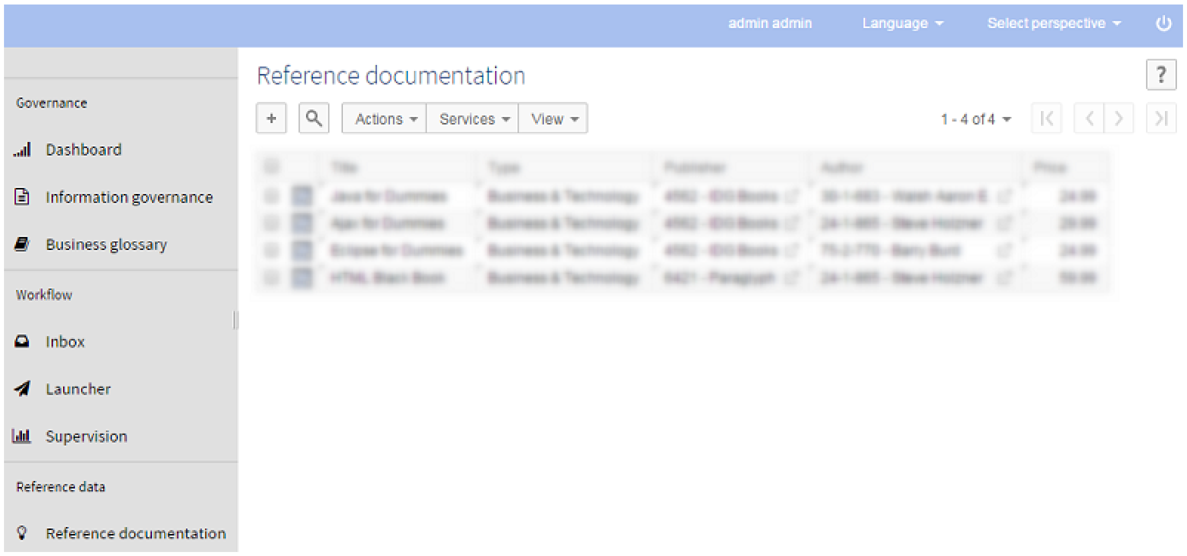
2.3 Perspectives

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

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

Perspectives are configured by authorized users. For more information on how to configure a perspective, see perspective administration [p 379].

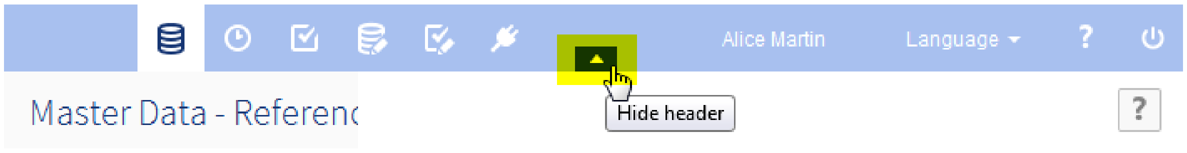
Example of a hierarchical menu:



2.4 User interface features

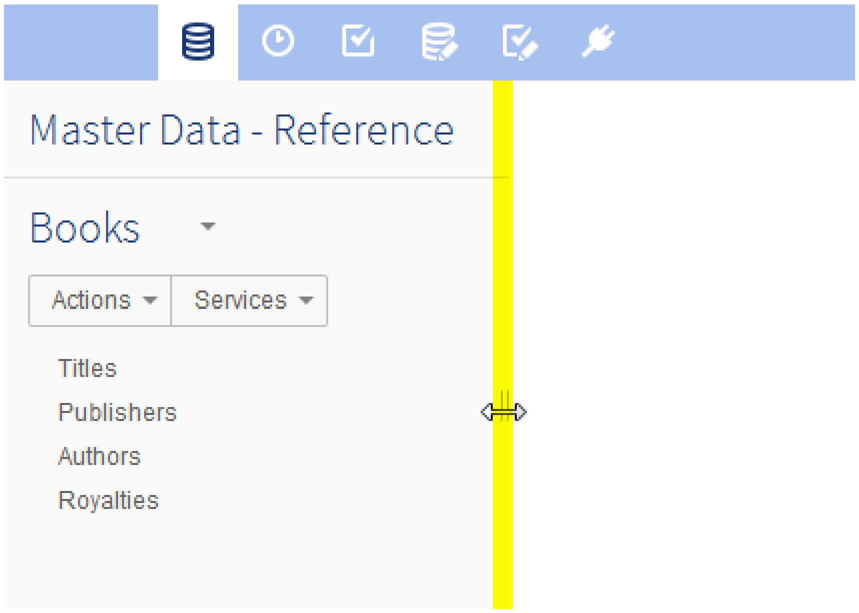
Hiding the header

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



Resetting the navigation pane width

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

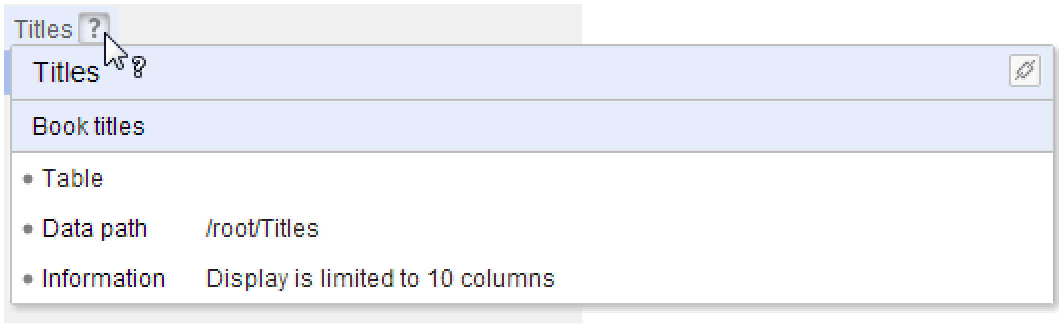


2.5 **Where to find EBX5 help**

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

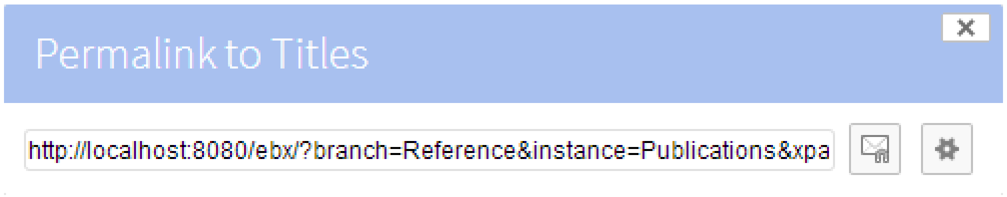
Contextual help

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



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

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



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

CHAPTER **3**

Glossary

This chapter contains the following topics:

3.1 **Governance**

repository

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

See also data space [p 23].

profile

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

See also user [p 19], role [p 20].

Related Java API `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 [p 20], profile [p 19].

Related concept User and roles directory [p 391].

Related Java API `UserReference`API.

role

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

Whenever a role profile is specified in 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 [p 20], profile [p 19].

Related concept User and roles directory [p 391].

Related Java API `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 [p 19], role [p 20].

Related concept User and roles directory [p 391].

Related Java API `Directory`API, `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 [p 391].

Related Java API `Session`API.

3.2 Data modeling

Main documentation section Data models [p 30]

data model

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

See also data set [p 22].

Related concept Data models [p 30].

field

A data model element that is defined with a name and a simple datatype. A field can be included in the data model directly or as a column of a table. In EBX5, fields can be assigned basic constraints, such as length and size, as well as more complex validation rules involving computations. Automated value assignment using field inheritance or computations based on other data can also be defined for fields. Aggregated lists can be created by setting the cardinality of a field to allow multiple values in the same record. Fields can be arranged into groups to facilitate structural organization in the data model.

By default, fields are denoted by the icon .

See also record [p 22], group [p 21], table (in data model) [p 21], validation rule [p 22], inheritance [p 23].

Related concepts Structure elements properties [p 47], Controls on data fields [p 59].

Related Java API `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 [p 451].

foreign key

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

Foreign keys are denoted by the icon .

See also primary key [p 21].

Related concept Constraints, triggers and functions [p 455].

table (in data model)

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

Tables are represented by the icon .

See also record [p 22], primary key [p 21], reusable type [p 22].

group

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

Groups are represented by the icon .

See also reusable type [p 22].

Related Java API `SchemaNode`API.

reusable type

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

validation rule

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

data model assistant (DMA)

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

See also Data models [p 30].

3.3 Data sets

Main documentation section Data sets [p 90]

record

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

Records are represented by the icon .

See also table (in data set) [p 22], primary key [p 21].

table (in data set)

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

Tables are represented by the icon .

See also record [p 22], primary key [p 21].

data set

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

Data sets are represented by the icon .

See also table (in data set) [p 22], field [p 21], group [p 21], custom table view [p 23].

Related concept Data sets [p 90].

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 [p 296].

custom table view

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

See also hierarchical table view [p 23].

Related concept Views [p 97].

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 [p 23].

Related concept Hierarchies [p 98].

3.4 Data management life cycle

Main documentation section Data spaces [p 72]

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 [p 23], repository [p 19], data space merge [p 24].

Related concept Data spaces [p 72].

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 [p 23], data space merge [p 24], repository [p 19].

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 [p 80].

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 [p 85]

3.5 History

Main documentation section History [p 275]

historization

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

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

See also table history view [p 25], transaction history view [p 25], history profile [p 24].

history profile

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

See also history profile [p 24].

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 [p 275].

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 [p 275].

3.6 Workflow modeling

Main documentation section Workflow models [p 114]

workflow model

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

Related concept Workflow models [p 114].

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 [p 25].

user task

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

User tasks are represented by the icon 👤.

See also workflow model [p 25].

workflow condition

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

Workflow conditions are represented by the icon ◇.

sub-workflow invocation

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

wait task

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

data context

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

3.7 Data workflows

Main documentation section Data workflows [p 140]

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 [p 25].

Related concept Data workflows [p 140].

work list

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

work item

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

Allocated work items are represented by the icon .

See also user task [p 25].

token

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

3.8 **Data services**

Main documentation section Data services [p 156]

data service

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

Data services offer:

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

lineage

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

Related concept: Generating a WSDL for lineage [p 161].

3.9 **Cross-domain**

node

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

- In the workflow model [p 25] context, a node is a workflow step or condition.
- In the data model [p 20] context, a node is a group, a table or a field.
- In the hierarchy [p 23] context, a node represents the value of a dimension.
- In an adaptation tree [p 23], a node is a data set.
- In a data set [p 22], a node is the node of the data model valuated in the context of the data set or the record.

Data models

CHAPTER 4

Introduction to data models

This chapter contains the following topics:

4.1 Overview

What is a data model?

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

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

Basic concepts used in data modeling

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

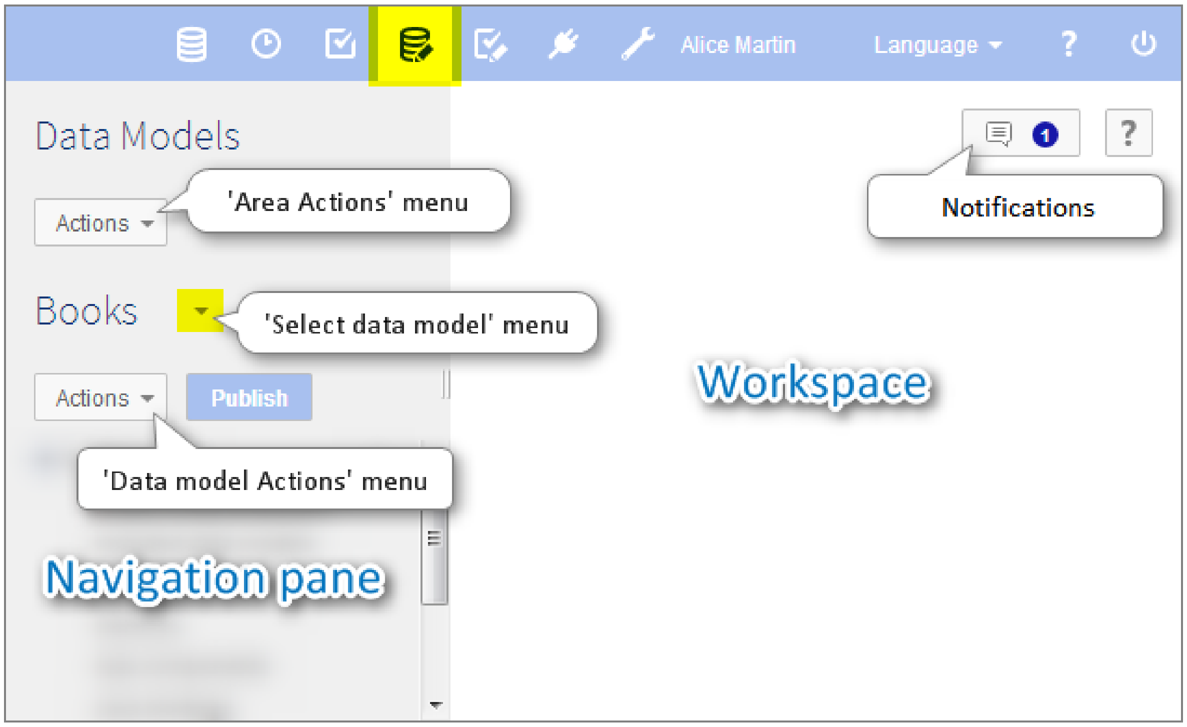
- field [p 21]
- primary key [p 21]
- foreign key [p 21]
- table (in data model) [p 21]
- group [p 21]
- reusable type [p 22]
- validation rule [p 22]

4.2 Using the Data Models area user interface

Navigating within the Data Model Assistant

Data models can be created, edited or imported, and published in the **Data Models** area of the user interface. The EBX5 data model assistant (DMA) facilitates the development of data models.

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



The navigation pane is organized into the following sections:

Configuration	The technical configuration of the data model.
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.

See also

Implementing the data model structure *[p 41]*

Configuring the data model *[p 37]*

Reusable types *[p 43]*

Data model element icons

field [p 21]

primary key [p 21]

foreign key [p 21]

table [p 21]

group [p 21]

Related concepts

Data spaces [p 72]

Data sets [p 90]

CHAPTER **5**

Creating a data model

This chapter contains the following topics:

5.1 **Creating a new data model**

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

5.2 **Selecting a data model type**

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

Semantic models

Semantic models enable the full use of data management features, such as life cycle management using 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 *[p 269]*

Data spaces *[p 72]*

CHAPTER 6

Configuring the data model

This chapter contains the following topics:

6.1 Information associated with a data model

To view and edit the owner and documentation of your data model, select 'Information' from the **data model 'Actions'** [p 31] menu for your data model in the navigation pane.

Note

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

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

6.2 Permissions

To define the user permissions on your data model, select 'Permissions' from the **data model 'Actions'** [p 31] menu for your data model in the navigation pane.

The configuration of the permissions of a data model are identical to the options for the permissions of a data set, as explained in Permissions [p 105].

6.3 **Data model properties**

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

Module name	Defines the module that contains the resources that will be used by this data model. This is also the target module used by the data model publication if publishing to a module.
Module path	Physical location of the module on the server's file system.
Sources 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	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. See Publication modes [p 67] for more information. **Model path in module**: Defines the target file for the data model generation. It must start with '/'.
Data set inheritance	Specifies whether data set inheritance is enabled for this data model. Data set inheritance is disabled by default. See Data set inheritance [p 109] for more information.
Documentation	Documentation of the data model defined by a Java class. This Java class can programmatically specify labels and descriptions for the elements of the data model. The labels and descriptions defined in this Java class are displayed in associated data sets in preference to the ones defined locally on an element. See Dynamic labels and descriptions [p 488] for more information.
Special extensions	Access permissions defined by programmatic rules in a Java class.
Disable auto-increment checks	Specifies whether to disable if the check of an auto-incremented field value in associated data sets regarding to the "max value" found in the table being updated. See Auto-incremented values [p 484] for more information.

6.4 **Included data models**

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

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

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

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

See also *Including external data models* *[p 448]*

6.5 **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.
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	Specifies the data synchronization policy. The possible policies are: • **On commit**: The replicated table content in the database is always up to date with respect to its source table. Every transaction that updates the 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	Specifies the tables in the data model to be replicated in the database. **Table path**: Specifies the path of the table in the current data model that is to be replicated to the database. **Table name in database**: Specifies the name of the table in the database to which the data will be replicated. This name must be unique amongst all replications units.

See also *Replication* *[p 283]*

CHAPTER 7

Implementing the data model structure

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

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

This chapter contains the following topics:

7.1 Common actions and properties

Adding elements to the data model

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

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

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

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

Names, labels, descriptions, and information

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

You have the option to provide localized user-friendly labels to be displayed in the user interface instead of the unique name of the element, as well as brief localized descriptions of the element. Unlike the unique name, the labels and descriptions are modifiable after creation. According to the language preference of each user, EBX5 will display the corresponding localized label and description of the element.

Deleting elements of the data model

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

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

Duplicating existing elements

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

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

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

Moving elements

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

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

7.2 Reusable types

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

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

Defining a reusable type

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

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

Using a reusable type

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

Including data types defined in other data models

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

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

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

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

7.3 **Data model element creation details**

Creating fields

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

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

Creating tables

While creating a table, you have the option to create the new table based on an existing reusable type. See Reusable types [p 43] for more information.

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

Creating groups

While creating a group, you have the option to create the new group based on an existing reusable type. See Reusable types [p 43] for more information.

Creating primary key fields

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

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

Creating or defining foreign key fields

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

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

Creating associations

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

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

- Inverse relationship of a *foreign key*. In this case, the association element is defined in a *source table* and refers to *a target table*. It is the counterpart of the foreign key field, which is defined in the target table and refers back the source table. You must define the foreign key that references the parent table of the association.
- Over a *link table*. In this case, the association element is defined in a *source table* and refers to a *target table* that is inferred from a *link table*. This link table defines two foreign keys: one referring to the source table and another one referring to the target table. The primary key of the link table must also refer to auto-incremented fields and/or the foreign key to the source or target table of the association. You must define the link table and these two foreign keys.

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

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

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

7.4 Modifying existing elements

Removing a field from the primary key

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

See primary key [p 21] in the glossary.

CHAPTER **8**

Properties of data model elements

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

See also *Data validation controls on elements [p 59]*

This chapter contains the following topics:

8.1 **Basic element properties**

Common basic properties

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

Information	Additional non-internationalized information associated with the element.
Minimum number of values	Minimum number of values for an element. As primary keys cannot be multi-valued, they must have this property set to '1' or 'undefined'. The minimum number of values is automatically set to '0' when defining the field as a selection node.
Maximum number of values	Maximum number of values for an element. When set to a value greater than '1', the element becomes multi-valued. As primary keys cannot be multi-valued, they must have this property set to '1' or 'undefined'. For tables, the maximum number of values is automatically set to 'unbounded' upon creation. The maximum number of values is automatically set to '0' when defining the field as a selection node.
Validation rules	This property is available for tables and fields in tables except `Password` fields, reusable types, fields in complex reusable types, and selection nodes. Used to define powerful and complex validation rules with the help of the provided XPath 1.0 criteria editor. See Criteria editor [p 315] for more information. This can be useful if the validation of the value depends on complex criteria or on the value of other fields. Using the associated wizard, you can define localized labels for the validation rule, as well as define a localized message with severity to be displayed if the criteria is not met. If a validation rule is defined on a table, it will be considered as a 'constraint on table' and each record of the table will be evaluated against it at runtime. See Constraints on table [p 477] for more information.

Basic properties for fields

The following basic properties are specific to fields:

Default value	Default value assigned to this field. In new data creation forms, the default value appears automatically in the user input field. The default value must comply with the defined type of the field. See Default value [p 493] for more information.
Conversion error message	Internationalized messages to display to users when they enter a value that is invalid for the data type of this field.
Computation rule	This property is available for fields in tables, except in reusable types. Defines a rule for computing the value of the field using the provided XPath 1.0 editor. See criteria editor [p 315] This can be useful if the value depends on other values in the same record, but does not require a programmatic computation. The following limitations exist for computation rules: • Computation rules can only be defined on simple fields inside a table. • Computation rules cannot be defined on fields of type `OResource` or `Password`. • Computation rules cannot be defined on selection nodes and primary key fields. • Computation rules cannot be defined when accessing an element from the validation report.

8.2 **Advanced element properties**

Common advanced properties

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

UI bean	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`API class in the Java API.
Transformation on export	This property is available for fields and for groups that are terminal nodes. Specifies a Java class that defines transformation operations to be performed when exporting an associated data set as an archive. The input of the transformation is the value of this element. See `NodeDataTransformer`API for more information.
Access properties	Defines the access mode for the current element, that is, if its data can be read and/or written. • 'Read & Write' corresponds to the mode `RW` in the data model XSD. • 'Read only' corresponds to the mode `R-` in the data model XSD. • 'Not a data set node' corresponds to the mode `CC` in the data model XSD. • 'Non-terminal node' corresponds to the mode `--` in the data model XSD. See Access properties [p 493] for more information.
Default view	Using the default view property, it is possible to specify: • Whether or not this element is shown in the default view of a data set, in the text search of a data set or in the data service "select" operation. This property is ignored if it is set on an element that is not in a table. Setting the property to 'Hidden' will hide the element from the default view of a table in a data set without having to define specific access permissions. • Whether or not this element is shown in a data set search tools. Setting the property to 'Hidden in all searches' will hide the element both in the text and typed search tools of a data set. Setting the property to 'Hidden only in text search' will only hide the element in the text

	search tool. If this property is not set, the element will be displayed in the text search tool by default. • Whether or not this element is shown in the data service `select` operation. Setting the property to 'Excluded from Data Services' will hide the element in the data service `select` operation. See Default view [p 495] for more information.
Default view > Combo-box selector	Specifies the name of the published view that will be used in the combo-box selection of the foreign key. A selection button will be displayed at the bottom right corner of the drop-down list. When defining a foreign key, this feature allows accessing an advanced selection view through the 'Selector' button that opens the advanced selection view, from where sorting and searching options can be used. If no published view is defined, the advanced selection view will be disabled. If the path of the referenced table is absolute then only the published views corresponding to this table will be displayed. If the path of the referenced table is relative then all the published views associated with the data model containing the target table will be displayed. See Defining a view for the combo box selector of a foreign key [p 496] 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 [p 498] for more information.

Apply last modifications policy	Defines if this element must be excluded from the service allowing to apply the last modifications that have been performed on a record to the other records of the same table. • 'Default' means that the last modification on this element can be applied to other records. • 'Ignored' implies that the last modification on this element cannot be applied to other records. This element will not be visible in the apply last modifications service. See Apply last modifications policy [p 498] 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 [p 499] 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 [p 480] for more information.

UI bean

See Common advanced properties [p 50].

Function (computed value)

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

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

Transformation on export

See Common advanced properties [p 50].

Access properties

See Common advanced properties [p 50].

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 [p 465] for more information.

Auto-increment

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

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

Auto-incremented values have the following behavior:

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

  That is, the uniqueness of the allocation spans over all 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 mass update transaction assigning specific values is performed concurrently with a transaction that allocates an auto-incremented value on the same field. It is possible that the latter transaction will allocate a value that has already been set in the former transaction, as there is no locking between different data spaces.

See Auto incremented values [p 484] for more information.

Default view

See Common advanced properties [p 50].

Node category

See Common advanced properties [p 52].

Inherited field

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

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

See inheritance [p 23] in the glossary.

For more information, see also Inherited fields [p 299].

Advanced properties for tables

The following advanced properties are specific to tables.

Table

Primary key	A list of fields in the table that compose the table's primary key. You can add or remove primary key fields here, as in the 'Data structure' view. Each primary key field is denoted by its absolute XPath notation that starts under the table's root element. If there are several elements in the primary key, the list is white-space delimited. For example, "/name /startDate".
Presentation	Specifies how records are displayed in the user interface of this table in a data set.
***Presentation* > Record labeling**	Defines the fields to provide the default and localized labels for records in the table. Can also specify a Java class to set the label programmatically, or set the label in a hierarchy. This Java class must implement either the `UILabelRenderer`API interface or the `UILabelRendererForHierarchy`API interface. **Attention:** Access rights defined on associated data sets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
***Presentation* > Default rendering for groups in forms**	Specifies the default display rendering mode of the groups contained in this table. If nothing is defined here, the default policy set in the Administration area will be used to display groups. See Ergonomics & behavior policy [p 386] for more information. **Enabled rendering for groups** Specifies a display rendering mode to be enabled for groups in the table in addition to the modes 'expanded' and 'collapsed', which are always available. Tabs must be enabled on the table to have the option to display groups as tabs. Similarly, links must be enabled to have the option to display groups as links. **Default rendering for groups** Specifies the default display rendering mode to use for the groups contained in this table. If a group does not specify a default mode then the default mode defined for this table will be used. Links must be enabled to define the default rendering mode as 'Link'. Select a rendering mode according to network and browser performance. Link mode is lighter

	as its content is not displayed on the same page, whereas the expanded and collapsed modes are heavier. **Note:** When the tabs view is enabled on a table, any groups that would otherwise have used links are automatically converted to collapsed mode. This is done to avoid the inherent display layout complexities that would arise from links and tabs coexisting in the same user interface.
***Presentation* > Specific rendering of forms**	Defines a specific rendering for customizing the record form in a data set.
History	Specifies when historization is to be performed, and the level of guarantee requested. The available history profiles can be edited in **Administration > History and logs**. See History configuration in the repository [p 275] for more information.
Indexes	Defines the fields to index in the table. Indexing speeds up table access for requests on the indexed fields. No two indexes can contain the exact same fields. **Index name**: Unique name for this index. **Fields to index**: The fields to index, each represented by an absolute XPath notation starting under the table `root` element.
Specific filters	Defines record display filters on the table.
Actions	Specifies the actions that are allowed on the table in associated data sets. By default, all actions are allowed unless specific access rights are defined in a data set.

Uniqueness constraints

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

Triggers

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

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

See Triggers [p 483] for more information.

Access properties

See Common advanced properties [p 50].

Default view

See Common advanced properties [p 50].

Node category

See Common advanced properties [p 52].

Advanced properties for groups

The following advanced properties are specific to groups.

Value container class (JavaBean)

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

UI bean

See Common advanced properties [p 50].

Transformation on export

See Common advanced properties [p 50].

Access properties

See Common advanced properties [p 50].

Default view

Visibility	See Common advanced properties [p 50].
Rendering	Defines the rendering mode of this group. If this property is not set, then the default view for groups specified by the container table will be used. 'Tab' and 'Link' are each only available when the container table enables it. ***Tab properties*** **Position:** This attribute specifies the position of the tab with respect to all the tabs defined in the model. This position is used for determining tab order. If a position is not specified, the tab will be displayed according to the position of the group in the data model.

Node category

See Common advanced properties [p 52].

Related concepts *Data validation controls on elements [p 59]*

CHAPTER **9**

Data validation controls on elements

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

See also *Properties of data model elements [p 47]*

This chapter contains the following topics:

9.1 Simple content validation

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

Fixed length	The exact number of characters required for this field.
Minimum length	The minimum number of characters allowed for this field.
Maximum length	The maximum number of characters allowed for this field.
Pattern	A regular expression pattern that the value of the field must match. It is not possible to simultaneously define a pattern for both a field and its data type.
Decimal places	The maximum number of decimal places allowed for this field.
Maximum number of digits	The maximum total number of digits allowed for this integer or decimal field. For fields of type 'Decimal', the default maximum length is 15. This is due to the fact that if the value exceeds 15 digits, loss of precision or rounding problems may occur when it is displayed in the user interface. Furthermore, if such an inaccurate value is displayed in a form and that form is submitted, the incorrect value will replace the original value.
Enumeration	Defines a list of predefined possible values for this field. If enumerations are defined in both a field and its type, then the enumeration of this field in associated 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 [p 469].

9.2 Advanced content validation

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

See also *Dynamic constraints [p 473]*

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

Fixed length [dynamic]	Defines a field to provide the exact number of characters required for this field.
Minimum length [dynamic]	Defines a field to provide the minimum number of characters allowed for this field.
Maximum length [dynamic]	Defines a field to provide the maximum number of characters allowed for this field.
Excluded values	Defines a list of values that are not allowed for this field.
Excluded segment	Defines an inclusive range of values that are not allowed for this field. **Minimum excluded value:** Lowest value not allowed for this field. **Maximum excluded value:** Highest value not allowed for this field.
Specific constraint (component)	Specifies one or more Java classes that implement the `Constraint`API interface of the Java API. See Programmatic constraints [p 476] for more information.
Specific enumeration (component)	Specifies a Java class to define an enumeration. The class must define an ordered list of values by implementing the `ConstraintEnumeration`API interface of the Java API.
Enumeration filled by another node	Defines the possible values of this enumeration using a reference to another list or enumeration element.

Validation properties

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

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

Related concepts *Properties of data model elements* *[p 47]*

CHAPTER **10**

Working with an existing data model

Once your data model has been created, you can perform a number of actions that are available from the **data model 'Actions'** [p 31] menu in the workspace.

This chapter contains the following topics:

10.1 Validating a data model

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

Note

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

See Validation [p 322] for detailed information on incremental data validation.

10.2 XML Schema Document (XSD) import and export

EBX5 includes built-in data model services to import from and export to XML Schema Document (XSD) files. XSD imports and exports can be performed from the **data model 'Actions'** [p 31] menu of the target data model in the navigation pane. An XSD import or export is always performed in the context of a single data model. That is, during imports, the structure of the target data model is completely replaced with the content of the imported XSD. Similarly, during exports, the entire data model is included in the XSD file.

When importing an XSD file, the file must be well-formed and must comply with EBX5 validation rules. If this document declares resources that are located in a module, the module must also be declared in the configuration of the data model. If the module has not been declared, you will not be able to import the XSD file. See Data model properties [p 38] for more information on declaring modules.

To perform an import select 'Import XSD' from the **data model 'Actions'** [p 31] 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'.

10.3 Duplicating a data model

To duplicate a data model, select 'Duplicate' from the **data model 'Actions'** [p 31] menu for that data model. You must give the new data model a name that is unique in the repository.

10.4 Deleting a data model

To delete a data model, select 'Delete' from the **data model 'Actions'** [p 31] menu for that data model. When you delete a data model, all of its existing publications will remain and continue to be accessible to their associated data sets. If you recreate a new data model with the same name as one that was previously deleted, the new data model will be reassociated with all the existing publications in the repository. At publication time of the new data model, you will have the opportunity to confirm the replacement of an existing publication.

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

See Publishing data models [p 67] for more information on the publication process.

CHAPTER 11

Publishing a data model

This chapter contains the following topics:

11.1 About publications

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

> **Note**
>
> The **Publish** button is only displayed to users who have permission to publish the data model. See Data model permissions [p 37] for more information.

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

11.2 Publication modes

You can publish a data model using either 'Embedded' mode or 'In module' mode. The 'Embedded' publication mode generates a publication that is managed and persisted within the EBX5 repository and thus has associated versioning and rollback functionality. The 'In module' publication mode creates an XML Schema Document contained in a module that is not managed or versioned within the repository.

Depending on the configuration of the data model, EBX5 automatically determines the publication process to use when you click the **Publish** button in the navigation pane. When a data model specifies the publication mode 'In module' and provides a target XSD to be generated, the publication process generates an XSD file contained in the module defined in the configuration.

11.3 Embedded publication mode

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

See Viewing and creating publications [p 68] for more information on the use of different publications.

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

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

> **Note**
>
> Snapshots, which are static archives of the state of the data model, must not be confused with data model *versions*, which act instead as parallel evolving branches of the data model. See Versioning embedded data models [p 69] for more information on data model versions.

Viewing and creating publications

To access the publications that exist for the current data model, select 'Manage publications' from its **data model 'Actions'** [p 31] 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'** [p 31] 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 12

Versioning a data model

This chapter contains the following topics:

12.1 About versions

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

12.2 Accessing versions

To see the existing versions of your data model, select 'Manage versions' from the **data model 'Actions'** [p 31] menu of the data model.

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

12.3 **Working with versions**

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

Access data model version	Go to the corresponding version of the data model.
Create version	Creates a new version based on the contents of the selected version. The new version is added as a child of the selected version, though its contents bear no relation to those of its parent version after creation.
Set as default version	Sets the selected version as the default version opened when users access the data model.
Export archive	Exports the selected data model version to an archive containing the version's content, including its permissions and information. The exported archive is located in the archives directory, which is accessible to repository administrators. Exporting to an existing archive name will overwrite the existing file. See Archives directory [p 369] 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.

12.4 **Known limitations on data model versioning**

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

Data spaces

CHAPTER **13**

Introduction to data spaces

This chapter contains the following topics:

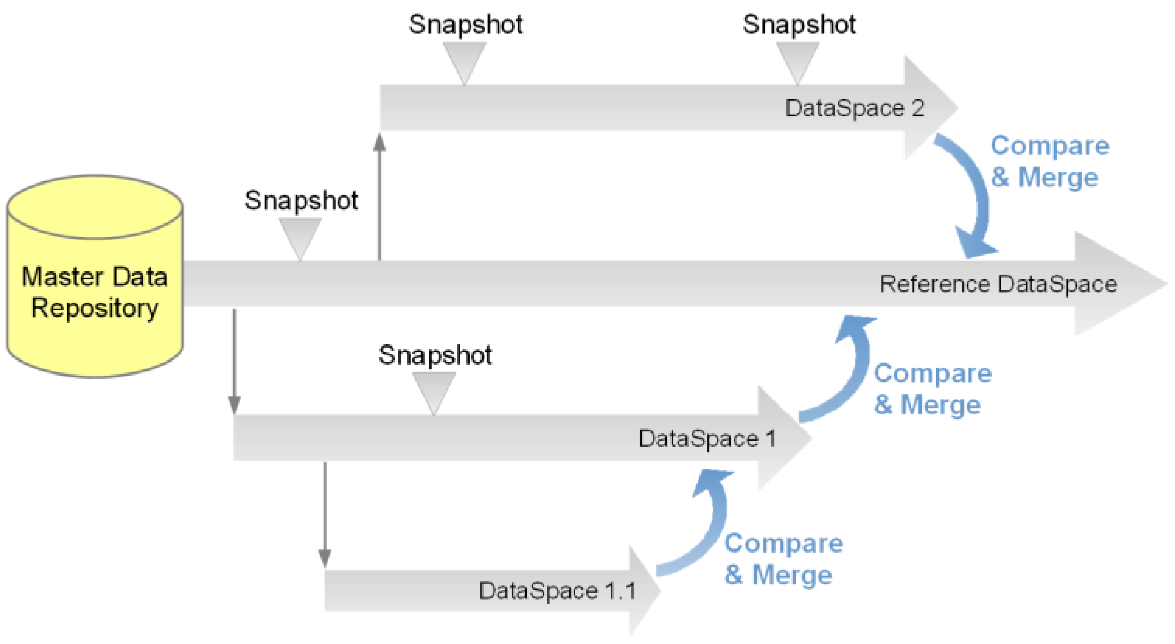
13.1 **Overview**

What is a data space?

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

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

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



Basic concepts related to data spaces

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

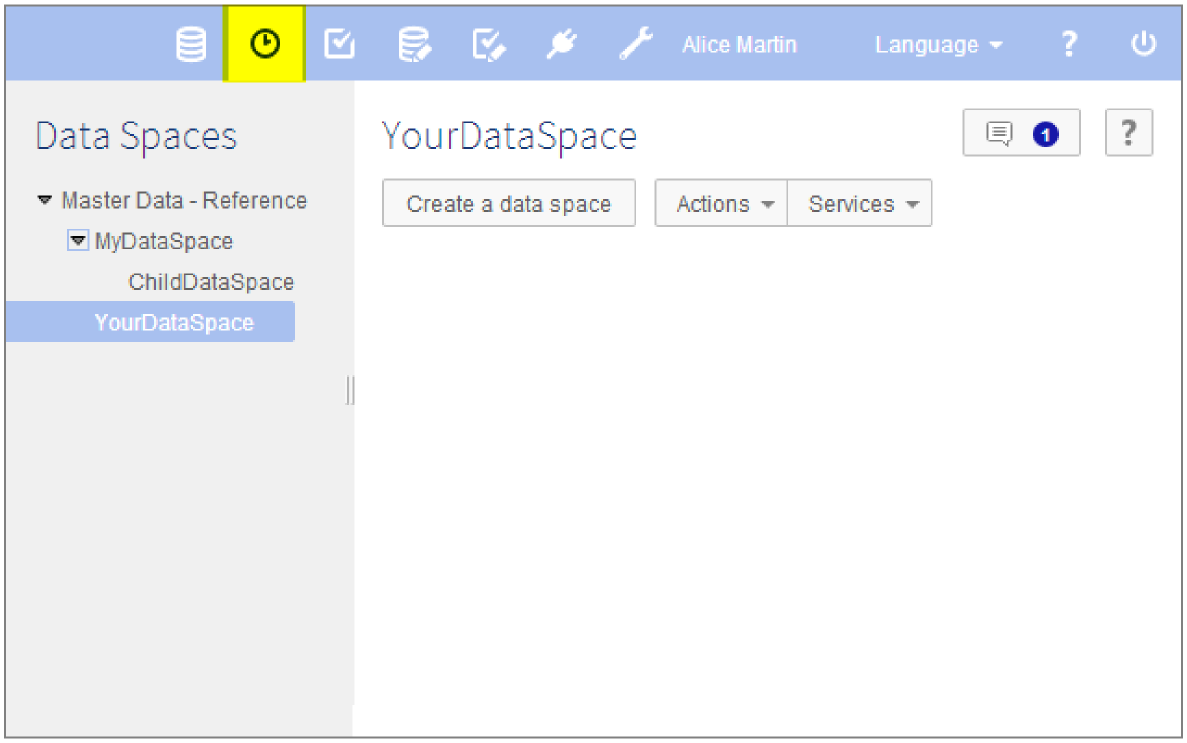
- data space [p 23]
- snapshot [p 24]
- data set [p 22]
- data space merge [p 24]
- reference data space [p 24]

13.2 **Using the Data Spaces area user interface**

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

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

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



See also

Creating a data space [p 75]

Snapshots [p 85]

Related concepts *Data sets* [p 90]

CHAPTER **14**

Creating a data space

This chapter contains the following topics:

14.1 **Overview**

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

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

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

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

Aside from the reference data space, which is the root of all semantic data spaces in the repository, semantic data spaces are always a child of another data space.

See also *Relational mode [p 76]*

14.2 **Properties**

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

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

14.3 **Relational mode**

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

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

See also *Relational mode* *[p 269]*

CHAPTER **15**

Working with existing data spaces

This chapter contains the following topics:

15.1 Data space information

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

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

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

15.2 **Data space permissions**

General permissions

Data space id	The data space to which the permissions will apply.
Profile selection	The profile to which the rule applies.
Restriction policy	Whether these permissions restrict the permissions assigned to a given user through policies defined for other profiles. See Restriction policy [p 309].
Data space access	The global access permission on the data space. **Read-only** • Can see the data space and its snapshots, as well as child data spaces, according to their permissions. • Can see the contents of the data space depending on their permissions; cannot make modifications. **Write** • Can see the data space and its snapshots, as well as child data spaces, according to their permissions. • Can modify the contents of the data space depending on their permissions. **Hidden** • Cannot see the data space nor its snapshots directly. • From a child data space, the current data space can be seen but not selected. • Cannot access the contents of the data space. • Cannot perform any actions on the data space.

Allowable actions

Users can be allowed to perform the following actions:

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

15.3 **Merging a data space**

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

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

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

Initiating a merge

To merge a data space into its parent data space:

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

Reviewing and accepting changes

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

> **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.

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

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

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

Types of modifications

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

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

Types of conflicts

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

Conflicts are categorized as follows:

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

Finalizing a merge

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

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

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

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

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

Setting the child merge policy of a data space

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

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

See also *Child merge policy* *[p 82]*

Abandoning a merge

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

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

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

15.4 **Comparing a data space**

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

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

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

See also *Compare contents [p 188]*

15.5 **Validating a data space**

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

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

15.6 **Data space archives**

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

Exporting

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

> **Note**
>
> See Archives directory [p 369] 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. It may be useful to export only the differences between the data space and its initial snapshot using a change set. There are two different export options that include a change set: 'The updates with their whole content' and 'The updates only'. The first option exports all current data and a change set containing differences between the current state and the initial snapshot. The second option only exports the change set. Both options lead to a comparison page, where you can select the differences to include in this change set. Differences are detected at the table level.
Data sets to export	The data sets to export from this data space. For each data set, you can export its data values, permissions, and/or information.

Importing

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

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

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

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

15.7 **Closing a data space**

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

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

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

See also *Closing unused data spaces and snapshots* *[p 372]*

CHAPTER 16

Snapshots

This chapter contains the following topics:

16.1 Overview of snapshots

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

See also *Snapshot* *[p 24]*

16.2 Creating a snapshot

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

The following information is required:

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

16.3 Viewing snapshot contents

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

16.4 Snapshot information

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

Documentation	Localized labels and descriptions associated with the snapshot.
Loading strategy	*Only administrators can modify this setting.* See Loading strategy [p 78].
Current Owner	Owner of the snapshot, who is, by default, allowed to modify its information and permissions. The owner does not necessarily have to be the creator of the snapshot.
Change owner	Whether the current owner of the snapshot is allowed to give ownership to another profile by modifying its 'Current owner' property. If the value is 'Forbidden', only an administrator can modify the owner.

16.5 Comparing a snapshot

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

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

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

See also *Compare contents [p 188]*

16.6 Validating a snapshot

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

Note

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

16.7 **Export**

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

> **Note**
>
> See Archives directory [p 369] 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.
Data sets to export	The data sets to export from this snapshot. For each data set, you can choose whether to export its data values, permissions, and information.

16.8 **Closing a snapshot**

If a snapshot is no longer needed, it can be closed. Once it is closed, a snapshot no longer appears under its associated data space in the Data Spaces area, nor can it be accessed.

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

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

See also *Closing unused data spaces and snapshots [p 372]*

Data sets

CHAPTER 17

Introduction to data sets

This chapter contains the following topics:

17.1 Overview

What is a data set?

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

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

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

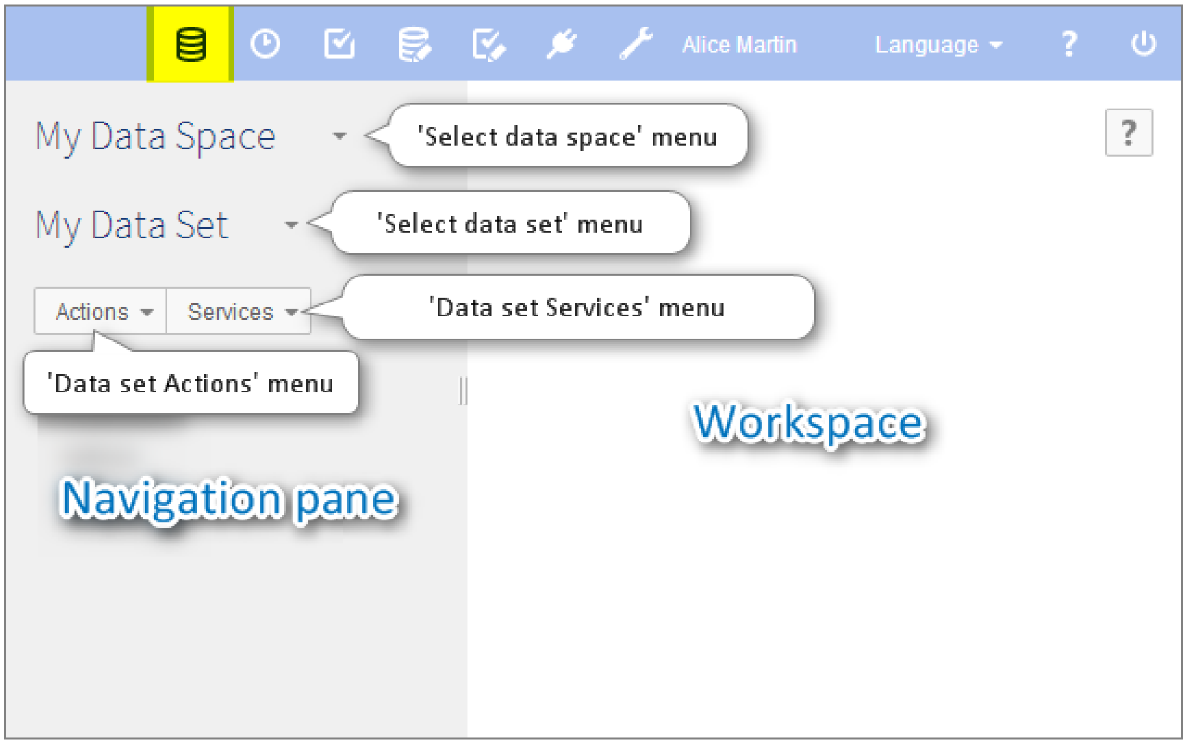
Basic concepts related to data sets

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

- data space [p 23]
- data set [p 22]
- record [p 22]
- field [p 21]
- primary key [p 21]
- foreign key [p 21]
- table (in data set) [p 22]
- group [p 21]

17.2 **Using the Data user interface**

Data sets can be created, accessed and modified in the **Data** area using the Advanced perspective [p 15] or from a specifically configured perspective. Only authorized users can access these interfaces.



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

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

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

See also

Creating a data set *[p 93]*

Searching and filtering data *[p 96]*

Working with records in the user interface *[p 103]*

Inheritance *[p 23]*

Related concepts

Data model *[p 30]*

Data space *[p 72]*

CHAPTER **18**

Creating a data set

This chapter contains the following topics:

18.1 Creating a root data set

To create a new root data set, that is, one that does not inherit from a parent data set, select the '**Select data set** [p 91]' ▼ menu in the navigation pane, click the '**Create a data set**' button in the pop-up, and follow through the wizard.

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

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

- A *packaged data model* is a data model that is located within a module, which is a web application.
- An *embedded data model* is a data model that is managed entirely within the EBX5 repository.
- An *external data model* is one that is stored outside of the repository and is referenced using its URI.

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

Attention

Table contents are not copied when duplicating a data set.

18.2 Creating an inheriting child data set

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

To create a child data set, select the '**Select data set** [p 91]' ▼ menu in the navigation pane, then click the + button next to the desired parent data set.

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

See also *Data set inheritance [p 109]*

CHAPTER **19**

Viewing table data

This chapter contains the following topics:

19.1 **'View' menu**

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

Views are managed in a dedicated sub-menu: 'Manage views' [p 100].

Views can also be grouped. An administrator has to define groups beforehand in 'Views configuration' under the 'Groups of views' table. The end user can then set a view as belonging to a group through the field 'View group' upon creation or modification of the view. See 'View description' [p 98] for more information.

19.2 **Sorting data**

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

Note that the default order is by ascending primary key.

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

To define custom sort criteria, select *View* > *Sort criteria* in the workspace.

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

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

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

19.3 **Searching and filtering data**

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

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

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

Search

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

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

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

See also *Regex pattern*

Text search

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

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

Examples:

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

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

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

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

Validation messages filter

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

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

Custom table searches

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

See also *Specifying UI filters on a table* *[p 167]*

19.4 **Views**

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

Two types of views can be created:

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

View description

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

Documentation	Localized label and description associated with the view.
Owner	Name of the owner of the view. This user can manage and modify it.
Other authorized profiles	Profiles authorized to use the view.
View mode	Simple tabular view or hierarchical view.
View group	Group to which this view belongs (if any).

Simple tabular views

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

Displayed columns	Specifies the columns that will be displayed in the table.
Sorted columns	Specifies the sort order of records in the table. See Sorting data [p 95].
Filter	Defines filters for the records to be displayed in the table. See Criteria editor [p 315].

Hierarchical views

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

Hierarchy dimension

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

Hierarchical view configuration options

This form allows configuring the hierarchical view options.

Display records in a new window	If 'Yes', a new window will be opened with the record. Otherwise, it will be displayed in a new page of the same window.
Prune hierarchy	If 'Yes', hierarchy nodes that have no children and do not belong to the target table will not be displayed.
Display orphans	If 'Yes', hierarchy nodes without a parent will be displayed.
Display root node	If 'No', the root node of the hierarchy will not be displayed in the view.
Root node label	Localized label of the hierarchy root node.

Labels

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

Filter

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

See also *Criteria editor* *[p 315]*

Ordering field

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

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

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

Attention

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

Actions on hierarchy nodes

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

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

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

View publication

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

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

19.5 Views management

Manage recommended views

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

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

Thus, for a given user, the applied rule will be the first that matches the user's profile.

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

Manage views

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

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

19.6 History

The history feature allows tracking changes on master data.

The history feature must have been previously enabled at the data model level. See Advanced properties for tables [p 55] for more information.

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

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

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

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

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

See also History *[p 275]*

CHAPTER 20

Editing data

This chapter contains the following topics:

20.1 Working with records in the user interface

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

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

Creating a record

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

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

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

Updating an existing record

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

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

Duplicating a record

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

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

Deleting records

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

Comparing two records

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

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

20.2 Importing and exporting data

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

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

See also

CSV Services *[p 247]*

XML Services *[p 241]*

20.3 Restore from history

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

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

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

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

> **Note**
>
> This feature has limitations linked to the limitations of the history feature:
>
> - the 'restore from history' feature is not available on tables containing lists that are not supported by history. See Data model limitations [p 280].
> - computed values, encrypted values and fields on which history has been disabled are ignored when restoring a record from history, since these fields are not historized.

See also *History* *[p 275]*

CHAPTER **21**

Working with existing data sets

This chapter contains the following topics:

21.1 Validating a data set

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

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

See Validation [p 322] for detailed information about incremental data validation.

21.2 Duplicating a data set

To duplicate an existing data set, select it from the '**Select data set** [p 91]' ▼ menu in the navigation pane, then select **Actions** > **Duplicate**.

21.3 Deactivating a data set

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

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

21.4 Managing data set permissions

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

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

See also *Profile* *[p 19]*

Profile	Defines the profile to which these permissions apply.
Restriction policy	If 'Yes', indicates that when the permissions defined here are more strict than otherwise defined, these permissions are respected. This is contrary to the default where the most permissive rights defined take precedence. See Restriction policies [p 309].
Data set actions	Specifies the permissions for actions on the data set.
Create a child data set	Indicates whether the profile can create a child data set. Inheritance also must be activated in the data model.
Duplicate the data set	Indicates whether the profile can duplicate the data set.
Delete the data set	Indicates whether the profile can delete the data set.
Activate/deactivate the data set	Indicates whether the profile can modify the *Activated* property in the data set information. See Deactivating a data set [p 105].
Create a view	Indicates whether the profile can create views and hierarchies in the data set.
Tables policy	Specifies the default permissions for all tables. Specific permissions can also be defined for a table by clicking the '+' button.
Create a new record	Indicates whether the profile can create records in the table.
Overwrite inherited record	Indicates whether the profile can override inherited records in the table. This permission is useful when using data set inheritance.
Occult inherited record	Indicates whether the profile can occult inherited records in the table. This permission is useful when using data set inheritance.
Delete a record	Indicates whether the profile can delete records in the table.
Values access policy	Specifies the default access permissions for all the nodes of the data set and allows the definition of permissions for

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

CHAPTER 22

Data set inheritance

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

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

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

See also *Data model configuration* *[p 38]*

This chapter contains the following topics:

22.1 Data set inheritance structure

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

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

22.2 Value inheritance

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

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

The button can be used to override a value.

Record inheritance

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

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

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

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

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

See also *Record lookup mechanism* *[p 298]*

Workflow models

CHAPTER 23

Introduction to workflow models

This chapter contains the following topics:

23.1 **Overview**

What is a workflow model?

Workflows in EBX5 facilitate the collaborative management of data in the repository. A workflow can include human actions on data and automated tasks alike, while supporting notifications on certain events.

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

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

See also

Workflow model (glossary) [p 25]

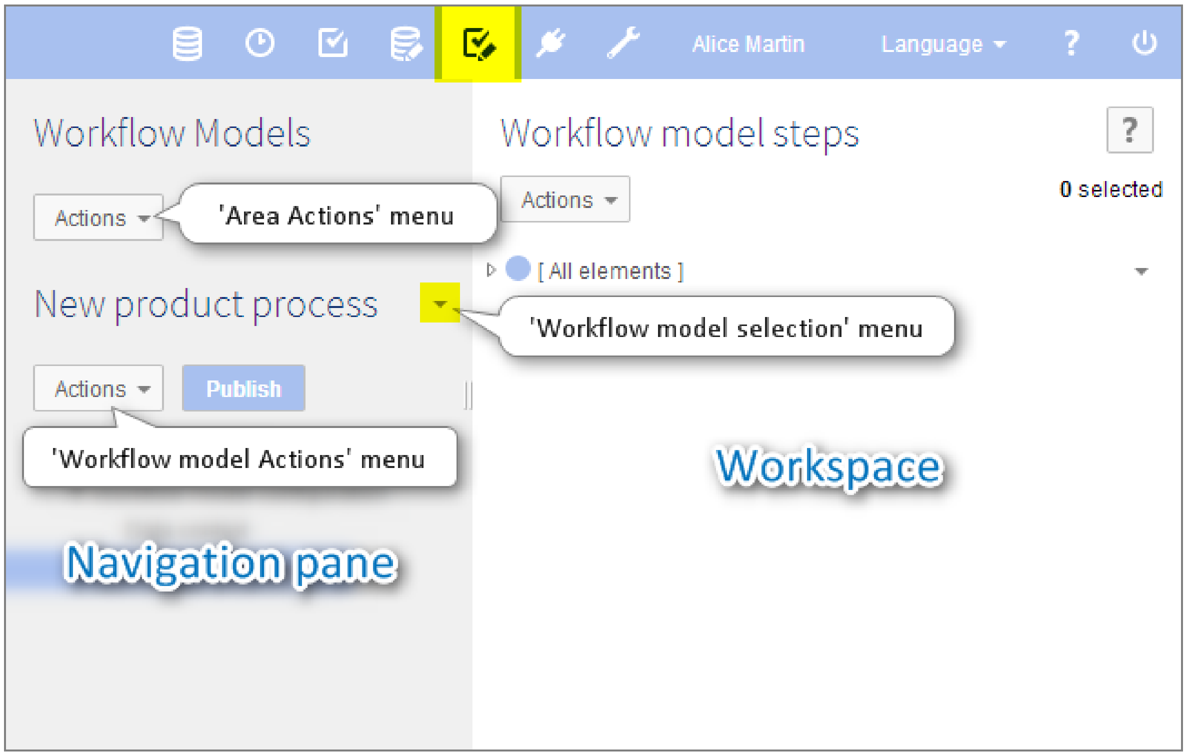
Data workflow (glossary) [p 26]

Basic concepts related to workflow models

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

- script task [p 25]
- user task [p 25]
- work item [p 26]
- workflow condition [p 25]
- sub-workflow invocation [p 26]
- wait task [p 26]
- data context [p 26]

23.2 Using the Workflow Models area user interface



Note

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

23.3 Generic message templates

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

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

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

When creating a new template, two fields are required:

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

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

system.time	System time of the repository.
system.date	System date of the repository.
workflow.lastComment	Last comment on the preceding user task.
workflow.lastDecision	Last decisions made on the preceding user task.
user.fullName	Full name of the notified user.
user.login	Login of the notified user.
workflow.process.label	Label of the current workflow.
workflow.process.description	Description of the current workflow.
workflow.workItem.label	Label of the current work item.
workflow.workItem.description	Description of the current work item.
workflow.workItem.offeredTo	Role to which the current work item has been offered.
workflow.workItem.allocatedTo	User to whom the current work item has been allocated.
workflow.workItem.link	Link to access the current work item in the work item inbox, using the web component API. This link can only be computed if a current work item is defined and if the URL is configured in Workflow-executions, in the email configuration.
workflow.workItem.link.allocateAndStart	Link to access the current work item in the work item inbox, using the web component API. If the target work item has not yet been started, it will be automatically allocated to and started by the user clicking the link. This link can only be computed if a current work item is defined and if the URL is configured in Workflow-executions, in the email configuration.

workflow.currentStep.label	Label of the current step.
workflow.currentStep.description	Description of the current step.

Example

Generic template message:

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

Resulting email:

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

23.4 Limitations of workflows

The following functionality is currently unsupported in EBX5:

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

Related concepts *Data workflows* *[p 140]*

CHAPTER 24

Creating and implementing a workflow model

This chapter contains the following topics:

24.1 Creating a workflow model

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

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

24.2 Implementing the steps

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

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

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

Progress strategy of the next step

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

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

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

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

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

24.3 User tasks

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

Participants

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

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

See also *Extension* *[p 122]*

Service

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

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

See also *EBX5 built-in services* *[p 185]*

Configuration

Main options > Enable reject

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

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

Main options > Enable confirmation request

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

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

Main options > Comments required

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

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

Main options > Customized labels

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

Termination > Task termination criteria

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

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

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

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

Termination > Reject tolerance

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

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

The following task termination criteria automatically tolerate all rejections:

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

Extension

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

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

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

Notification

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

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

See also *Generic message templates* *[p 115]*

Reminder

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

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

Deadline

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

There are two deadline types:

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

24.4 Script tasks

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

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

Library script task	EBX5 includes a number of built-in library script tasks, which can be used as-is. Any additional library script tasks must be declared in a `module.xml` file. A library script task must define its label, description and parameters. When a user selects a library script task for a step in a workflow model, its associated parameters are displayed dynamically.
Specific script task	Specifies a Java class that performs custom actions. The associated class must belong to the same module as the workflow model. Its labels and descriptions are not displayed dynamically to users in workflow models.

Packaging EBX5 modules [p 437]

Library script tasks

EBX5 includes the following built-in library script tasks:

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

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

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

Attention

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

See the **example** `Package com.orchestranetworks.workflow`API in the Java API.

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

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

Specific script tasks

Specific script tasks are classes that extend the class `ScriptTask`API.

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

Attention

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

See the **example** `Package com.orchestranetworks.workflow`API in the Java API.

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

24.5 Conditions

Conditions are decision steps in workflows.

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

Library condition	EBX5 includes a number of built-in library conditions, which can be used as-is. Any additional library script tasks must be declared in a `module.xml` file. A library condition must define its label, description and parameters. When a user selects a library condition for a step in a workflow model, its associated parameters are displayed dynamically.
Specific condition	Specifies a Java class that implements a custom condition. The associated class must belong to the same module as the workflow model. Its labels and descriptions are not displayed dynamically to users in workflow models.

Packaging EBX5 modules [p 437]

Library conditions

EBX5 includes the following built-in library conditions:

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

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

See the **example** `Package com.orchestranetworks.workflow`^API in the Java API.

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

Specific conditions

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

See the **example** `Package com.orchestranetworks.workflow`^API in the Java API.

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

24.6 Sub-workflow invocations

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

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

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

Two types of sub-workflow invocations exist:

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

24.7 Wait tasks

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

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

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

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

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

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

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

CHAPTER 25

Configuring the workflow model

This chapter contains the following topics:

25.1 Information associated with a workflow model

To view and edit the owner and documentation of your workflow model, select 'Information' from the workflow model 'Actions' [p 115] menu for your workflow model in the navigation pane.

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

25.2 Workflow model properties

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

Module name	Module containing specific Java resources (user task extensions, specific scripts and conditions).
Notification of start	The list of profiles to which to send notifications, based on a template, when a data workflow is launched. See Generic message templates [p 115].
Notification of completion	The list of profiles to which to send notifications, based on a template, when a data workflow is completed). The notification is only sent if the workflow has been completed under normal circumstances, that is, not due to an administration action. See Generic message templates [p 115].
Priority	By default, each workflow associated with this model will be launched with this priority. Setting a priority is optional. If no priority is defined here, and a default priority is set for the repository, the repository default priority will be used for any associated workflow with no priority assigned. See Work item priorities [p 147] for more information. **Note:** Only users who are defined as workflow administrators will be able to manually modify the priority level of any associated data workflows.
Activate quick launch	By default, when a workflow is launched, the user is prompted to enter a documentation for the new workflow in an intermediate form. This documentation is optional. Setting the 'Activate quick launch' property to 'Yes' allows skipping this documentation step and proceeding directly to the workflow launch.
Automatically open the first step	Allows determining the navigation after a workflow is launched. By default, once a workflow is launched, the current table (workflow launchers or monitoring > publications) is automatically displayed. Enabling this property will allow the workflow user to keep working on the launched workflow. If, after the first workflow step is executed, a work item is reached, and this work item can be started by the workflow creator, then the work item is automatically opened (if several work items are reached, the first created is opened). This will save the

	user from selecting the corresponding work item from the work items inbox. If no work item has been reached, the next step progress strategy is evaluated. If no work item has been opened, the table from which the workflow has been launched is displayed. **Limitation:**This property will be ignored if the first step is a sub-workflow invocation.
Permissions	Permissions on actions related to the data workflows associated with the workflow model.
Programmatic action permissions	Defines a custom component that handles the permissions of the workflow. If set, this overrides all permissions defined in the property 'Permissions'.

25.3 Permissions on associated data workflows

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

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

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

Note

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

See also *Workflow administration* *[p 151]*

25.4 Workflow model snapshots

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

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

25.5 **Deleting a workflow model**

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

See also *Publishing workflow models [p 137]*

CHAPTER **26**

Publishing workflow models

This chapter contains the following topics:

26.1 About workflow publications

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

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

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

26.2 Publishing and workflow model snapshots

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

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

See also *Workflow model snapshots* *[p 134]*

26.3 Sub-workflows in publications

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

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

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

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

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

Data workflows

CHAPTER **27**

Introduction to data workflows

This chapter contains the following topics:

27.1 Overview

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

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

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

See also

Work items *[p 145]*

Launching and monitoring data workflows *[p 149]*

Administration of data workflows *[p 151]*

Permissions on associated data workflows *[p 132]*

Related concepts *Workflow models* *[p 114]*

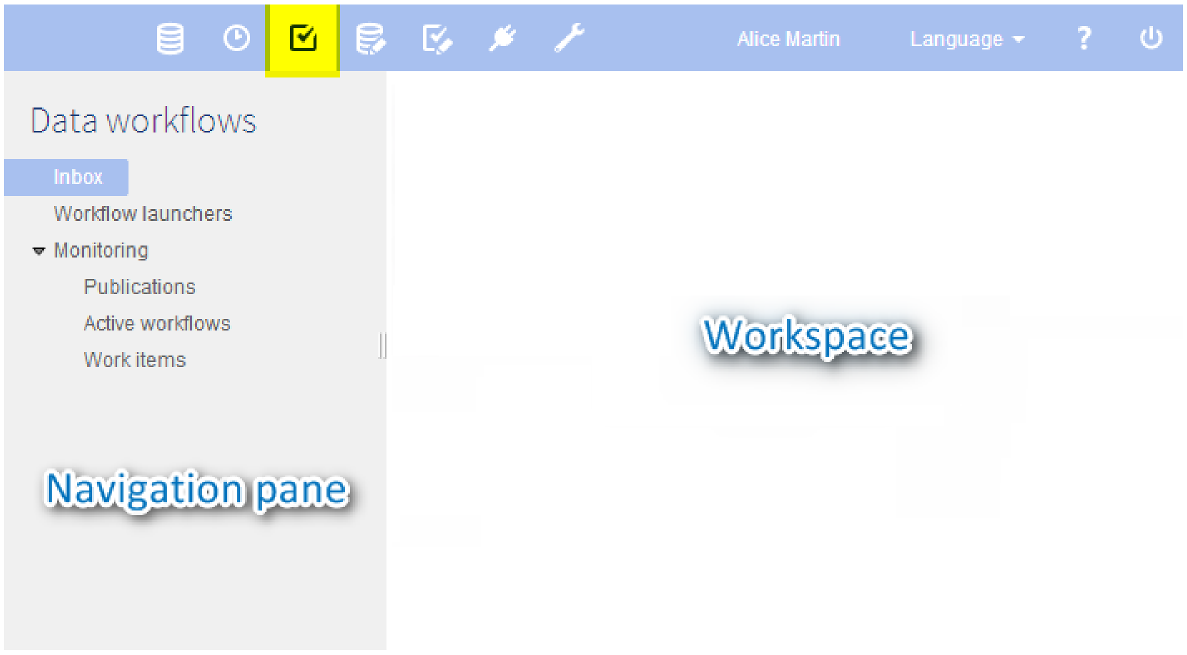
CHAPTER 28

Using the Data Workflows area user interface

This chapter contains the following topics:

28.1 Navigating within the interface

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



Note

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

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

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

Note

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

See also

Using EBX5 as a web component *[p 169]*

`ServiceKey` API

28.2 Navigation rules

Work items inbox

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

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

See the progress strategy of a workflow step [p 120] in workflow modeling.

Workflow launchers

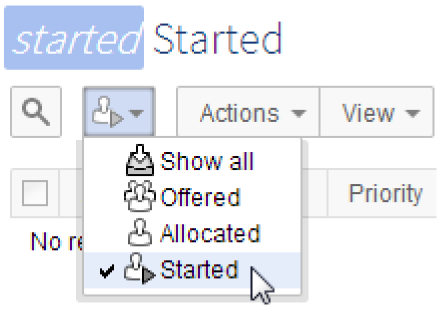
By default, once a workflow has been launched, the workflow launchers table is displayed.

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

See the automatic opening of the first workflow step [p 130] in workflow modeling.

28.3 Filtering items in views

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



28.4 Graphical workflow view

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

CHAPTER 29

Work items

This chapter contains the following topics:

29.1 About work items

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

See also *Integration of UI services with workflows or perspectives* *[p 180]*

Work item states

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

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

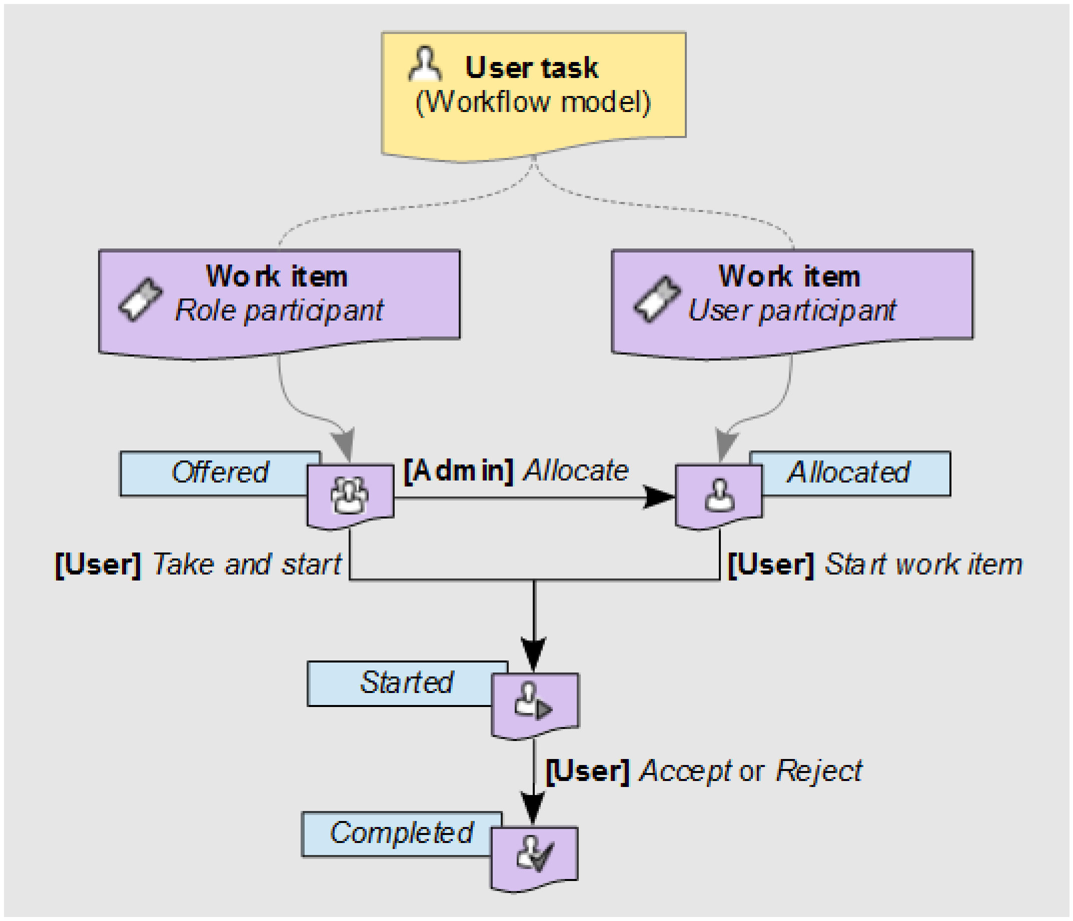
By default, for each role included as a participant of the user task in the workflow model, the data workflow creates one work item in the *offered* state. Any user who is a member of that role can claim the work item using the action 'Take and start', thereby moving the work item to the *started* state.

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

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

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

Diagram of work item states



29.2 Working on work items as a participant

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

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

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

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

29.3 Work item priorities

Work items may carry a priority value, which can be useful for sorting and filtering outstanding work items. The priority of a work item is set at the level of its data workflow, rather than being specific to the individual work item itself. Thus, if a data workflow is considered urgent, all its associated open work items are also considered to be urgent. By default, there are six priority levels ranging from 'Very low' to 'Urgent', however the visual representation and naming of the available priority levels depend on the configuration of your EBX5 repository.

See also *user task (glossary)* *[p 25]*

Related concepts *User tasks* *[p 120]*

CHAPTER 30

Launching and monitoring data workflows

This chapter contains the following topics:

30.1 Launching data workflows

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

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

30.2 Monitoring activities

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

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

30.3 Managing work item allocation

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

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

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

See also

Work items *[p 145]*

Permissions on associated data workflows *[p 132]*

Related concepts *Workflow models* *[p 114]*

CHAPTER 31

Administration of data workflows

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

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

This chapter contains the following topics:

31.1 Overview of data workflow execution

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

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

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

The token can be in the following states:

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

See also *Data workflow administration* *[p 401]*

31.2 Data workflow administration actions

Actions on publications

Disabling a workflow publication

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

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

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

Unpublishing a workflow publication

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

1. Disable the workflow publication to prevent users from continuing to launch new data workflows from it, as described in Disabling a workflow publication [p 152].
2. Unpublish the workflow publication by selecting **Actions** > **Unpublish** from the workflow publication's entry in the same table of publications.

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

Actions on data workflows

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

Replaying a step

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

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

Terminating and cleaning an active data workflow

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

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

Forcing termination of an active data workflow

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

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

Forcing resumption of a waiting data workflow

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

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

Cleaning a completed data workflow

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

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

Modifying the priority of a data workflow

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

See also *Permissions on associated data workflows* *[p 132]*

Data services

CHAPTER 32

Introduction to data services

This chapter contains the following topics:

32.1 Overview

What is a data service?

A data service [p 27] is a standard Web service that can access data or interact with EBX5, thus offering some of the same functionality as is available through the user interface.

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

See also *Reference Manual [p 199]*

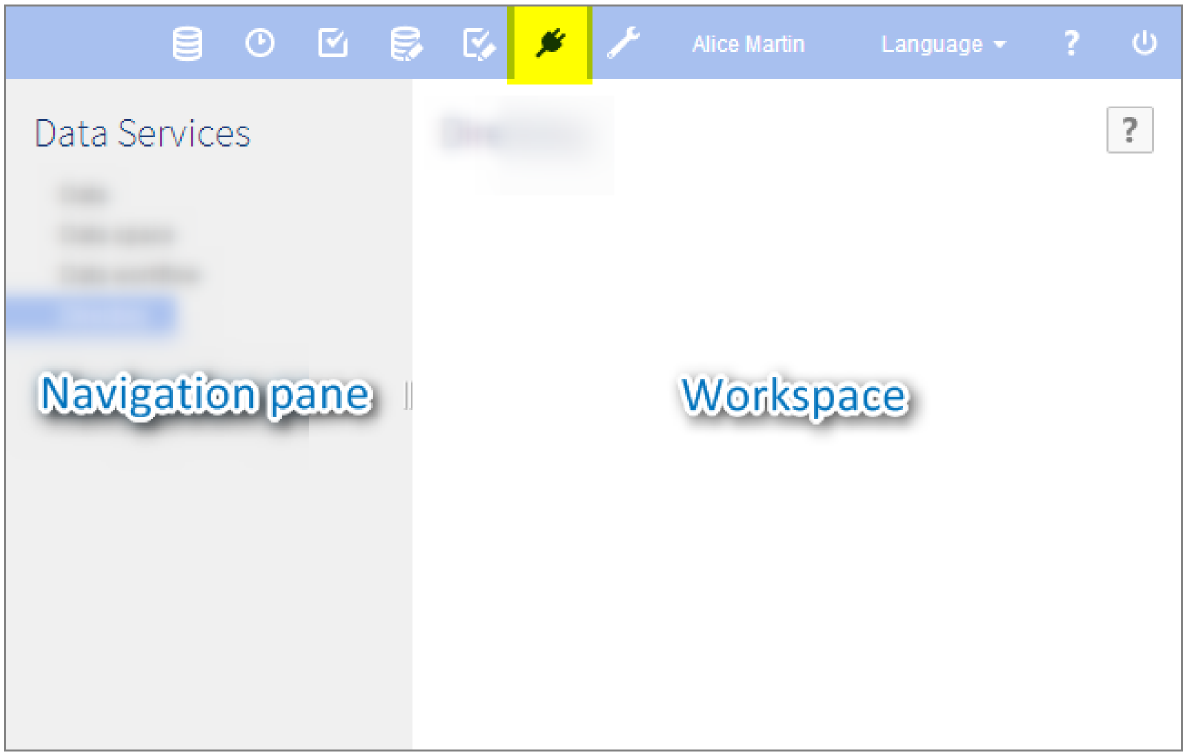
Lineage

Lineage [p 27] is used to establish user permission profiles for non-human users, namely data services. When accessing data using WSDL interfaces, data services use the permission profiles established through lineage.

Glossary

See also *Data services [p 27]*

32.2 Using the Data Services area user interface



Note

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

Related concepts

Data space [p 72]

Data set [p 90]

Data workflows [p 140]

Introduction [p 199]

CHAPTER 33

Generating data service WSDLs

This chapter contains the following topics:

33.1 Generating a WSDL for operations on data

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

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

Operations on data sets

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

- Get data set table(s) changes between data spaces or snapshots
- Replication unit refresh

Operations on tables

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

- Insert record(s)
- Select record(s)
- Update record(s)
- Delete record(s)
- Count record(s)
- Get changes between data space or snapshot
- Get credentials
- Run multiple operations on tables in the data set

See also

WSDL download using a HTTP request *[p 206]*

Operations generated from a data model *[p 213]*

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

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

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

Operations on data spaces

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

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

See also

WSDL download using a HTTP request *[p 206]*

Operations on data sets and data spaces *[p 230]*

33.3 Generating a WSDL for data workflow operations

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

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

Operations on data workflows

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

See also

WSDL download using a HTTP request [p 206]

Operations on data workflows [p 236]

33.4 Generating a WSDL for lineage

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

The operations available for accessing tables are the same as for WSDL for operations on data [p 159].

Steps for generating the WSDL for lineage are as follows:

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

See also *Lineage [p 156]*

33.5 Generating a WSDL for user interface management

This action is only available to administrators.

To generate a WSDL for managing the user interface, select 'User interface' from the **Data Services** area.

Operations for user interface management

- Close user interface
- Open user interface

See also

WSDL download using a HTTP request [p 206]

User interface operations *[p 240]*

33.6 Generating a WSDL to modify the default directory

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

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

Operations on the default directory

The operations available for accessing tables are the same as for WSDL for operations on data [p 159].

See also

WSDL download using a HTTP request *[p 206]*

Directory services *[p 239]*

Reference Manual

Integration

CHAPTER 34

Overview of integration and extension

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

This chapter contains the following topics:

34.1 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 [p 329].

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 [p 165].

See also *Using EBX5 as a web component [p 169]*

34.2 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 [p 175]*

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`*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`API Java class. Next, declare the UI bean on the element in its data model.

See also

*`UIBeanEditor`*API

Properties of data model elements [p 50]

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`*API

Specifying UI filters on a table

In addition to the default filters and search panes in the user interface, it is possible to specify additional filters dependent on the structure of the table. For that a specific class must be defined in the definition of the table and must extend UITableFilter.

See `UITableFilter`API for more information.

34.3 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* *[p 199]*

34.4 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 *[p 241]*

CSV import and export *[p 247]*

34.5 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`API Java class for more information.

CHAPTER **35**

Using EBX5 as a web component

This chapter contains the following topics:

35.1 **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 [p 329]*

35.2 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 [p 171].

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 [p 165] or UI bean [p 166], 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`API in the API.

35.3 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 parameter `firstCallDisplay` may change this automatic display according to its value.

The repository elements that can be selected are as follows:

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

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

See also *Scope [p 173]*

See also *firstCallDisplay [p 173]*

35.4 **Request specifications**

Base URL

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

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

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

User authentication and session information parameters

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

Entity and service selection parameters

Parameter	Description	Required
`branch`	Selects the specified 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 `xpath` or `viewPublication` is specified.
`viewPublication`	Specifies the publication name of the tabular view to apply to the selected content. This publication name is the one declared during the publication of the view. It can be found in the 'Administration' area under **Views configuration > Views**. All settings of the view, that is, its filters, sort order, and displayed columns, are applied to the result. A 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 `xpath` parameter as a logical 'AND' operation.	No
`xpath`	Specifies a node selection in the data set. Value may be a valid absolute path located in the selected data set. The notation must conform to a simplified XPath, with abbreviated syntax. It can also be a predicate surrounded by "[" and "]" if a table can be automatically selected using other web component parameters (for example, `viewPublication` or `workflowView`). For XPath syntax, see XPath supported syntax [p 253] See `UIHttpManagerComponent.setPredicate`API for more information.	No
`service`	Specifies the service to access. For more information on built-in UI services, see Built-in services [p 185]. In the Java API, see `ServiceKey`API for more information.	No
`workflowView`	Specifies the workflow section to be selected. See `WorkflowView`API for more information.	No.
`perspectiveName`	Specifies the name of the perspective to be selected. Known limitation: the parameter `scope` is not supported by the perspectives.	No.
`perspectiveActionId`	Specifies the identifier of the perspective action to be selected. Known limitation: the parameter `scope` is not supported by the perspectives.	No.

Layout parameters

Parameter	Description	Required
`scope`	Specifies the scope to be used by the web component. Value can be `full`, `data`, `dataspace`, `dataset` or `node`. See `UIHttpManagerComponent.Scope`API for more information.	No, default will be computed according to target selection.
`firstCallDisplay`	Specifies which display must be used instead of the one determined by the combination of selection and `scope` parameter. See `FirstCallDisplay`API for more information.	No, default will be computed according to the target selection.
`closeButton`	Specifies how to display the session close button. Value can be `logout` or `cross`. See `UIHttpManagerComponent.CloseButtonSpec`API for more information.	No. If scope is not `full`, no close button will be displayed by default.

`dataSetFeatures`	Specifies which features to display in a UI service at the data set level or a form outside of a table.
	These options pertain only to features in the workspace. It is recommended to use this property with the smallest `scope` possible, namely `dataset` or `node`.
	Syntax:
	`<prefix> ":" <feature> [ "," <feature>]*`
	where
	• `<prefix>` is `hide` or `show`,
	• `<feature>` is `services`, `title`, `breadcrumb`, `save`, or `revert`.
	For example,
	`hide:title,breadcrumb`
	`show:save,revert`
	See `UIHttpManagerComponent.DataSetFeatures`API for more information.
		No.

`viewFeatures`	Specifies which features to display in a tabular or a hierarchy view (at the table level).
	These options pertain only to features in the workspace. It is recommended to use this property with the smallest `scope` possible, namely `dataset` or `node`.
	Syntax:
	`<prefix> ":" <feature> [ "," <feature>]*`
	where
	• `<prefix>` is `hide` or `show`,
	• `<feature>` is `create`, `views`, `selection`, `filters`, `services`, `refresh`, `title`, or `breadcrumb`.
	For example,
	`hide:title,selection`
	`show:service,title,breadcrumb`
	See `UIHttpManagerComponent.ViewFeatures`API for more information.
		No.

`recordFeatures`	Specifies which features must be displayed in a form at the record level.	No.

Parameter	Description	Required

	These options pertain only to features in the workspace. It is recommended to use this property with the smallest `scope` possible, namely `dataset` or `node`.
	Syntax:
	`<prefix> ":" <feature> [ "," <feature>]*`
	where
	• `<prefix>` is `hide` or `show`,
	• `<feature>` is `services`, `title`, `breadcrumb`, `save`, `saveAndClose`, `close`, or `revert`.
	For example,
	`hide:title`
	`show:save,saveAndClose,revert`
	See `UIHttpManagerComponent.RecordFeatures`API for more information.	

`recordsByPage`	Specifies the number of records that will be displayed per page in a table view (either tabular or hierarchical).	No.
`startWorkItem`	Specifies a work item must be automatically taken and started. Value can be `true` or `false`. See `ServiceKey`API for more information.	No. Default value is `false`, where the target work item state remains unchanged.

35.5 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 'instanceId' 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 36

User interface services (UI services)

This chapter contains the following topics:

36.1 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 [p 185] 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 or a perspective, it needs to be declared for this usage. See Integration of UI services with workflows and perspectives [p 180] 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,
- For a perspective, in which case the UI service must be declared in an action menu item,
- From the Java API, using `UIHttpManagerComponent.setService`API.

36.2 **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`	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** `ServiceContext`API ServletContext.getRequestDispatcher(String)	Yes
`activatedForPaths`	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 `osd:table`, 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 `/root/myTable`: • */root/myTable* activates the UI service globally on the table. • */root/myTable{n}* activates the UI service only if exactly 'n' records are selected, where 'n' is a positive integer. • */root/myTable{+}* 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`	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.	No. When set, it is exclusive with `activatedForPaths` attribute.
`class`	Attribute that specifies a Java class that implements `ServicePermission`API.	No. If not defined, service has no permission restriction.
`osd:confirmation`	Element that specifies a confirmation message or that suppresses user confirmation before the UI service is launched. See Confirmation message [p 177] below.	No. If not defined, a default confirmation message is displayed.
`displayInMenu`	Attribute that specifies if the service is displayed in UI menus and buttons.	No. Default value is true.
`orderInMenu`	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.	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: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"`.

36.3 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* *[p 437]*

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`	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 '/'. **See also** *`ServiceContext`*API *ServletContext.getRequestDispatcher(String)*	Yes
`type`	Element that defines the availability of the UI service as a whitespace-delimited list. • 'branch', which makes the UI service available to *data spaces* • 'version', which makes the UI service available to *snapshots* • 'workflow', which makes the UI service available to be defined for user tasks in workflow models • 'perspective', which makes the UI service available to be defined for action menu items in perspective configurations. If the list includes neither 'branch' or 'version', the UI service will not be available in all **Services** menus. If the value 'workflow' is specified, the UI service will be available for workflows on any data space or snapshot. If the value 'perspective' is specified, the UI service will be available for perspectives on any data space or snapshot. See Services on data spaces or snapshots for data workflows or perspectives [p 181]	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 [p 179] 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>
```

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

36.4 **Integration of UI services with workflows or perspectives**

Overview

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

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

Workflow

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.getInteraction`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`API.

Perspective

Once this declaration has been made, the UI service becomes available for action menu items. The perspective administrator then maps the input parameters that are specified in the module configuration file `module.xml`. Output parameters are not used by perspectives but can be defined in `module.xml` if the UI service is also used for worflows.

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. This property is ignored by perspectives.	No
`documentation`	Label and description of the property.	No

Services on data spaces or snapshots for data workflows or perspectives

The declaration of an UI service on data spaces and/or snapshots can indicate if the UI service can also be used with workflows or perspectives.

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.

If the `type` element contains the value `perspective` in the whitespace-delimited list, the UI service will be available for definition of action menu items in perspective configurations.

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

Example

This example declares a UI services that is available in the **Services** menu for data spaces, for user tasks in workflow models and for action menu items in perspective configurations. 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 perspective</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 or perspectives

UI services on data sets are declared at the data model-level. However, in order to make them definable for workflow user tasks or perspective action menu items, 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
`serviceName`	Attribute of `serviceLink` that defines the name of the UI service as it is specified in the data model.	Yes
`importFromSchema`	Element that specifies the path to the data model, relative to the current module (web application).	Yes
`properties`	Element that defines input and output parameters.	Yes. At least the input 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`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 37

Built-in UI services

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

- when defining workflow model tasks [p 120]
- when defining perspective action menu items [p 16]
- as extended UI services [p 182] when used with service extensions [p 182]
- when using EBX5 as a web component [p 169]

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

This chapter contains the following topics:

37.1 Access a data space

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

Service name parameter: `service=@selectDataSpace`

Input parameters

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

37.2 **Access data (default service)**

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

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

Input parameters

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

37.3 **Access the data space merge view**

The merge.view service is automatically considered complete.

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

Input parameters

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

37.4 **Compare contents**

Workflows automatically consider the compare service as complete.

Service name parameter: `service=@compare`

Input parameters

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

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

37.5 Create a new record

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

Service name parameter: `service=@creation`

Input parameters

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

Output parameters

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

37.6 Data workflows

This service provides access to the data workflows user interfaces.

Service name parameter: `service=@workflow`

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

Input parameters

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

37.7 **Duplicate a record**

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

Service name parameter: `service=@duplicate`

Input parameters

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

Output parameters

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

37.8 Export data to a CSV file

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

Service name parameter: `service=@exportToCSV`

Input parameters

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

37.9 **Export data to an XML file**

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

Service name parameter: `service=@exportToXML`

Input parameters

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

37.10 **Import data from a CSV file**

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

Service name parameter: `service=@importFromCSV`

Input parameters

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

37.11 **Import data from an XML file**

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

Service name parameter: `service=@importFromXML`

Input parameters

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

37.12 **Merge a data space**

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

Service name parameter: `service=@merge`

Input parameters

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

Output parameters

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

37.13 **Validate a data space, a snapshot or a data set**

Workflows automatically consider the validation service as complete.

Service name parameter: `service=@validation`

Input parameters

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

Output parameters

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

CHAPTER 38

Introduction

This chapter contains the following topics:

38.1 Overview of data services

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

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

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

Other generic operations for:

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

38.2 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 [p 333] 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 [p 354] and Using JMS [p 200] for more information.

38.3 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`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.XX.YY.ZZ.AA.BB(AA.java:44) at
    com.orchestranetworks.XX.YY.ZZ.CC.DD(CC.java:40) ...
   </description>
  </m:StandardException>
 </detail>
</soapenv:Fault>
```

Using JMS

It is possible to access data services using JMS, instead of HTTP. The JMS architecture relies on one mandatory queue and one optional queue, see configuration JMS [p 354]. 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`.

38.4 **Data services security**

Authentication

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

- Specific authentication based on HTTP `Request` see `Directory.authenticateUserFromHttpRequest`API for more information. An implementation of this method can, for example, extract a password-digest or a ticket from the HTTP request.
- 'Basic Authentication Scheme' method based on the HTTP-Header `Authorization` in base 64, as described in RFC 2324.

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

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

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

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

> **Note**
>
> Only available for Operations [p 213] from HTTP request.

- Standard authentication based on HTTP `Request`. User and password are extracted from request parameters. For more information, see Request parameters [p 207].

  See `Directory.authenticateUserFromLoginPassword`API for more information.

> **Note**
>
> Only available for WSDL generation [p 205].

Overriding the SOAP security header

It is possible to override the default WSS header in order to define another security authentication mechanism. Such an override is taken into account for both HTTP and JMS. To define and override, use the 'SOAP Header Security declaration' configuration settings under Administration > Lineage, which includes the following fields:

Schema location	The URI of the Security XML Schema to import into the WSDL.
Target namespace	The target namespace of elements in the schema.
Namespace prefix	The prefix for the target namespace.
Message name	The message name to use in the WSDL.
Root element name	The root element name of the security header. The name must be the same as the one declared in the schema.
wsdl:part element name	The name of the `wsdl:part` of the message.

The purpose of overriding the default security header is to change the declaration of the WSDL message matching the security header so that it contains the following:

```
<wsdl:definitions ... xmlns:MyPrefix="MyTargetNameSpace" ...
  ...
  <xs:schema ...>
    <xs:import namespace="MyTargetNameSpace" schemaLocation="MySchemaURI"/>
    ...
  </xs:schema>
  ...
  <wsdl:message name="MySecurityMessage">
    <wsdl:part name="MyPartElementName" element="MyPrefix:MySecurityRootElement"/>
  </wsdl:message>
  ...
  <wsdl:operation name="...">
    <soap:operation soapAction="..." style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
      <soap:header message="impl:MySecurityMessage" part="MyPartElementName" use="literal"/>
  ...
  </wsdl:operation>
</wsdl:definitions>
```

The corresponding XML Schema header declaration would be as follows:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="MyNameSpace"
  xmlns:MyPrefix="MyNameSpace">
  <element name="MySecurityRootElement" type="MyPrefix:SpecificSecurity"/>
  <complexType name="SpecificSecurity">
    <sequence>
      <element name="AuthToken" type="string"/>
    </sequence>
  </complexType>
</schema>
```

A SOAP message using the XML schema and configuration above would have the following header:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Header>
    <m:MySecurityRootElement xmlns:m="MyNameSpace">
      <AuthToken>String</AuthToken>
    </m:MySecurityRootElement>
    ...
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    ...
  </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

To handle this non-default header, you must implement the method: `Directory.authenticateUserFromSOAPHeader`API.

> **Note**
>
> Only available for Operations [p 213].

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).

38.5 Monitoring

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

See also

Configuring the EBX5 logs [p 352]

EBX5 main configuration file [p 347]

38.6 Known limitations

Naming convention

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

Date, time & dateTime format

Data services only support the following date and time formats:

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

CHAPTER 39

WSDL generation

This chapter contains the following topics:

39.1 Supported standard

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

39.2 Operation types

A WSDL can be generated for different types of operations:

Operation type	WSDL description
`custom`	WSDL for EBX5 add-ons.
`dataset`	WSDL for data set and replication operations.
`directory`	WSDL for default EBX5 directory operations. It is also possible to filter data using the `tablePaths` [p 208] or `operations` [p 208] parameters.
`repository`	WSDL for data space or snapshot management operations.
`tables`	WSDL for operations on the tables of a specific data set.
`userInterface`	WSDL for user interface management operations (these operations can only be accessed by administrators).
`workflow`	WSDL for EBX5 workflow management operations.

39.3 **Supported access methods**

EBX5 supports the following downloading methods:

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

Global access permissions can be independently defined for each method. For more information see Global permissions [p 378].

A WSDL can only be downloaded by authorized profiles:

Operation type	**Access right permissions**
`custom`	All profiles, if at least one web service is registered.
`dataset`	All profiles.
`directory`	All profiles, if the following conditions are valid: • No specific directory implementation is used. (The built-in Administrator role is only subject to this condition). • Global access permissions are defined for the administration. • 'Directory' data set permissions have writing access for the current profile.
`repository`	All profiles.
`tables`	All profiles.
`userInterface`	Built-in Administrator role or delegated administrator profiles, if all conditions are valid: • Global access permissions are defined for the administration. • 'User interface' data set permissions have writing access for the current profile.
`workflow`	All profiles.

39.4 **WSDL download from the data services user interfaces**

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

> **Note**
>
> See generating data service WSDLs [p 159] in the user guide for more information.

39.5 **WSDL download using a HTTP request**

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

Request details

- HTTP(S) URL format:

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

  Both `<pathInfo>` and `<key - values>` are mandatory. For more information on possible values see: Request parameters [p 207]. An error is returned for incorrect parameters.

- HTTP(S) response status codes:

Status code	Information
200	The WSDL content was successfully generated and is returned by the request (optionally in an attachment [p 209]).
400	Bad request: request argument is incorrect.
401	Unauthorized: request requires an authenticated user.
403	Forbidden: request is not allowed for the authenticated user.
405	Method not allowed: request is not allowed in this configuration.
500	An internal error occurred: request generates an error (a stack trace and a detailed error message are returned).

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

- HTTP(S) parameters size restrictions:

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

Request parameters

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

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

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

Parameter name	As path info	As key - values	Required	Description
`WSDL`	no	yes	yes	Used to indicate the WSDL download. Empty value.
`login`	no	yes	no	A user identifier. Required when the standard authentication method is used. String type value.
`password`	no	yes	no	A password. Required when the standard authentication method is used. String type value.
`type`	yes	no	yes	An operation type [p 205]. Possible values are: `custom`, `dataset`, `directory`, `userInterface`, `repository`, `tables` or `workflow`. String type value.
`branch` `version`	yes	yes	(*)	A data space or a snapshot identifier. (*) required for `tables` and `dataset` types, otherwise ignored. String type value.
`instance`	yes	yes	(*)	A data set identifier. String type value.
`tablePaths`	no	yes	no	A list of table paths. Optional for `tables` or `directory` types, otherwise ignored. If not defined, all tables are selected. Each table path is separated by a comma character. String type value.
`operations`	no	yes	no	Allows generating a WSDL for a subset of operations. Optional for `tables` or `directory` operation types, otherwise ignored. If not defined, all operations for the given type are generated. This parameter's value is a concatenation of one or more of the following characters: • C = Count record(s) • D = Delete record(s) • E = Get credentials • G = Get changes

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

Request examples

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

- The WSDL will contain all repository operations.

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

- The WSDL will contain all workflow operations.

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

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

  Path info

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

  Key - values

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

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

  Path info

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

  Key - values

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

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

  Path info

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

  Key - values

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

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

  Path info

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

Key - values

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

CHAPTER 40

Operations

This chapter contains the following topics:

40.1 Operations generated from a data model

For a data model used in an EBX5 repository, it is possible to dynamically generate a corresponding WSDL defining operations used to access the data sets that are an instance of this data model. For example, for a table located at the path `/root/XX/exampleTable`, the generated requests would follow the structure of its underlying data model and include the name of the table `<m:{operation}_exampleTable xmlns:m="urn:ebx-schemas:dataservices_1.0">`.

Attention

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

Content policy

Access to the content of records, the presence or absence of XML elements, depend on the resolved permissions [p 303] of the authenticated user session. Additional aspects, detailed below, can impact the content.

Disabling fields from data model

The `hiddenInDataServices` property, defined in the data model, allows always hiding fields in data services, regardless of the user profile. This parameter has an impact on the generated WSDL: any hidden field or group will be absent from the request and response structure.

Modifying the `hiddenInDataServices` parameter value has the following impact on a client which would still use the former WSDL:

- On request, if the data model property has been changed to `true`, and if the concerned field is present in the WSDL request, an exception will be thrown.
- On response, if the schema property has been changed to `false`, WSDL validation will return an error if it is activated.

This setting of "Default view" is defined inside data model.

See also

Hiding a field in Data Services *[p 496]*

Permissions *[p 303]*

Association field

Read-access on table records can export the association fields as displayed in UI Manager. This feature can be coupled with the 'hiddenInDataServices' model parameter.

> **Note**
>
> Limitations: change and update operations do not manage association fields. Also, the select operation only exports the first level of association elements (the content of associated objects cannot contain association elements).

Common request parameters

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

Element	Description	Required
branch	The identifier of the data space to which the data set belongs.	Either this parameter or the 'version' parameter must be defined. Required for the 'insert', 'update' and 'delete' operations.
version	The identifier of the snapshot to which the data set belongs.	Either this parameter or the 'branch' parameter must be defined
instance	The unique name of the data set which contains the table to query.	Yes
predicate	XPath predicate [p 253] defines the records on which the request is applied. If empty, all records will be retrieved, except for the 'delete' operation where this field is mandatory.	Only required for the 'delete' operation
data	Contains the records to be modified, represented within the structure of their data model. The whole operation is equivalent to an XML import. The details of the operations performed on data content are specified in the section Import [p 241].	Only required for the insert and update operations
viewPublication	This parameter can be combined with the predicate [p 215] parameter as a logical `AND` operation. The behavior of this parameter is described in the section EBX5 as a web component [p 172]. It cannot be used if the 'viewId' parameter is used, and cannot be used on hierarchical views.	No
viewId	*Deprecated since version 5.2.3*. This parameter has been replaced by the parameter 'viewPublication'. While it remains available for backward compatibility, it will eventually be removed in a future version. This parameter cannot be used if the 'viewPublication' parameter is used.	No
blockingConstraintsDisabled	This property is available for all table updates data service operations. If `true`, the validation process disables blocking constraints defined in the data model. If this parameter is not present, the default is `false`.	No
disableRedirectionToLastBroadcast	This property is available for all data service operations. If `true`, 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.	No

Element	Description	Required
	If this parameter is not present, the default is `false` (redirection on a D3 master enabled), unless the configuration property ebx.dataservices.disableRedirectionToLastBroadcast.default [p 354] has been set. If the specified data space is not a delivery data space on a D3 master node, this parameter is ignored.	

Select operation

Select request

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 215].	
version	See the description under Common parameters [p 215].	
instance	See the description under Common parameters [p 215].	
predicate	See the description under Common parameters [p 215]. This parameter can be combined with the viewPublication [p 215] parameter as a logical `AND` operation.	
viewPublication	See the description under Common parameters [p 215].	
includesTechnicalData	The response will contain technical data if `true`. See also the optimistic locking [p 229] section. Each returned record will contain additional attributes for this technical information, for instance: `...<table ebxd:lastTime="2010-06-28T10:10:31.046" ebxd:lastUser="Uadmin" ebxd:uuid="9E7D0530-828C-11DF-B733-0012D01B6E76">... .`	No
exportCredentials	If `true` the select will also return the credentials for each record.	No
pagination	Enables pagination, see child elements below.	No
pageSize (nested under the pagination element)	When pagination is enabled, defines the number of records to retrieve.	When pagination is enabled, yes
previousPageLastRecordPredicate (nested under the pagination element)	When pagination is enabled, XPath predicate that defines the record after which the page must fetched, this value is provided by the previous response, as the element `lastRecordPredicate`. If the passed record is not found, the first page will be returned.	No
disableRedirectionToLastBroadcast	See the description under Common parameters [p 215].	

Select response

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

See also the optimistic locking [p 229] section.

Delete operation

Delete request

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 215].	
instance	See the description under Common parameters [p 215].	
predicate	See the description under Common parameters [p 215].	
occultIfInherit	Occults the record if it is in inherit mode. Default value is `false`.	No
checkNotChangedSinceLastTime	Timestamp used to ensure that the record has not been modified since the last read. Also see the optimistic locking [p 229] section.	No
blockingConstraintsDisabled	See the description under Common parameters [p 215].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 215].	

Delete response

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

with:

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

Count operation

Count request

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

with:

Element	Description
branch	See the description under Common parameters [p 215].
version	See the description under Common parameters [p 215].
instance	See the description under Common parameters [p 215].
predicate	See the description under Common parameters [p 215].
disableRedirectionToLastBroadcast	See the description under Common parameters [p 215].

Count response

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

with:

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

Update operation

Update request

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

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 215].	
instance	See the description under Common parameters [p 215].	
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. If `false` and node is declared `hiddenInDataServices`, the target value is not changed. The complete behavior is described in the sections Insert and update operations [p 242].	No
blockingConstraintsDisabled	See the description under Common parameters [p 215].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 215].	
data	See the description under Common parameters [p 215].	

See also *Optimistic locking [p 229]*

Update response

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

with:

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

Insert operation

Insert request

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 215].	
instance	See the description under Common parameters [p 215].	
byDelta	If `true` and an element does not currently exist in the incoming message, the target value is not changed. If `false` and node is declared `hiddenInDataServices`, the target value is not changed. The complete behavior is described in the sections Insert and update operations [p 242].	No
blockingConstraintsDisabled	See the description under Common parameters [p 215].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 215].	
data	See the description under Common parameters [p 215].	

Insert response

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

with:

Element	Description
status	'00' indicates that the operation has been executed successfully. '95' indicates that at least one operation has violated a blocking constraint, resulting in the overall operation being aborted.
predicate	A predicate matching the primary key of the inserted record. When several records are inserted, the predicates follow the declaration order of the records in the input message.

Get changes operations

Get changes requests

Changes between two data sets:

```
<m:getChangesOnDataSet_{schemaName} xmlns:m="urn:ebx-schemas:dataservices_1.0">
```

```
 <branch>String</branch>
 <version>String</version>
 <instance>String</instance>
 <compareWithBranch>String</compareWithBranch>
 <compareWithVersion>String</compareWithVersion>
 <compareWithInstance>String</compareWithInstance>
 <resolvedMode>boolean</resolvedMode>
</m:getChangesOnDataSet_{schemaName}>
```

Changes between two tables:

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 215].	
version	See the description under Common parameters [p 215].	
instance	See the description under Common parameters [p 215].	
compareWithBranch	The identifier of the data space with which to compare.	One of either this parameter or the 'compareWithVersion [p 223]' parameter must be defined.
compareWithVersion	The identifier of the snapshot with which to compare.	One of either this parameter or the 'compareWithBranch [p 223]' parameter must be defined.
compareWithInstance	The identifier of the data set with which to compare. If it is undefined, instance [p 223] parameter is used.	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
pagination	Enables pagination context for the operations `getChanges` and `getChangesOnDataSet`. Allows client to define pagination context size. Each page contains a collection of inserted, updated and/or deleted records of tables according to the maximum size. Get changes persisted context is built at first call according to the page size parameter in request. The pagination context is persisted on the server file system [p 375] and allows invoking the next page until last page or when a timeout is reached. For creation: Defines `pageSize` parameter. For next: Defines `context` element with `identifier` from previous response. Enables pagination, see child elements below.	No
pageSize (nested under pagination element)	Defines maximum number of records in each page. Minimal size is `50`.	No (Only for creation)
context (nested under pagination element)	Defines content of pagination context.	No (Only for next)
identifier (nested under context element)	Pagination context identifier.	Yes

Element	Description	Required
disableRedirectionToLastBroadcast	See the description under Common parameters [p 215].	

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`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>
```

with:

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

Get changes operation with pagination enabled

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

For creation:

Extract of request:

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

Extract of response:

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

For next:

Extract of request:

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

Extract of response:

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

For last:

Extract of request:

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

Extract of response:

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

Get credentials operation

Get credentials request

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 215].	
version	See the description under Common parameters [p 215].	
instance	See the description under Common parameters [p 215].	
viewPublication	See the description under Common parameters [p 215].	

Get credentials response

```
<ns1:getCredentials_{TableName}Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <XX>
  <TableName>
   <a>R</a>
   <b>W</b>
   <c>H</c>
   <d>W</d>
   ...
  </TableName>
 </XX>
</ns1:getCredentials_{TableName}Response>
```

With the following possible values:

- R: for read-only
- W: for read-write
- H: for hidden

Multiple chained operations

Multiple operations request

It is possible to run multiple operations across tables in the data set, while ensuring a consistent response. The operations are executed sequentially, according to the order defined on the client side.

All operations are executed in a single transaction with a `SERIALIZABLE` isolation level. If all requests in the multiple operation are read-only, they are allowed to run fully concurrently along with other read-only transactions, even in the same data space.

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

See Concurrency and isolation levels [p 503].

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

```
</m:multi_>
```

with:

Element	Description	Required
branch	See the description under Common parameters [p 215].	
version	See the description under Common parameters [p 215].	
instance	See the description under Common parameters [p 215].	
blockingConstraintsDisabled	See the description under Common parameters [p 215].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 215].	
request	This element contains one operation, like a single operation without branch, version and instance parameters. This element can be repeated multiple times for additional operations. Each request can be identified by an 'id' attribute. In a response, this 'id' attribute is returned for identification purposes. Operations such as `count`, `select`, `getChanges`, `getCredentials`, `insert`, `delete` or `update`.	Yes

Note:

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

Limitation:

- The `multi` operation applies to one model and one data set (parameter `instance`).
- The `select` operation cannot use the pagination parameter.

See also

*Procedure*API

*Repository*API

Multiple operations response

See each response operation for details.

```
<ns1:multi_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <response id="id1">
  <{operation}_{TableName}Response>
  ...
  </{operation}_{TableName}Response>
 </response>
 <response id="id2">
  <{operation}_{TableName}Response>
  ...
  </{operation}_{TableName}Response>
 </response>
</ns1:multi_Response>
```

with:

Element	Description
response	This element contains the response of one operation. It is be repeated multiple times for additional operations. Each response is identified by an 'id' attribute set in the request or automatically generated. The content of the element corresponds to the response of a single operation, such as `count`, `select`, `getChanges`, `getCredentials`, `insert`, `delete` or `update`.

Optimistic locking

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

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

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

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

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

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

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

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

40.2 **Operations on data sets and data spaces**

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

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

Element	*Description*	*Required*
	Multiple `documentation` elements may be used for several languages.	
locale (nested under the `documentation` element)	Locale of the documentation.	Only required when the `documentation` element is used
label (nested under the `documentation` element)	Label for the language.	No
description (nested under the `documentation` element)	Description for the language.	No

Validate a data space

Validate data space request

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

Validate data space response

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

Validate a data set

Validate data set request

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

Validate data set response

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

Create a data space

Create data space request

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

Create data space response

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

with:

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

Create a snapshot

Create snapshot request

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

Create snapshot response

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

with:

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

Locking a data space

Lock data space request

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

with:

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

Lock data space response

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

with:

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

Unlocking a data space

Unlock data space request

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

Unlock data space response

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

with:

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

Merge a data space

Merge data space request

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

with:

Element	*Description*	*Required*
deleteDataOnMerge	This parameter is available for the merge data space operation. Sets whether the specified data space and its associated snapshots will be deleted upon merge. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.dataDeletionOnCloseOrMerge.default` in the EBX5 main configuration file [p 354]. See Deleting data and history [p 372] for more information.	No
deleteHistoryOnMerge	This parameter is available for the merge data space operation. Sets whether the history associated with the specified data space will be deleted upon merge. Default value is `false`. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.historyDeletionOnCloseOrMerge.default` in the EBX5 main configuration file [p 354]. See Deleting data and history [p 372] for more information.	No

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

Merge data space response

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

with:

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

Close a data space or snapshot

Close data space or snapshot request

Close data space request:

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

Close snapshot request:

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

with:

Element	*Description*	*Required*
deleteDataOnClose	This parameter is available for the close data space and close snapshot operations. Sets whether the specified snapshot, or data space and its associated snapshots, will be deleted upon closure. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine this default value by specifying the property `ebx.dataservices.dataDeletionOnCloseOrMerge.default` in the EBX5 main configuration file [p 354]. See Deleting data and history [p 372] for more information.	No
deleteHistoryOnClose	This parameter is available for the close data space operation. Sets whether the history associated with the specified data space will be deleted upon closure. Default value is `false`. When this parameter is not specified in the request, the default value is `false`. It is possible to redefine the default value by specifying the property `ebx.dataservices.historyDeletionOnCloseOrMerge.default` in the EBX5 main configuration file [p 354]. See Deleting data and history [p 372] for more information.	No

Close data space or snapshot response

Close data space response:

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

Close snapshot request:

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

Replication refresh

Replication refresh request

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

with:

Element	*Description*	*Required*
branch	See the description under Common parameters [p 230].	Yes
instance	See the description under Common parameters [p 230].	Yes
unitName	Name of the replication unit. **See also** *Replication refresh information* [p 284]	Yes

Replication refresh response

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

with:

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

40.3 Operations on data workflows

Parameters for operations on data workflows are as follows:

Element	**Description**	**Required**
parameters	Input parameters for the process instance to be created.	No
parameter (nested under the `parameters` element). Multiple `parameter` elements may be used.	An input parameter for the process instance.	No
name (nested under the `parameter` element)	Name of the parameter.	Yes
value (nested under the `parameter` element)	Value of the parameter.	No

Start a workflow

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

Sample request:

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

with:

Element	Description	Required
publishedProcessKey	Key of the published workflow to start.	Yes
documentation	See the description under Common parameters [p 230].	No
parameters	See the description under Common parameters [p 236].	No

Sample response:

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

with:

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

Resume a workflow

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

Sample request:

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

```
</m:workflowProcessInstanceResume>
```

with:

Element	Description	Required
resumeId	Resume identifier of the waiting task.	Yes
parameters	See the description under Common parameters [p 236].	No

Sample response:

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

with:

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

End a workflow

End a workflow process instance from its key representation.

Sample request:

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

with:

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

Sample response:

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

with:

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

40.4 Administrative services

Directory services

The services on directory provide operations on the 'Users' and 'Roles' tables of the default directory. To execute an operation related to these services, the authenticated user must be a member of the built-in role 'Administrator'.

The technical data space and data set must be set to `ebx-directory`. For all SOAP operation syntaxes, see Operations generated from a data model [p 213] for more information.

Create a directory user

This example of a SOAP insert request adds a user to the EBX5 directory.

```
<m:insert_user xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <branch>ebx-directory</branch>
 <instance>ebx-directory</instance>
 <data>
 <directory>
  <user>
   <login>login</login>
   <lastName>lastname</lastName>
   <firstName>firstname</firstName>
   <email>firstname.lastname@email.com</email>
   <password>***</password>
   <passwordMustChange>true</passwordMustChange>
   <builtInRoles>
    <administrator>false</administrator>
    <readOnly>false</readOnly>
   </builtInRoles>
   <specificRoles>R1</specificRoles>
   <specificRoles>R2</specificRoles>
   <comments>a comment</comments>
  </user>
 </directory>
 </data>
</m:insert_user>
```

For the insert SOAP response syntax, see insert response [p 221] for more information.

User interface operations

See Application locking [p 384] for more information.

Parameters for operations on the user interface are as follows:

Element	Description	Required
closedMessage	Message to be displayed to users when the user interface is closed to access.	No

Close user interface request

The close operation removes all user sessions that are not acceptable in this mode.

```
<m:close xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <closedMessage>Access is temporarily forbidden.</closedMessage>
</m:close>
```

Close user interface response

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

Open user interface request

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

Open user interface response

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

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.

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

Imports and exports can also be done programmatically.

Default import and export option values can be set in the Administration area, under **User interface > Graphical interface configuration > Default option values > Import/Export**.

This chapter contains the following topics:

41.1 Imports

Attention

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

Import mode

When importing an XML file, you must specify one of the following import modes, which will dictate how the import procedure handles the source records.

Insert mode	Only record creations are allowed. If a record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled.
Update mode	Only modifications of existing records are allowed. If no record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled.
Update or insert mode	If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created.
Replace (synchronization) mode	If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created. If a record exists in the target table but is not present in the source XML file, that record deleted from the table.

Insert and update operations

The mode '*by delta*' allows ignoring data model elements that are missing from the source XML document. This mode can be enabled through data services or the Java API. The following table summarizes the behavior of insert and update operations when elements are not present in the source document.

See the data services operations update [p 219] and insert [p 220], as well as `ImportSpec.setByDelta`API in the Java API for more information.

State in source XML document	Behavior
Element does not exist in the source document	**If 'by delta' mode is disabled (default):** Target field value is set to one of the following: • If the element defines a default value, the target field value is set to that default value. • If the element is of a type other than a string or list, the target field value is set to `null`. • If the element is an aggregated list, the target field value is set to an empty list. • If the element is a string that distinguishes `null` from an empty string, the target field value is set to `null`. If it is a string that does not distinguish between the two, an empty string. • If the element (simple or complex) is hidden in data services, the target value is not changed. **See also** *Hiding a field in Data Services* *[p 496]* **Note:** The user performing the import must have the permissions necessary to create or change the target field value. Otherwise, the value will remain unchanged. **If 'by delta' mode has been enabled through data services or the Java API:** • For the `update` operation, the field value remains unchanged. • For the `insert` operation, the behavior is the same as when `byDelta` mode is disabled.
Element exists but is empty (for example, `<fieldA/>`)	• For nodes of type `xs:string` (or one of its sub-types), the target field's value is set to `null` if it distinguishes `null` from an empty string. Otherwise, the value is set to empty string. • For non-`xs:string` type nodes, an exception is thrown in conformance with XML Schema. **See also** *EBX5 whitespace management for data types* *[p 481]*
Element is present and `null` (for example, `<fieldA xsi:nil="true"/>`)	The target field is always set to `null` except for lists, for which it is not supported. In order to use the `xsi:nil="true"` attribute, you must import the namespace `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`.

It may happen that the XML document contains elements that do not exist in the target data model. By default, in this case, the import procedure will fail. It is possible, however, to allow users to launch import procedures that will ignore the extra columns defined in the XML files. This can be done in the configuration parameters of the import wizard for XML. The default value of this parameter can be configured in the 'User interface' configuration under the 'Administration' area.

Optimistic locking

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

41.2 Exports

> **Note**
>
> Exported XML documents are always encoded in UTF-8.

When exporting to XML, if the table has filters applied, only the records that correspond to the filter are included in the exported file.

The XML export options are as follows:

Download file name	Specifies the name of the XML file to be exported. This field is pre-populated with the name of the table from which the records are being exported.
User-friendly mode	Specifies whether exported values will be represented in a user-friendly way, or in the standard XML raw format. For example, in user-friendly mode, dates and numbers are formatted according to the user's locale, and foreign keys and enumerated values display their associated labels. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Include technical data	Specifies whether internal technical data will be included in the export. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Include computed values	Specifies whether computed values will be exported.
Is indented	Specifies whether the file should be indented to improve readability.

41.3 **Handling of field values**

Date, time & dateTime format

The following date and time formats are supported:

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

41.4 **Known limitations**

Association fields

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

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

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

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

Imports and exports can also be done programmatically.

Default import and export option values can be set in the Administration area, under **User interface > Graphical interface configuration > Default option values > Import/Export**.

See also *Default option values* *[p 386]*

This chapter contains the following topics:

42.1 Exports

When exporting to CSV, if the table has filters applied, only the records that correspond to the filter are included in the exported file.

The CSV export options are as follows:

Download file name	Specifies the name of the CSV file to be exported. This field is pre-populated with the name of the table from which the records are being exported.
File encoding	Specifies the character encoding to use for the exported file. The default is UTF-8.
Enable inheritance	Specifies if inheritance will be taken into account during a CSV export. If inheritance is enabled, resolved values of fields are exported with the technical data that define the possible inheritance mode of the record or the field. If inheritance is disabled, resolved values of fields are exported and occulted records are ignored. By default, this option is disabled. **Note:**Inheritance is always ignored, if the table data set has no parent or if the table has no inherited field.
User-friendly mode	Specifies whether exported values will be represented in a user-friendly way, or in a raw format. For example, in user-friendly mode, dates and numbers are formatted according to the user's locale, and foreign keys and enumerated values display their associated labels. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Include technical data	Specifies whether internal technical data will be included in the export. **Note:** If this option is selected, the exported file will not be able to be re-imported.
Include computed values	Specifies whether computed values will be exported.
Column header	Specifies the whether or not to include column headers in the CSV file. • **No header** • **Label:** For each column in the spreadsheet, the CSV displays its label. Each label is localized according to the locale preference of the current session. If no user-friendly label is defined for a node, the technical name of the node is used. • **XPath:** For each column in the spreadsheet, the CSV displays the path to the node in the table.

Field separator	Specifies the field separator to use for exports. The default separator is comma.
List separator	Specifies the separator to use for values lists. The default separator is line return.

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

42.2 Imports

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

Insert mode	Only record creation is allowed. If a record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled.
Update mode	Only modifications of existing records are allowed. If no record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled.
Update or insert mode	If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created.
Replace (synchronization) mode	If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created. If a record exists in the target table but is not present in the source XML file, that record is deleted from the table.

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

Enable inheritance	Specifies whether the inheritance will be taken into account during a CSV import. If technical data in the CSV file define an inherit mode, corresponding fields or records are forced to be inherited. If technical data define an occult mode, corresponding records are forced to be occulted. Otherwise, fields are overwritten with values read from the CSV file. By default, this option is disabled. **Note:** Inheritance is always ignored if the data set of the table has no parent or if the table has no inherited field.

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

42.3 Handling of field values

Aggregated lists

The CSV import and export services support multi-valued fields, namely aggregated lists. This is only supported for simple typed lists, such as lists of `string`, `date`, or `int`, and for table references. If a table reference is linked to a composite primary key, each item in the list is a formatted string, for example, "true|99". Aggregated lists of groups are not exported.

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

Hidden fields

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

'Null' value for strings

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

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

Date, time & dateTime format

The following date and time formats are supported:

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

42.4 **Known limitations**

Aggregated lists of groups

The CSV import and export services do not support importing multi-valued groups, that is, aggregated lists of complex type elements. Exporting such nodes will not cause any error, however, no value will be exported.

Terminal groups

In a CSV file, it is impossible to differentiate a created terminal group that contains only empty fields from a non-created one.

As a consequence, some differences may appear during comparison after performing an export followed by an import. To ensure the symmetry of import and export, use XML import and export instead. See XML import and export [p 241].

Column label headers

If two columns share the same label header, an export of the table can be performed successfully, but exported data cannot later be re-imported.

Association fields

The CSV import and export services do not support importing association values, that is multi-valued of complex type elements.

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

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

CHAPTER 43
Supported XPath syntax

This chapter contains the following topics:

43.1 Overview

The XPath notation used in EBX5 must conform to the *abbreviated syntax* of the XML Path Language (XPath) Version 1.0 standard, with certain restrictions. This document details the abbreviated syntax that is supported.

43.2 Example expressions

The general XPath expression is:

```
path[predicate]
```

Absolute path

```
/library/books/
```

Relative paths

```
./Author
../Title
```

Root and descendant paths

```
//books
```

Table paths with predicates

```
../../books/[author_id = 0101 and (publisher = 'harmattan')]
/library/books/[not(publisher = 'dumesnil')]
```

Complex predicates

```
starts-with(col3,'xxx') and ends-with(col3,'yyy') and osd:is-not-null(./
col3))
```

```
contains(col3 ,'xxx') and ( not(col1=100) and date-greater-
than(col2,'2007-12-30') )
```

Predicates on label

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

43.3 **Syntax specifications for XPath expressions**

Overview

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

Predicate specification

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

Expression	Format	Notes/Example
<time criterion>	Quoted and formatted as 'HH:mm:ss' or 'HH:mm:ss.SSS'	`'11:55:00'`
<dateTime criterion>	Quoted and formatted as 'yyyy-MM-ddTHH:mm:ss' or 'yyyy-MM-ddTHH:mm:ss.SSS'	`'2007-12-31T11:55:00'`

XPath 1.0 formula

It is possible to use an XPath 1.0 formula in the criterion value part of an atomic predicate expression (right-hand side).

For example, instead of `[./a=3]`, you may use the expression `[./a=(floor(./d)+ 2.0)]`.

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

Predicate on label

The `osd:label()` function can be applied to the path portion of the atomic predicate, in order to resolve the predicate on the label instead of the value. In this case, only string operators and string criteria can be used, i.e. `ends-with(osd:label(./price),'99')`.

A predicate on label is localized, so the criterion must be expressed in the same locale as the predicate-filtered request. For example: `request.setLocale(Locale.FRENCH); request.setXPathFilter("osd:label(./delivery_date)='30/12/2014'");`

Note

It is forbidden to use the `osd:label` function if the right part of the predicate is a contextual value.

Note

If the `osd:label` function is used in a data model, for example on a selection or in the filter predicate of a table reference node, the default locale of the data model (as defined in its module declaration) must be used for the criterion format (even though this is generally not recommended).

See also *`SchemaNode.displayOccurrence`*API

Contextual values

For predicates that are relative to a selected node, the criterion value (that is, the right-hand side of the predicate) can be replaced with a contextual path using the syntax `${<relative-path>}` where *<relative-path>* is the location of the element relative to the selected node.

Note

When calling a method, the criterion is the second parameter, and the first parameter cannot be a relative value.

Aggregated lists

For predicates on aggregated lists, the predicate returns `true` regardless of the comparator if one of the list elements verifies the predicate.

> **Note**
>
> Special attention must be paid to the comparator `!=`. For example, for an aggregated list, `./list != 'a'` is not the same as `not(./list = 'a')`. Where the list contains the elements `(e1,e2,..)`, the first predicate is equivalent to `e1 != 'a' or e2 != 'a' ...`, while the second is equivalent to `e1 != 'a' and e2 != 'a' ...`.

'Null' values

Null values must be explicitly treated in a predicate using the operators `osd:is-null` and `osd:is-not-null`.

For example, `/root/products[./price<100]` or `/root/products[./price!=100]` will not return any products whose prices are not set (`null`). For the latter case to return unset values as well, the predicate must instead be: `/root/products[./price!=100 or osd:is-null(./price)]`.

How to manage single and double quotes in literal expressions

By default, a literal expression is delimited by single quotes (`'`). If the literal expression contains single quotes and no double quotes, the expression must be delimited by double quotes (`"`). If the literal expression contains both single and double quotes, the single quotes must be doubled.

The method `XPathExpressionHelper.encodeLiteralStringWithDelimiters`API in the Java API handles this.

Examples of using `encodeLiteralStringWithDelimiters`

Value of Literal Expression	**Result of this method**
`Coeur`	`'Coeur'`
`Coeur d'Alene`	`"Coeur d'Alene"`
`He said: "They live in Coeur d'Alene".`	`'He said: "They live in Coeur d''Alene".'`

Extraction of foreign keys

In EBX5, the foreign keys are grouped into a single field with the `osd:tableRef` [p 455] 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 **Syntax of the internal String representation of primary keys** `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).

Localization

CHAPTER **44**

Labeling and localization

This chapter contains the following topics:

44.1 **Overview**

EBX5 offers the ability to handle the labeling and the internationalization of data models.

Localizing user interactions

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

1. Session: Each user can select a default locale on the user profile page.
2. Data model: If a data model has been localized into other languages than those natively supported by EBX5, the user can select one of those languages for that particular data model. See Extending EBX5 internationalization [p 263] for more information.

The languages supported by a particular data model must be specified in its module declaration [p 437].

Textual information

In EBX5, most master data entities can have a label and 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 independent of the unique name, so that it remains localizable and modifiable;
- Any node in the data model can have a static label and description;
- Values can have a static label when they are enumerated;
- Validation messages can be customized, and permission restrictions can provide text explaining the reason;
- Each record is dynamically displayed according to its content, as well as the context in which it is being displayed (in a hierarchy, as a foreign key, etc.);

All this textual information can be localized into the locales that are declared by the module.

See also

Labels and messages *[p 487]*

Tables declaration *[p 451]*

Foreign keys declaration *[p 455]*

44.2 Value formatting policies

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

A formatting policy is used to define how to display the values of simple types [p 442].

For each locale declared by the module, its formatting policy is configured in a file located at `/WEB-INF/ebx/{locale}/frontEndFormattingPolicy.xml`. For instance, to define the formatting policy for Greek (`el`), the engine looks for the following path in the module:

```
/WEB-INF/ebx/el/frontEndFormattingPolicy.xml
```

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

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

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

The elements `date`, `dateTime` and `time` are mandatory.

44.3 Syntax for locales

There are two ways to express a locale:

1. The XML recommendation follows the IETF BCP 47 recommendation, which uses a hyphen '-' as the separator.
2. The Java specification uses an underscore '_' instead of a hyphen.

In any XML file (XSD, formatting policy file, etc.) read by EBX5, either syntax is allowed.

For a web path, that is, a path within the web application, only the Java syntax is allowed. Thus, formatting policy files must be located in directories whose locale names respect the Java syntax.

See also *Extending EBX5 internationalization* *[p 263]*

CHAPTER **45**

Extending EBX5 internationalization

This chapter contains the following topics:

45.1 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 [p 260],
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.

45.2 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 EBX5 main configuration file. For example:

  ebx.locales.available=en-US, fr-FR, xx

  - The first locale is always considered the default.
  - The built-in locales, en-US and fr-FR, may be removed if required.

  See EBX5 main configuration file [p 347].
- Deploy the following files in the EBX5 class-path:
  - A formatting policy file, named `com.orchestranetworks.i18n.frontEndFormattingPolicy_xx.xml`,
  - A set of localized message files in a resource bundle.

Specific localization deployment

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

45.3 **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,

Persistence

CHAPTER 46

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* [p 267] refers to any tables that are stored as-is, and thus whose contents can be accessed directly in the database.

This chapter contains the following topics:

46.1 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.

For a comparison between relational mode and semantic mode, see the chapter Overview of modes [p 269]

46.2 Historization

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

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

See also *History [p 275]*

46.3 **Replication**

Replication enables direct SQL access to tables of master data, by making a copy of data in the repository to relational table replicas in the database. Replication can be enabled 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* *[p 283]*

46.4 **Mapped mode**

Overview of mapped mode

Mapped mode refers to cases where tables are persisted in the underlying relational database in a format that allows their data to be accessed directly, outside of 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. 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 *[p 374]*

Data model evolutions *[p 289]*

Data model restrictions due to mapped mode

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

- Limitations of supported databases [p 331]
- Unlimited-length strings: All string fields, except foreign keys, of type `xs:string`, its derived types, and `xs:anyURI` must define a 'maxLength' or 'length' facet. Since a foreign key field is composed of the final primary key field(s) of its target table(s), this facet requirement applies to each of those final primary key fields instead of the foreign key field itself. Additionally, limitations of the underlying database concerning the maximum length of its character types apply, such as *VARCHAR* and *NVARCHAR2*.
- Large lists of columns might not be indexable. Example for Oracle: the database enforces a limit on the maximum cumulated size of the columns included in an index. For strings, this size also depends on the character set. If the database server fails to create the index, you should consider redesigning your indexes, typically by using a shorter length for the concerned columns, or by including fewer columns in the index. The reasoning is that an index leading to this situation would have headers so large that it could not be efficient anyway.
- The attribute `type="osd:password"` is ignored.
- Terminal complex types are supported, however at record-level, they cannot be globally set to `null`.

More generally, tables in mapped mode are subject to any limitations of the underlying RDBMS. For example, the maximum number of columns in a table applies (1000 for Oracle 11g R2, 1600 for PostgreSQL). Note that a history table contains twice as many fields as declared in the schema (one functional field, plus one generated field for the operation code).

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

See also *Data model evolutions [p 289]*

CHAPTER 47

Relational mode

This chapter contains the following topics:

47.1 Overview of modes

Semantic mode explained

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 [p 270] is explicitly specified [p 270].

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 [p 72]

data set inheritance [p 297]

inherited fields [p 299]

Relational mode explained

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

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

Generally, when a 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.

Direct comparison of semantic and relational modes

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 [p 273].
Direct SQL reads	No	Yes, see SQL reads [p 273].
Direct SQL writes	No	Yes, but only under precise conditions, see SQL writes [p 273].
Data validation	Yes, enables tolerant mode.	Yes, some constraints become blocking, see Validation [p 271].
Transactions	See Concurrency and isolation levels [p 503].	See Concurrency and isolation levels [p 503].

47.2 Enabling relational mode for a data model

The data model declares that it is in relational mode. Due to the necessary restrictions of relational mode, such as not having 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**
`dataSpace`	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
`dataSet`	Specifies the data set where the data model must be published.	Yes
`tablesPrefix`	Specifies the common prefix used for naming the generated tables in the database.	Yes

47.3 **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 [p 275]. 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 [p 272]). When a transaction does not comply with a blocking constraint, it is cancelled and a `ConstraintViolationException`API is thrown.

See also *Blocking and non-blocking constraints [p 478]*

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`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.

Important:

- Blocking aspect of the foreign key constraint does not concern filters that may be defined. That is, a foreign key constraint is non-blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected and thus an error will be added to the validation report.
- Foreign key constraints are not in blocking mode upon archive import. Indeed, all blocking constraints, excepted structural constraints, are always disabled when importing archives. This allows flexibility upon archive import where under certain circumstances the import of foreign keys referencing records that are not yet imported must be tolerant.

Constraints on the whole table

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

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

See also *`ConstraintOnTable`*API

47.4 SQL access to data in relational mode

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

See also *SQL access to history [p 278]*

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 could cause 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 [p 272] 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.

47.5 Limitations of relational mode

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

See Supported databases [p 331] for the databases on which relational mode is supported.

Data model restrictions for tables in relational mode

Some restrictions apply to data models in relational mode:

- Data model restrictions due to mapped mode [p 268]
- Aggregated lists [p 447] are not supported in relational tables. Such a schema will cause a compilation error.
- User-defined attributes on relational tables result in data model compilation errors.
- Data set inheritance [p 297].
- Inherited fields [p 299].
- Programmatic constraints, since the computation cost of validation would be too high. However, **constraints on tables** `ConstraintOnTable`API remain available.

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

See also *Data model evolutions* *[p 289]*

Other limitations of relational mode

- Limitations of mapped mode [p 268]
- 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.
- It is not possible to set the `AdaptationValue.INHERIT_VALUE` to a node belonging to a data model in relational mode.

CHAPTER 48 History

This chapter contains the following topics:

48.1 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 [p 280] for more details). XML audit trail [p 395] 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 [p 24]

Relational mode [p 269]

Replication [p 283]

Data model evolutions [p 289]

48.2 Configuring history

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

Configuring history in the repository

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

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

- An internationalized label.
- A list of data spaces (branches) for which history is activated. It is possible to specify whether direct children and/or all descendants should also be concerned.

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

Profile Id	**Description**
`ebx-referenceBranch`	This profile is activated only on the reference data space.
`ebx-allBranches`	This profile is activated on all data spaces.
`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.

Configuring history in the data model

Activating table history

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

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

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

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

Historization must be activated for each table separately. See model design [p 434] documentation for more details.

Disabling history on a specific field or group

For a historized table, the default behavior is to historize all its supported elements (see Impacts and limitations of historized mode [p 280]).

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.

48.3 History views and permissions

Table history view

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

The next section explains how permissions are resolved.

For more information, see table history view [p 25] section. To access the table history view from Java, the method `AdaptationTable.getHistory`API must be invoked.

Permissions for table history

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

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

When defining a programmatic rule, it may be required to distinguish between the functional data set context and the history view context, either because the expected permissions are not the same, or because some fields are not present in the history structure. This is the case for data set fields, computed values and fields for which history has been disabled [p 276]. The methods `Adaptation.isHistory`API and `AdaptationTable.getHistory`API can then be used in the programmatic rule in order to implement specific behavior for history.

Transaction history views

The transaction history view gives access to the executed transactions, independently of a table, a 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 [p 25].

48.4 SQL access to history

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

See also *SQL access to data in relational mode [p 272]*

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 [p 367].

Relational schema overview

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

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

Common and generic tables	The main table is `HV_TX`; each record of this table represents a transaction. Only transactions that involve at least one historized table are recorded. These common tables are all prefixed by "HV".
Specific generated tables	For each historized table, a specific history table is generated. This table contains the history of the data modifications on the table. In the 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".

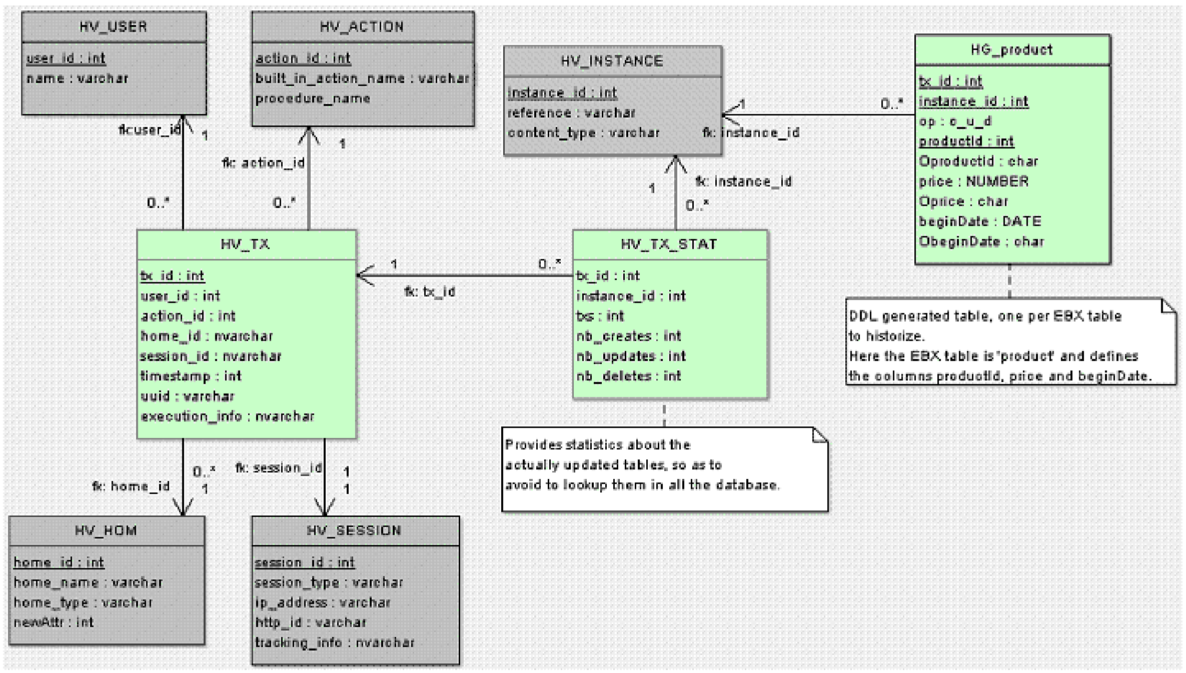
Example of a generated history table

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 [p 296] 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.

48.5 Impacts and limitations of historized mode

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

See Supported databases [p 331] 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 [p 271].

Data model restrictions for historized tables

Some restrictions apply to data models containing historized tables:

- Data model restrictions due to mapped mode [p 268]
- Limitations exist for two types of aggregated lists: aggregated lists under another aggregated list, and aggregated lists under a terminal group. Data models that contain such aggregated lists can be used, however these list will be ignored (not historized).
- Computed values are ignored.
- User-defined attributes on historized tables result in data model compilation errors.

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

See also *Data model evolutions [p 289]*

Other limitations of historized mode

- 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 non-historized field are not supported for historized tables.

  As a consequence, default labels referencing a computed field are not supported for historized tables.

  The workaround is to implement the `UILabelRenderer` interface and adapt the label computation for history.
- D3: the history can be enabled in the delivery data space of a master node, but in the delivery data space of the slave nodes, the historization features are always disabled.
- Recorded user in history: for some specific operations, the user who performs the last operation and the one recorded in the corresponding history record may be different.

  This is due to the fact that these operations are actually a report of the data status at a previous state:
  - Archive import: when importing an archive on a data space, the time and user of the last operation performed in the child data space are preserved, while the user recorded in history is the user who performs the import.
  - Programmatic merge: when performing a programmatic merge on a data space, the time and user of the last operation performed in the child data space are preserved, while the user recorded in history is the user who performs the merge.
  - D3: for distributed data delivery feature, when a broadcast is performed, the data from the master node is reported on the slave node and the time and user of the last operation performed in the child data space are preserved, while the user recorded in history is 'ebx-systemUser' who performs the report on the slave node upon the broadcast.

CHAPTER 49 Replication

This chapter contains the following topics:

49.1 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 'onCommit' refresh mode: updating data in the EBX5 repository triggers the associated inserts, updates, and deletions on the replica database tables.

See also

Relational mode *[p 269]*

History *[p 275]*

Data model evolutions *[p 289]*

Repository administration *[p 366]*

49.2 **Configuring replication**

Enabling replication

To define a *replication unit* on a data model, use the element `osd:replication` under the elements `annotation/appinfo`. Each replication unit specifies tables in a single 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.	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: • `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 [p 286].	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 [p 287]
- 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 [p 287]).

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.

49.3 Accessing a replica table using SQL

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

See also *SQL access to history [p 278]*

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 [p 367]*

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.

49.4 Requesting an 'onDemand' replication refresh

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

There are several ways to request a replication refresh:

- **User interface**: In the 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 [p 236] for data services for more information.
- **Java API**: Call the `ReplicationUnit.performRefresh`API methods in the `ReplicationUnit` API to launch a refresh of the replication unit.

49.5 Impact and limitations of replication

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

See Supported databases [p 331] 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 [p 271].

Data model restrictions for replicated tables

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

- Data model restrictions due to mapped mode [p 268]
- 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 [p 297] for more information.
- Field inheritance is also only supported for the 'onDemand' refresh policy. This means that, at data model compilation, an error is reported if the refresh mode is 'onCommit' and the table to be replicated has an inherited field. See inherited fields [p 299] for more information.
- Computed values are ignored.
- Limitations exist for two types of aggregated lists: aggregated lists under another aggregated list, and aggregated lists under a terminal group. Data models that contain such aggregated lists can be used, however these list will be ignored (not replicated).
- User-defined attributes are not supported. A compilation error is raised if they are included in a replication unit.
- 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 [p 289]*

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.

Distributed data delivery (D3)

Replication is available on both D3 master and slave delivery data spaces. On master, the replication behavior is the same as in a standard semantic data space, but on slaves, the replicated content is that of the last broadcast snapshot.

In a slave delivery data space, some restrictions occur:

- The refresh policy defined in the data model has no influence on the behavior described above: replication always happens on snapshot.
- The action item `Refresh replicas` is not available.
- It is not allowed to invoke the `ReplicationUnit.performRefresh`API method.

See also *D3 overview [p 412]*

Other limitations of replication

- Limitations of supported databases [p 331]
- 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 **50**

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* *[p 267]*

This chapter contains the following topics:

50.1 Types of permitted evolutions

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

Model-level evolutions

The following modifications can be made to existing data models:

- A data model in semantic mode can be declared to be in relational mode. Data should be manually migrated, by exporting then re-importing an XML or archive file.
- Relational mode can be disabled on the data model. Data should be manually migrated, by exporting then re-importing an XML or archive file.
- Replication units can be added to the data model. If their refresh policy is 'onCommit', the corresponding replica tables will be created and refreshed on next schema compilation.
- Replication units can be removed from the data model. The corresponding replica tables will be dropped immediately.
- The data model can be deleted. If it declares replication units, the corresponding replica tables will be dropped immediately. If it is relational or contains historized tables, this change marks the associated mapped tables as disabled. See Purging database resources of mapped tables [p 374] for the actual removal of associated database objects.

Table-level evolutions

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

- A new table can be added. Upon creation, the table can also declare one or more mapped modes.
- An existing table can be deleted. If it declares replication units, the corresponding replica tables will be dropped immediately. If it historized or relational, this change marks the mapped table as disabled. See Purging database resources of mapped tables [p 374] for the actual removal of associated database objects.
- An existing table in semantic mode can be declared to be in relational mode. Data should be manually migrated, by exporting then re-importing an XML or archive file.
- History can be enabled or disabled on a table. History will not take into account the operations performed while it is disabled.
- A table can be renamed. Data should be manually migrated, by exporting then re-importing an XML or archive file, because this change 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 semantic mode, the data of the deleted field will be removed from each record upon its next update. For a replica table, the corresponding column is automatically removed. In history or relational mode, the field is marked as disabled.
- A field can be specifically disabled from the history or replication which applies to its containing table, by using the attribute `disable="true"`. For a replica table, the corresponding column is automatically removed. For a history table, the column remains but is marked as disabled. See Disabling history on a specific field or group [p 276] and Disabling replication on a specific field or group [p 285].
- The facets of a field can be modified, except for the facets listed under Limitations/restrictions [p 291].

The following changes are accepted, but they can lead to loss of data. Data should be manually migrated, by exporting then re-importing an XML or archive file, because these changes are considered to be a combination of deletion and creation.

- A field can be renamed.
- The type of a field can be changed.

Index-level evolutions

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

50.2 **Limitations/restrictions**

> **Note**
>
> All limitations listed in this section that affect mapped mode can be worked around by purging the mapped table database resources. For the procedure to purge mapped table database resources, see Purging database resources of mapped tables [p 374].

Limitations related to primary key evolutions

When a primary key definition is modified:

- In semantic mode, the existing records are only loaded into the cache if they:
    - Respect the uniqueness constraint of the primary key,
    - Comply with the new structure of the primary key.
- In mapped mode, the underlying RDBMS only accepts a primary key modification if all table records respect its uniqueness and non-nullity constraints. In particular, if a table already has existing records:
    - Adding a new field to the primary key requires assigning a default value to this field. Workaround: first add the field, value it for the existing records, then add the field to the primary key.
    - Removing an existing field from the primary key will be rejected if it would cause the existing records to no longer have a unique primay key (assigning a default value makes no change in this case).
    - It is generally not possible to rename a field of the primary key; more formally, it is only possible if the field was not needed for making all primary keys unique. Indeed, renaming a field translates to a combination of deletion and creation; consequently, the operation will be rejected if it would cause the existing records to no longer have a unique primay key (assigning a default value makes no change in this case).

Limitations related to foreign key evolutions

- When the declaration of a facet `osd:tableRef` is added or modified, or the primary key of the target table of a facet `osd:tableRef` is modified:
    - In semantic mode, the existing values for this field are only loaded into the cache if they comply with the new structure of the target primary key.
    - In mapped mode, the structure of a foreign key field is set to match that of the target primary key. A single field declaring an `osd:tableRef` constraint may then be split into a number of columns, whose number and types correspond to that of the target primary key. Hence, the following cases of evolutions will have an impact on the structure of the mapped table:
        - declaring a new `osd:tableRef` constraint on a table field;
        - removing an existing `osd:tableRef` constraint on a table field;
        - adding (resp. removing) a column to (resp. from) a primary key referenced by an existing `osd:tableRef` constraint;
        - modifying the type or path for any column of a primary key referenced by an existing `osd:tableRef` constraint.

      These cases of evolution will translate to a combination of field deletions and/or creations. Consequently, the existing data should be migrated manually.

Limitations related to field-level evolutions

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

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

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

  PostgreSQL has the same limitations, but additionally, any modification of a field type (including modifications of its length) will invalidate all related prepared statements, and require restarting the application server.
- When a cardinality of an element is modified:
  - In semantic mode, this change is supported. However, two cases are distinguished:
    - When changing a single element to an aggregated list, the previous single value is preserved and added to the new aggregated list.
    - When changing an aggregated list to a single element, only the last value of the aggregated list is preserved in the single element. 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.

Other

CHAPTER **51**

Inheritance and value resolution

This chapter contains the following topics:

51.1 Overview

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

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

> **Note**
>
> Inheritance mechanisms described in this chapter should not be confused with "structural inheritance", which usually applies to models and is proposed in UML class diagrams, for example.

See also *Inheritance (glossary)* *[p 23]*

Data set inheritance

Data set inheritance is particularly useful when data applies to global enterprise contexts, such as subsidiaries or business partners.

Based on a hierarchy of data sets, it is possible to factorize common data into the root or intermediate data sets and define specialized data in specific contexts.

The data set inheritance mechanisms are detailed below in Data set inheritance [p 297].

Inherited fields

Contrary to data set inheritance, which exploits a global built-in relationships between data sets, inherited fields exploit finer-grained dependencies that are specific to the data structure. It allows factorizing and specializing data at the business entities-level.

For example, if the model specifies that a 'Product' is associated with a 'FamilyOfProducts', it is possible that some attributes of 'Product' inherit their values from the attributes defined in its associated 'FamilyOfProducts'.

> **Note**
>
> It is not possible to use both attribute inheritance and data set inheritance in the same data set.

Computed values (functions)

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

The function is able to take into account the current context, such as the values of the current record or computations based on another table, and to make requests to third-party systems.

See also *Computed values* *[p 482]*

51.2 Data set inheritance

Data set inheritance declaration

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

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

The element `osd:inheritance` defines the property `dataSetInheritance` to specify the use of inheritance on 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, the inheritance mechanism is disabled.

Value lookup mechanism

The data set inheritance lookup mechanism for values proceeds as follows:

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

   It can be explicitly `null`.

2. Otherwise, looks up the first locally defined value according to the built-in child-to-parent relationship of the data set in the hierarchy of data sets.

3. If no locally defined value is found, the default value is returned.

   If no default value is defined, `null` is returned.

   Note: Default values cannot be defined on:

   - A single primary key node
   - Auto-incremented nodes
   - Nodes defining a computed value

Record lookup mechanism

Like values, table records can also be inherited as a a unit by multiple contexts, but they can also be partially redefined (*overwritten*), defined for a specific context (*root mode*), or be *occulted*.

Formally, a table record has one of four distinct definition modes:

root record	Locally defined in the table and has no parent. This means that no record with the same primary key exists in the parent table, or that this parent is an occulting record.
overwriting record	Locally defined in the table and has a parent record. This means that a record with the same primary key exists in the parent table, and that this parent is not an occulting record. The overwriting record inherits its values from its parent, except for the values that it explicitly redefines.
inherited record	Not locally defined in the current table and has a parent record. All values are inherited. Functions are always resolved in the current record context and are not inherited.
occulting record	Specifies that if a parent with same primary key is defined, this parent will not be visible in table descendants.

See also *Data set inheritance* *[p 109]*

Defining inheritance behavior at the table level

It is also possible to specify management rules in the declaration of a table in the data model.

See also *Properties related to data set inheritance [p 455]*

51.3 Inherited fields

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

Field inheritance declaration

Specific inheritance must be specified on terminal nodes in the underlying data model and is declared as follows:

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

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

If `sourceRecord` is not defined in the data model, the inherited fields are fetched from the current record.

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

The following conditions must be satisfied for specific inheritance:

- The element `sourceNode` is mandatory.
- The expression for the path to the source record must be a consistent path of foreign keys, from the current element to the source record. This expression must involve only one-to-one and zero-to-one relationships.
- The `sourceRecord` cannot contain any aggregated list elements.
- Each element of the `sourceRecord` must be a foreign key.
- If the inherited field is also a foreign key, the `sourceRecord` cannot refer to itself to get the path to the source record of the inherited value.
- Every element of the `sourceRecord` must exist.
- The source node must belong to the table containing the source record.
- The source node must be terminal.
- The source node must be writeable.
- The source node type must be compatible with the current node type
- The source node cardinalities must be compatible with those of the current node.
- The source node cannot be the same as the inherited field if the fields to inherit from are fetched into the same record.

Value lookup mechanism

The lookup mechanism for inherited fields values proceeds as follows:

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

   It can be explicitly `null`
2. Otherwise, looks up the source record and value to inherit from, according to the properties that are defined in the data model.
3. The process is recursive; if the source node does not locally define a value, it is then looked up according to the inheritance behavior of the source node.

51.4 Optimize & Refactor service

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

- **Handles duplicated values:** Detects and removes all parameter values that are duplicates of the inherited value.
- **Mutualizes common values:** Detects and mutualizes the common values among the descendants of a common ancestor.

Procedure details

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 ancestors.
> - Table optimization is performed on records with the same primary key.
> - Inherited fields are not optimized.
> - *The optimization and refactoring functions do not modify the resolved view of a 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 restrictive access rights on a profile.

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

CHAPTER 52 Permissions

Permissions dictate the access each user has to data and actions.

This chapter contains the following topics:

52.1 Overview

Permissions are related to whether actions are authorized or not. They are also related to access rights, that is, whether an entity is hidden, read, or read-write. The main entities controlled by permissions are:

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

Users, roles and profiles

The definition and resolution of permissions make extensive use of the notion of *profiles*, which is the generic term applied to users or roles.

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

These relationships are defined in the user and roles directory. See Users and roles directory [p 391].

Special definitions:

- An *administrator* is a member of the built-in role 'ADMINISTRATOR'.
- An *owner of a data set* is a member of the *owner* attribute specified in the information of a root data set. In this case, the built-in role 'OWNER' is activated when permissions are resolved in the context of the data set.
- An *owner of a data space* is a member of the *owner* attribute specified for a 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 for a particular entity.

User-defined permission rules are created through the user interface. See the section Defining permissions and access rights in the user interface [p 306].

Programmatic permission rules can be created by developers. See the section Defining permissions and access rights with programmatic rules [p 309].

Resolution of permissions

Permissions are always resolved in the context of an authenticated user session. Thus, the defined permission rules can take user's roles into account.

> **Note**
>
> In the Java API, the class `SessionPermissions`API provides access to the resolved permissions.

See also *Resolving permissions* *[p 309]*

Owner and administrator permissions on a data set

An administrator or owner of a data set can perform the following actions:

- Manage its permissions
- Change its owner, if the data set is a root data set
- Change its localized labels and descriptions

Attention

While the definition of permissions can restrict an administrator or data set owner's right to view data or perform certain actions, it remains possible for them to modify their own access, as they will always have access to permissions management.

Owner and administrator permissions on a data space

To be a *super owner* of a data space, a user must either:

- Own the data space and be allowed to manage its permissions, or
- Own a data space that is an ancestor of the current data space and be allowed to manage the permissions of that ancestor data space.

An administrator or super owner of a data space can perform the following actions:

- Manage its permissions of data space.
- Change its owner
- Lock it or unlock it
- Change its localized labels and descriptions

Attention

While the definition of permissions can restrict an administrator or data space owner's right to view data or perform certain actions, it remains possible for them to modify their own access, as they will always have access to permissions management.

Impact of merge on permissions

When a data space is merged, the permissions of the child data set are merged with those of the parent data space if and only if the user specifies to do so during the merge process. The permissions of its parent data space are never impacted.

If some elements are hidden for the profile attempting to perform a merge, it will not be possible to proceed as the impacts of the merge on data will not be fully visible.

Important considerations about permissions

The following statements should be kept in mind while working with permissions:

- Resolution of custom display labels for tables ('defaultLabel' property) and relationships ('display' property) ignores permission, and fields usually hidden due to access rights restrictions will be displayed in such labels. As a result, these labels should not contain any confidential field. Otherwise, a permission strategy should also be defined to restrict the display of the whole label.
- When a procedure disables all permission checks by using `ProcedureContext.setAllPrivileges`API, the client code must check that the current user session is allowed to run the procedure.
- To increase permissions resolution, a dedicated cache is implemented at the session-level; it only takes user-defined permission rules into account, not programmatic rules (which are not cached since they are contextual and dynamic). The session cache life cycle depends on the context, as described hereafter:
    - In the UI, the cache is cleared for every non-ajax event (i.e on page display, pop-up opening, etc.).
    - In programmatic procedures, the cache lasts until the end of the procedure, unless explicitly cleared (see below).

    Attention

    When modifying permissions in a procedure context (by importing an EBX5 archive or merging a data space programmatically), the session cache **must** be cleared via a call to `Session.clearCache`API. Otherwise, these modifications will not be reflected until the end of the procedure.

52.2 **Defining permissions and access rights in the user interface**

Each level has a similar schema, which allows defining permission rules for profiles.

Data space permissions

For a given data space, the allowable permissions for each profile are as follows:

Data space access	Defines access rights as read-write, read-only or hidden.
Restriction	Indicates whether this data space profile-permission association should have priority over other permissions rules.
Create a child data space	Indicates whether the profile can create child data spaces from the current data space.
Create a child snapshot	Indicates whether the profile can create snapshots of the current data space.
Initiate merge	Indicates whether the profile can merge the current data space with its parent data space.
Export archive	Indicates whether the profile can export the current data space as an archive.
Import archive	Indicates whether the profile can import an archive into the current data space.
Close a data space	Indicates whether the profile can close the current data space.
Close a snapshot	Indicates whether the profile can close a snapshot of the current data space.
Rights on services	Indicates if a profile has the right to execute services on the data space. By default, all data space services are allowed. An administrator or super owner of the current data space or a given user who is allowed to modify permissions on the current data space can modify these permissions to restrict data space services for certain profiles.
Permissions of child data space when created	Defines the same data space permissions as the current data space on child data spaces.

Access rights on data spaces

Mode	Authorization
Write	• Can view the data space. • Can access data sets according to data set permissions.
Read-only	• Can view the data space and its snapshots. • Can view child data spaces, if allowed by permissions. • Can view contents of the data space, though cannot modify them.
Hidden	• Can neither see the data space nor its snapshots. • If allowed to view child data space, can see the current data space but cannot select it. • Cannot access the data space contents, including data sets. • Cannot perform any actions on the data space.

Permissions on a new data space

When a user creates a child data space, the permissions of this new data space are automatically assigned to the profile's owner, based on the permissions defined under 'Permissions of child data space when created' in the parent data space. If multiple permissions are defined for the owner through different roles, resolved permissions [p 309] are applied.

Attention

Only the administrator and owner of a data space can define a new owner for the data space or modify associated permissions. They can also modify the general information on the data space and permissions of the users.

Furthermore, in a workflow, when using a "Create a data space" or "Create a snapshot" built-in script task, resolved permissions are computed using the owner defined in the script task's configuration, rather than the current session. This is because, in these cases, the current session is associated with a system user.

Permissions on data set

For a given data set, the allowable permissions for each profile are as follows:

Actions on data sets

Restricted mode	Indicates whether this data set profile-permission association should have priority over other permissions rules.
Create a child data set	Indicates whether the profile has the right to create a child data set of the current data set.
Duplicate data set	Indicates whether the profile has the right to duplicate the current data set.
Change the data set parent	Indicates whether the profile has the right to change the parent data set of a given child data set.

Actions on tables

For a specific table, permissions can be defined, which override the default permissions defined in its containing data set. The allowable permissions for each profile are as follows:

Create a record	Indicates whether the profile has the right to create records in the table.
Modify a record	Indicates whether the profile has the right to modify records in the table.
Hide a record	Indicates whether the profile has the right to hide records in the table.
Delete a record	Indicates whether the profile has the right to delete records in the table.

Access rights on node values

Permissions defined on specific terminal nodes override their default access rights.

Read-write	Can view and modify node values.
Read	Can view nodes, but cannot modify their values.
Hidden	Cannot view nodes.

Permissions on services

By default, all data set services are allowed. An administrator or an owner of the current data space can modify these permissions to restrict data set services for certain profiles.

52.3 Defining permissions and access rights with programmatic rules

Access rights can be defined on a data set using programmatic rules on target nodes. This can be done for all data set nodes, table records, a specific node, or a complex node and its child nodes. To define programmatic rules for access rights, create a class that implements the Java interface `SchemaExtensions`.

See also *`SchemaExtensions`*API

It is also possible to define a permission for a service using a programmatic rule.

To define programmatic rules for service permissions, create a class that implements the Java interface `ServicePermission`.

See also *`ServicePermission`*API

One example of using programmatic rules is updating a field according to the value of another field.

Attention

Only one programmatic access right can be defined for each node, data set or record. Thus, defining a new programmatic access right will replace an existing one.

52.4 Resolving permissions

The resolution of permissions is always performed in the context of a user session according to the associated roles. In general, resolution of restrictive permissions is performed between a given level and its parent level. Thus, at any given level, a user cannot have a higher permission than the one resolved at a parent level.

Programmatic permissions are always considered to be restrictive.

Restriction policies

When a restrictive permission is defined for a profile, it takes precedence over the other permissions potentially accorded by the user's other roles. Generally:

- If restrictions are defined, the minimum permissions based on all the restricted profiles is applied.
- If no restrictions are defined, the maximum permissions based on all profiles is applied.

Permission resolution examples

Given two profiles *P1* and *P2* concerning the same user, the following table lists the possibilities when resolving that user's access to a service.

P1 authorization	P2 authorization	Permission resolution
Allowed	Allowed	Allowed. Restrictions do not make any difference.
Forbidden	Forbidden	Forbidden. Restrictions do not make any difference.
Allowed	Forbidden	Allowed, unless P2's authorization is a restriction.
Forbidden	Allowed	Allowed, unless P1's authorization is a restriction.

In another example, a data space can be hidden from all users by defining a restrictive association between the built-in profile "Profile.EVERYONE" and the access right "hidden".

52.5 **Resolving access rights**

Resolving access rights defined using the user interface

Access rights defined using the user interface are resolved on three levels: data space, data set and node.

If a profile is associated with restrictive access rights at a given level, the minimum of all restrictive rights defined at that level is resolved. If no restrictions are defined at that level, the maximum of all access rights defined at that level is resolved.

At any given level, the most restrictive access rights between those resolved at this level and higher levels are applied. For instance, if a user's data set access permissions resolve to read-write access, but the container data space only allows read access, the user will only have read-only access to this data set.

> **Note**
>
> The data set inheritance mechanism applies to both values and access rights. That is, access rights defined on a data set will be applied to its child data sets. It is possible to override these rights in the child data set.

Access rights resolution example

In this example, there are three users who belong to the following defined roles and profiles:

User	Profile
User 1	• user1 • role A • role B
User 2	• role A • role B • role C
User 3	• user3 • role A • role C

The access rights of the profiles on a given element are as follows:

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

After resolution based on the role and profile access rights above, the right that are applied to each user are as follows:

User	Resolved access rights
User 1	Hidden
User 2	Read
User 3	Read/Write

Resolving data space and snapshot access rights

At data space level, access rights are resolved as follows:

- If a user has several rights defined through multiple profiles:
  - If the rights include restrictions, the minimum of the restrictive profile-rights associations is applied.
  - Otherwise, the maximum of the profile-rights associations is applied.
- If the user has no rights defined:
  - If the user is an administrator or is the owner of the data space, read-write access is given for this data space.
  - Otherwise, the data space will be hidden.

Resolving data set access rights

At the data set level, the same principle applies as at the data space level. After resolving the access rights at the data set level alone, the final access rights are determined by taking the minimum rights between the resolved data space rights and the resolved data set rights. For example, if a data space is resolved to be read-only for a user and one of its data sets is resolved to be read-write, the user will only have read-only access to that data set.

Resolving node access rights

At the node level, the same principle applies as at the data space and data set levels. After resolving the access rights at the node level alone, the final access rights are determined by taking the minimum rights between the resolved data set rights and the resolved node rights.

Specific access rights can be defined at the node level. If no specific access right is defined, the default access right is used for the resolution process.

> **Note**
>
> The resolution procedure is slightly different for table and table child nodes.

Special case for table and table child nodes

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

For each user-defined permission rule that matches one of the user's profiles, the access rights for *N* are either:

1. The locally defined access rights for *N*;
2. Inherited from the access rights defined on the table node;
3. Inherited from the default access rights for data set values.

All matching user-defined permission rules are used to resolve the access rights for *N*. Resolution is done according to the restriction policy [p 309].

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

Resolving access rights defined with programmatic rules

There are three levels of resolution for programmatic access right rules: 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 used by the resolution procedure.

Rule resolution on data set

For a data set, the last rule set is considered as the resolved rule

Rule resolution on record

For a record, the resolved rule is the minimum between the resolved rule set on the data set and the rule set on this record. See `SchemaExtensionsContext.setAccessRuleOnOccurrence`API for more details.

Access rule resolution on node

For a node that is a child node of a record, the resolved rule is the minimum between the resolved rule the record and the rule set on this node.

Otherwise, the resolved rule is the minimum between the resolved rule set on the data set and the rule set on this node. See `SchemaExtensionsContext.setAccessRuleOnNode`API for more details.

Display policy for foreign key drop-down menus

If a record is hidden due to access rules, it will not appear in foreign key drop-down menus.

Attention

The resolved access rights on a data set or data set node is the minimum between the resolved access rights defined in the user interface and the resolved programmatic rules, if any.

52.6 Resolving permissions for actions and services

The resolution of actions and services available to a given user follows the same process as the resolution of access rights.

When several action lists are defined on a data set for a given profile, the final actions list is dynamically generated after evaluating each individual action across all lists associated with the same user.

If some profile-action list associations are restrictive, an action is forbidden if at least one restrictive association forbids it. If there are no restrictive associations, an action is allowed if any association allows it.

Action and service permission resolution example

In this example, there are two users belonging to different roles and profiles:

User	Profiles
User 1	• user1 • role A • role B
User 2	• role C • role D

The permissions associated with the roles and profiles on a given table are as follows:

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

The actions available to each user after rights resolution are as follows:

Users	Available actions
User 1	Create a record
	Duplicate a record
User 2	Create a record
	Modify a record
	Duplicate a record

CHAPTER **53**

Criteria editor

This chapter contains the following topics:

53.1 **Overview**

The criteria editor is included in several different areas of the user interface. It allows defining table filters, as well as validation and computation rules on data. This editor is based on the XPath 1.0 Recommendation.

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

See also *Supported XPath syntax [p 253]*

53.2 **Conditional blocks**

Conditional blocks are made up of atomic criteria and other conditional blocks. They express a condition based on the criteria. The following types of blocks exist:

- **No criteria match**: None of the criteria in the block match.
- **Not all criteria match**: At least one criterion in the block does no match.
- **All criteria match**: All criteria in the block match.
- **At least one criterion matches**: One or more of the criteria match.

53.3 Atomic criteria

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

Field	Specifies the field of the table to which the criterion applies.
Operator	Specifies the operator used. Available operators depend on the data type of the field.
Value	Specifies the value or expression. See Expression [p 316] below.
Code only	If checked, specifies searching the underlying values for the field instead of labels, which are searched by default.

Expression

The expression can either be a fixed value or a formula. When creating a filter, only fixed values are authorized. During creation of a validation or computation rule, a formula can be created using the wizard.

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

CHAPTER 54

Performance guidelines

This chapter contains the following topics:

54.1 Basic performance checklist

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 the usual performance bottlenecks.

Expensive programmatic extensions

For reference, the table below details the programmatic extensions that can be implemented.

Use case	Programmatic extensions that can be involved
Validation	• **programmatic constraints** `Constraint`API • **computed values** `ValueFunction`API
Table access	• **record-level permission rules** `SchemaExtensionsContext.setAccessRuleOnOccurrence`API • **programmatic filters** `AdaptationFilter`API
EBX5 content display	• **computed values** `ValueFunction`API • **UI Components** `UIBeanEditor`API • **node-level permission rules** `SchemaExtensionsContext.setAccessRuleOnNode`API
Data update	• **triggers** `Package com.orchestranetworks.schema.trigger`API

For large volumes of data, cumbersome algorithms have a serious impact on performance. For example, a constraint algorithm's complexity is $O(n^2)$. If the data size is 100, the resulting cost is proportional to 10 000 (this generally produces an immediate result). However, if the data size is 10 000, the resulting cost will be proportional to 10 000 000.

Another reason for slow performance is calling external resources. Local caching usually solves this type of problem.

If one of the use cases above displays poor performance, it is recommended to track the problem either through code analysis or using a Java profiling tool.

Directory integration

Authentication and permissions management involve the user and roles directory [p 391].

If a specific directory implementation is deployed and accesses an external directory, it can be useful to ensure that local caching is performed. In particular, one of the most frequently called methods is `Directory.isUserInRole`API.

Aggregated lists

In a data model, when an element's cardinality constraint `maxOccurs` is greater than 1 and no `osd:table` is declared on this element, it is implemented as a Java `List`. This type of element is called an aggregated list [p 447], as opposed to a table.

It is important to consider that there is no specific optimization when accessing aggregated lists in terms of iterations, user interface display, etc. Besides performance concerns, aggregated lists are limited with regard to many functionalities that are supported by tables. See tables introduction [p 451] for a list of these features.

Attention

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

54.2 Checklist for data space usage

Data spaces [p 72] available in semantic mode, are an invaluable tool for managing complex data life cycles. While this feature brings great flexibility, it also implies a certain overhead cost, which should be taken into consideration for optimizing usage patterns.

This section reviews the most common performance issues that can appear in case of an intensive use of many data spaces containing large tables, and how to avoid them.

> **Note**
>
> Sometimes, the use of data spaces is not strictly needed. As an extreme example, consider the case where every transaction triggers the following actions:
>
> 1. A data space is created.
> 2. The transaction modifies some data.
> 3. 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. Thus, using `Procedure`API already provides sufficient isolation to avoid conflicts from concurrent operations. It would then be more efficient to directly do the modifications in the target data space, and get rid of the steps which concern branching and merging.
>
> 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, it is probably sufficient to do so directly on the main branch. In fact, it would be neither practical nor sustainable, with regard to file tagging/copying, if every file modification involved branching the whole project, modifying the files, then merging the dedicated branch.

Insufficient memory

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

This memory swap overhead can only occur for tables in a data space with an on-demand loading strategy [p 321].

Such an issue can be detected by looking at the monitoring log file [p 321]. If it occurs, various actions can be considered:

- reducing the number of child data spaces that contain large tables;
- reducing the number of indexes specifically defined for large tables (in semantic mode, the current limitation applies: for a given table, the content of its indexes is not shared among child data spaces);
- using relational mode instead of semantic mode;
- or (obviously) allocating more memory, or optimizing the memory used by applications for non-EBX5 objects.

See also

Memory management [p 320]

Relational mode [p 269]

Transaction cancels

In semantic mode, when a transaction has performed some updates in the current data space and then aborts, loaded indexes of the modified tables are reset. If updates on a large table are often cancelled and, at the same time, this table is intensively accessed, then the work related to index rebuild will slow down the access to the table; moreover, the induced memory allocation and garbage collection can reduce the overall performance.

See also

Functional guard and exceptions `TableTrigger`API

`Procedure`API

Reorganization of database tables

As with any database, inserting and deleting large volumes of data may lead to fragmented data, which can deteriorate performance over time. To resolve the issue, reorganizing the impacted database tables is necessary. See Monitoring and cleanup of relational database [p 370].

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

See also *Monitoring and cleanup of relational database [p 370]*

54.3 Memory management

Loading strategy

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

The following table details the loading modes which are available in semantic mode. Note that the application server must be restarted so as to take into account any loading strategy change.

On-demand loading and unloading	In this default mode, each resource in a data space is loaded or built only when it is needed. The resources of the data space are "soft"-referenced using the standard Java `SoftReference` class. This implies that each resource can be unloaded "at the discretion of the garbage collector in response to memory demand". The main advantage of this mode is the ability to free memory when needed. As a counterpart, this implies a load/ build cost when an accessed resource has not yet been loaded since the server started up, or if it has been unloaded since.
Forced loading	If the forced loading strategy is enabled for a data space or snapshot, its resources are loaded asynchronously at server startup. Each resource of the data space is maintained in memory until the server is shut down or the data space is closed. This mode is particularly recommended for long-living data spaces and/or those that are used heavily, namely any data space that serves as a reference.
Forced loading and prevalidation	This strategy is similar to the forced loading strategy, except that the content of the loaded data space or snapshot will also be validated upon server startup.

Monitoring

Indications of EBX5 load activity are provided by monitoring the underlying database, and also by the 'monitoring' logging category [p 352].

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

Tuning memory

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

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

54.4 **Validation**

The internal incremental validation framework will optimize the work required when updates occur. The incremental validation process behaves as follows:

- The first call to a data set validation report performs a full validation of the data set. The loading strategy [p 320] can also specify a data space to be prevalidated at server startup.
- Data updates will transparently and asynchronously maintain the validation report, insofar as the updated nodes specify explicit dependencies. For example, standard and static facets, foreign key constraints, dynamics facets, selection nodes specify explicit dependencies.
- If a mass update is executed or if there are too many validation messages, the incremental validation process is stopped. The next call to the validation report will then trigger a full validation.
- If a transaction is cancelled, the validation state of the updated data set is reset. The next call to the validation report will trigger a full validation as well.

Certain nodes are systematically revalidated, however, even if no updates have occurred since the last validation. These are the nodes with *unknown dependencies*. A node has unknown dependencies if:

- It specifies a **programmatic constraint** `Constraint`API in the default *unknown dependencies* mode,
- It declares a **computed value** `ValueFunction`API, or
- It declares a dynamic facet that depends on a node that is itself a **computed value** `ValueFunction`API.

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.

> **Note**
>
> It is possible for an administrator user to manually reset the validation report of a data set. This option is available from the validation report section in EBX5.

54.5 **Mass updates**

Mass updates can involve several hundreds of thousands of insertions, modifications and deletions. These updates are usually infrequent (usually initial data imports), or are performed non-interactively (nightly batches). Thus, performance for these updates is less critical than for frequent or interactive operations. However, similar to classic batch processing, it has certain specific issues.

Transaction boundaries

It is generally not advised to use a single transaction when the number of atomic updates in the transaction is beyond the order of 10 ^{4}. Large transactions require a lot of resources, in particular, memory, from EBX5 and from the underlying database.

To reduce transaction size, it is possible to:

- Specify the property ebx.manager.import.commit.threshold [p 357]. However, this property is only used for interactive archive imports performed from the EBX5 user interface.
- Explicitly specify a **commit threshold** `ProcedureContext.setCommitThreshold`API inside the batch procedure.
- Structurally limit the transaction scope by implementing `Procedure`API for a part of the task and executing it as many times as necessary.

On the other hand, specifying a very small transaction size can also hinder performance due to the persistent tasks that need to be done for each commit.

> **Note**
>
> If intermediate commits are a problem because transactional atomicity is no longer guaranteed, it is recommended to execute the mass update inside a dedicated data space. This data space will be created just before the mass update. If the update does not complete successfully, the data space must be closed, and the update reattempted after correcting the reason for the initial failure. If it succeeds, the data space can be safely merged into the original data space.

Triggers

If required, triggers can be deactivated using the method `ProcedureContext.setTriggerActivation`API.

54.6 Accessing tables

Functionalities

Tables are commonly accessed through EBX5 and also through the `Request`API API and data services. This access involves a unique set of functions, including a *dynamic resolution* process. This process behaves as follows:

- **Inheritance**: Inheritance in the data set tree takes into account records and values that are defined in the parent data set, using a recursive process. Also, in a root data set, a record can inherit some of its values from the data model default values, defined by the `xs:default` attribute.
- **Value computation**: A node declared as an `osd:function` is always computed on the fly when the value is accessed. See `ValueFunction.getValue`API.
- **Filtering**: An XPath predicate [p 253], a **programmatic filter** `AdaptationFilter`API, or a record-level **permission rule** `SchemaExtensionsContext.setAccessRuleOnOccurrence`API requires a selection of records.
- **Sort**: A sort of the resulting records can be performed.

Accessing tables in semantic mode

Architecture and design

In order to improve the speed of operations on tables, indexes are managed by the 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 specialized design for indexing mechanisms. This design can be summarized as follows:

- *Indexes* maintain an in-memory data structure on a full table.
- An index is not persisted, and building it requires loading all table blocks from the database. This tradeoff still results in a beneficial outcome, since the index can be retained in memory for longer than its corresponding table blocks.

Attention

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

Performance considerations

The request optimizer favors the use of indexes when computing a request result.

Attention

- Only XPath filters are taken into account for index optimization.
- Non-primary-key indexes are not taken into account for child data sets.

Assuming the indexes are already built, the impacts on performance are as follows:

1. If the request does not involve filtering, programmatic rules, or sorting, accessing its first few rows (these fetched by a paged view) is almost instantaneous.
2. If the request can be resolved without an extra sort step (this is the case if it has no sort criteria, or if its sort criteria relate to those of the index used for computing the request), accessing the first few rows of a table should be fast. More precisely, it depends on the cost of the specific filtering algorithm that is executed when fetching at least 2000 records.
3. Both cases above guarantee an access time that is independent of the size of the table, and provides a view sorted by the index used. If an extra sort is required, the time taken by the first access depends on the table size according to an `Nlog(N)` function, where `N` is the number of records in the resolved view.

Note

The paginated requests automatically add the primary key to the end of the specified criterion, in order to ensure consistent ordering. Thus, the primary key fields should also be added to the end of any index intended to improve the performance of paginated requests. These include tabular and hierarchical views, and drop-down menus for table references.

If indexes are not yet built, or have been unloaded, additional time is required. The build time is `O(Nlog(N))`.

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

It is possible to get information through the monitoring [p 321] and request logging categories.

Other operations on tables

The new records creations or record insertions depend on the primary key index. Thus, a creation becomes almost immediate if this index is already loaded.

Accessing tables in relational mode

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

- Handling of all request sort criteria, by translating them to an `ORDER BY` clause.
- Whenever possible, handling of the request filters, by translating them to a `WHERE` clause.

> **Attention**
>
> Only XPath filters are taken into account for index optimization. If the request includes non-optimizable filters, table rows will be fetched from the database, then filtered in Java memory by EBX5, until the requested page size is reached. This is not as efficient as filtering on the database side (especially regarding I/O).

Information on the transmitted SQL request is logged to the category *persistence*. See Configuring the EBX5 logs [p 352].

Indexing

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

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

Setting a fetch size

In order to improve performance, a fetch size should be set according to the expected size of the result of the request on a table. If no fetch size is set, the default value will be used.

- In semantic mode, the default value is 2000.
- In mapped mode, the default value is assigned by the JDBC driver: 10 for Oracle and 0 for PostgreSQL.

> **Attention**
>
> On PostgreSQL, the default value of 0 instructs the JDBC driver to fetch the whole result set at once, which could lead to an `OutOfMemoryError` when retrieving large amounts of data. On the other hand, using fetchSize on PostgreSQL will invalidate server-side cursors are at the end of the transaction. If, in the same thread, you first fetch a result set with a fetchsize, then execute a procedure that commits the transaction, then, accessing the next result will raise an exception.

See also

*`Request.setFetchSize`*API

*`RequestResult`*API

Administration Guide

Installation & configuration

CHAPTER 55

Supported environments

This chapter contains the following topics:

55.1 Browsing environment

Supported web browsers

The EBX5 web interface supports the following browsers:

Microsoft Internet Explorer 8, 9, 10, 11	Compatibility mode is not supported. Performance limitations: page loading in IE8 is generally 20 times slower than in other browsers; in IE9, IE10 and IE11, page loading is two times slower. This issue is observed when forms have many input components, and particularly many multi-occurrenced groups.
Mozilla Firefox 3.6 and latest version (see details)	As Mozilla Firefox is updated frequently, Orchestra Networks only fully supports version 3.6, while testing and making a best effort to support the latest version available.
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.

Screen resolution

The minimum screen resolution for EBX5 is 1024x768, and only the default browser zoom (100%) is supported.

Refreshing pages

Browser page refreshes are not supported by EBX5. When a page refresh is performed, the last user action is re-executed, therefore could result 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 for the user interface to work properly:

- JavaScript
- Ajax
- Pop-ups

Attention

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

55.2 Supported application servers

EBX5 supports the following configurations:

- Java Runtime Environment: JRE 1.5 or higher version, which obviously includes the limitations specified by the Java Virtual Machine implementation vendor. For example, 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 2.4 (or higher), for example Tomcat 5.5 or higher, WebSphere Application Server 6.1 or higher, WebLogic Server 9.2 or higher. See Java EE deployment overview [p 346].
- The application server must use 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`.

Attention

Limitations apply regarding clustering and hot deployment/undeployment:

Clustering: EBX5 does not include a cache synchronization mechanism, thus it cannot be deployed into a cluster of active instances. See Technical architecture [p 366] for more information.

Hot deployment/undeployment: EBX5 does not support hot deployment/undeployment of web applications registered as EBX5 modules or of EBX5 built-in web applications.

55.3 Supported databases

The 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.	Mapped table modes are not supported (History [p 275], Relational mode [p 269] and Replication [p 283]).
Oracle Database 10gR2 or higher.	Relational mode [p 269] is only supported for Oracle 11g R2 or higher. 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`. The distinction of `null` values encounters certain limitations. On simple `xs:string` elements, Oracle does not support the distinction between empty strings and `null` values. See Empty string management [p 482] for more information. The user with which EBX5 connects to the database requires the following privileges: • CREATE SESSION, • CREATE TABLE, • ALTER SESSION, • CREATE SEQUENCE, • A non-null quota on its default tablespace.
PostgreSQL 8.4 or higher.	When using PostgreSQL as the underlying database, a request fetch size must be set, otherwise the JDBC driver will fetch the whole result set at once. This could lead to an `OutOfMemoryError` when retrieving large amounts of data. See `Request.setFetchSize`API. The user with which EBX5 connects to the database requires the CONNECT privilege on the database hosting the EBX repository. Beyond this, the default privileges on the public schema of this database are suitable.
Microsoft SQL Server 2008 or higher.	When used with Microsoft SQL Server, EBX5 uses the default database collation to compare and sort strings stored in the database. This applies to strings used in the data model definition, as well as data stored in relational and history tables. The default database collation can be specified when the database is created. Otherwise, the database engine

	server collation is used. To avoid naming conflicts or unexpected behavior, a case- and accent-sensitive collation as the default database collation must be used (the collation name is suffixed by "CS_AS" or the collation is binary). The default setting to enforce transaction isolation on SQL Server follows a pessimistic model. Rows are locked to prevent any read/write concurrent accesses. This may cause liveliness issues for mapped tables (history or relational). To avoid such issues, it is recommended to activate snapshot isolation on your SQL Server database. The user with which EBX5 connects to the database requires the following privileges: • CONNECT, SELECT and CREATE TABLE on the database hosting the EBX repository, • ALTER, CONTROL, UPDATE, INSERT, DELETE on its default schema.
H2 v1.3.170 or higher.	H2 is not supported for production environments. The default H2 database settings do not allow consistent reads when records are modified. Relational tables are locked following a pessimistic model. To prevent concurrency issues, it is possible to activate the MVCC feature. Note, however, that the H2 documentation states this feature is not yet fully tested.

For other relational databases, 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 [p 272].

See also

Repository administration [p 366]

Data source of the EBX5 repository [p 338]

Configuring the EBX5 repository [p 349]

CHAPTER **56**

Java EE deployment

This chapter contains the following topics:

56.1 Introduction

This chapter details deployment specifications for EBX5 on a Java application server. For specific information regarding supported application servers and inherent limitations, see Supported environments. [p 329]

56.2 Software components

EBX5 uses the following components:

- Library `ebx.jar`
- Third-party Java libraries [p 333]
- EBX5 built-in web applications [p 336] and optional custom web applications [p 336]
- EBX5 main configuration file [p 347]
- EBX5 repository [p 366]
- Default user and roles directory [p 391], integrated within the EBX5 repository, or a third-party system (LDAP, RDBMS) for user authentication

See also *Supported environments [p 329]*

56.3 Third-party libraries

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

Database drivers

The EBX5 repository requires a database. Generally, the required driver is configured along with a data source, if one is used. Depending on the database defined in the main configuration file, one of the following drivers is required. Keep in mind that, whichever database you use, the version of the JDBC client driver must be equal or higher to the version of the database server.

H2	Version 1.3.170 validated. Any 1.3.X version should be suitable. Note that H2 is not supported for production environments. http://www.h2database.com/
Oracle JDBC	Oracle database 10gR2, 11gR2 and 12cR1 are validated on their latest patchset update. Determine the driver that should be used according to the database server version and the Java runtime environment version. You can include `ojdbc6.jar` or `ojdbc7.jar` depending on the Java runtime environment version you use; it does not make any difference as EBX5 does not make use of any JDBC 4.1 specific feature. Oracle database JDBC drivers download.
DB2 JDBC	DB2 UDB V9.7 validated. JAR files to include: `db2jcc_license_cu.jar` and `db2jcc.jar`
SQL Server JDBC	SQL Server 2008, 2008R2 and 2012, with all corrective and maintenance patches applied, are validated. Remember to use an up-to-date JDBC driver, as some difficulties have been encountered with older versions. JAR file to include: `sqljdbc4.jar`
PostgreSQL	PostgreSQL 8.4, 9.0, 9.1, 9.2 and 9.3 validated Include the latest JDBC driver released for your database server and Java runtime environment. PostgreSQL JDBC drivers download.

See also

Data source of the EBX5 repository [p 338]

Configuring the EBX5 repository [p 349]

SMTP and emails

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

The following libraries are used by email features in EBX5. See Activating and configuring SMTP and e-mails [p 353] 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 [p 347]*

Java Message Service (JMS)

When using JMS, version 1.1 or higher is required.

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

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

> **Note**
>
> In EBX5, the supported JMS model is exclusively Point-to-Point (PTP). PTP systems allow working with queues of messages.

See also *EBX5 main configuration file [p 347]*

56.4 Web applications

EBX5 provides pre-packaged EARs that can be deployed directly if your enterprise has no custom EBX5 module web applications to add. If you are deploying custom web applications as EBX5 modules, it is recommended to 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 [p 346] 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 [p 337] 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:** The data model assistant requires the `ebx-manager` user interface web application to be deployed.
`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 `ebx-manager` user interface web application.

Custom web applications

Every custom web application that is defined as an EBX5 module [p 437] must be registered using the method `ModulesRegister.registerWebApp()` in the Java API. Registration of modules is explained in the EBX5 modules [p 438] chapter.

See also

Registration [p 438]

Module registration [p 359]

56.5 Deployment details

Introduction

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

Attention

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

See also

EBX5 main configuration file *[p 347]*

Supported application servers *[p 330]*

User interface and web access

The web application '`ebx`' (packaged as `ebx.war`) contains the servlet `FrontServlet`, which handles initialization and serves as the sole 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`:

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

Configuring the application server for 'FrontServlet'

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

  For example, on Tomcat, this configuration is 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 [p 349] 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	Default JNDI name	Description
`jdbc/EBX_REPOSITORY`	Weblogic: `EBX_REPOSITORY` JBoss: `java:/EBX_REPOSITORY`	JDBC data source for EBX5 Repository. Java type: `javax.sql.DataSource`

See also

Configuring the EBX5 repository [p 349]

Rules for the access to the database and user privileges [p 367]

[HTML documentation] Oracle data source configuration on WebSphere Application Server 6

[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 [p 353] 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	Default JNDI name	Description
`mail/EBX_MAIL_SESSION`	Weblogic: `EBX_MAIL_SESSION` JBoss: `java:/EBX_MAIL_SESSION`	Java Mail session used to send e-mails from EBX5. Java type: `javax.mail.Session`

JMS connection factory

> **Note**
>
> If the EBX5 main configuration does not activate JMS through the property `ebx.jms.activate`, the environment entry below will be ignored by the EBX5 runtime. See JMS [p 354] 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	Default JNDI name	Description	Required
`jms/EBX_JMSConnectionFactory`	Weblogic: `EBX_JMSConnectionFactory` JBoss: `java:/EBX_JMSConnectionFactory`	JMS connection factory used by EBX5 to create connections with the JMS provider configured in the operational environment of the application server. Java type: `javax.jms.ConnectionFactory`	Yes

> **Note**
>
> For deployment on JBoss and WebLogic application servers with JNDI capabilities, you must 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 [p 339] and the following queues, declared in the `WEB-INF/web.xml` deployment descriptor of the '`ebx`' web application. This is the only method for configuring JMS for data services.

When a SOAP request is received, the SOAP response is optionally returned if the header field `JMSReplyTo` is defined. If so, the fields `JMSCorrelationID` and `JMSType` are retained.

See JMS [p 354] 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 [p 354] in the EBX5 main configuration properties for more information on this property.

Reserved resource name	Default JNDI name	Description	Required
`jms/EBX_QueueIn`	Weblogic: `EBX_QueueIn` JBoss: `java:/jms/EBX_QueueIn`	JMS queue for incoming SOAP requests sent to EBX5 by other applications. Java type: `javax.jms.Queue`	No
`jms/EBX_QueueFailure`	Weblogic: `EBX_QueueFailure` JBoss: `java:/jms/EBX_QueueFailure`	JMS queue for failures. It contains incoming SOAP requests for which an error has occurred. This allows replaying these messages if necessary. Java type: `javax.jms.Queue` **Note:** For this property to be read, the main configuration must also activate the queue for failures through the property `ebx.jms.activate.queueFailure`. See JMS [p 354] 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 [p 339] 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 [p 354] and Distributed data delivery (D3) [p 354] in the EBX5 main configuration properties for more information on these properties.

Common declarations on master and slave nodes (for shared queues)

Reserved resource name	Default JNDI name	Description
`jms/EBX_D3MasterQueue`	Weblogic: `EBX_D3MasterQueue` JBoss: `java:/jms/EBX_D3MasterQueue`	D3 master JMS queue (only for D3 mode 'slave' or 'hub'). It specifies the queue name used to send SOAP requests to the D3 master node. The message producer sets the master repository ID as a value of the header field `JMSType`. Java type: `javax.jms.Queue`
`jms/EBX_D3ReplyQueue`	Weblogic: `EBX_D3ReplyQueue` JBoss: `java:/jms/EBX_D3ReplyQueue`	D3 Reply JMS queue (for all D3 modes except 'single' mode). It specifies the name of the reply queue for receiving SOAP responses. The consumption is filtered using the header field `JMSCorrelationID`. Java type: `javax.jms.Queue`
`jms/EBX_D3ArchiveQueue`	Weblogic: `EBX_D3ArchiveQueue` JBoss: `java:/jms/EBX_D3ArchiveQueue`	D3 JMS Archive queue (for all D3 modes except 'single' mode). It specifies the name of the transfer archive queue used by D3 node. The consumption is filtered using the header field `JMSCorrelationID`. If the archive weight is higher than the threshold specified in the property `ebx.jms.d3.archiveMaxSizeInKB`, the archive will be divided into several sequences. Therefore, the consumption is filtered using the header fields `JMSXGroupID` and `JMSXGroupSeq` instead. Java type: `javax.jms.Queue`
`jms/EBX_D3CommunicationQueue`	WebLogic: `EBX_D3CommunicationQueue` JBoss: `java:/jms/EBX_D3CommunicationQueue`	D3 JMS Communication queue (for all D3 modes except 'single' mode). It specifies the name of the communication queue where the requests are received. The consumption is filtered using the header field `JMSType` which corresponds to the current repository ID. Java type: `javax.jms.Queue`

> **Note**
>
> These JNDI names are set by default, but can be modified inside the web application archive `ebx.war`, included in `EBX5ForWebLogic.ear` (if using Weblogic) or `EBX5.ear` (if using JBoss, Websphere or other application servers).

Optional declarations on master nodes (for slave-specific queues)

> **Note**
>
> Used for ascending compatibility prior to 5.5.0 or for mono-directional queues topology.

The deployment descriptor of the master node must be manually modified by declaring specific communication and archive queues for each slave node. It consists in adding resource names in 'web.xml' inside 'ebx.war'. The slave-specific queues can be used by one or several slaves.

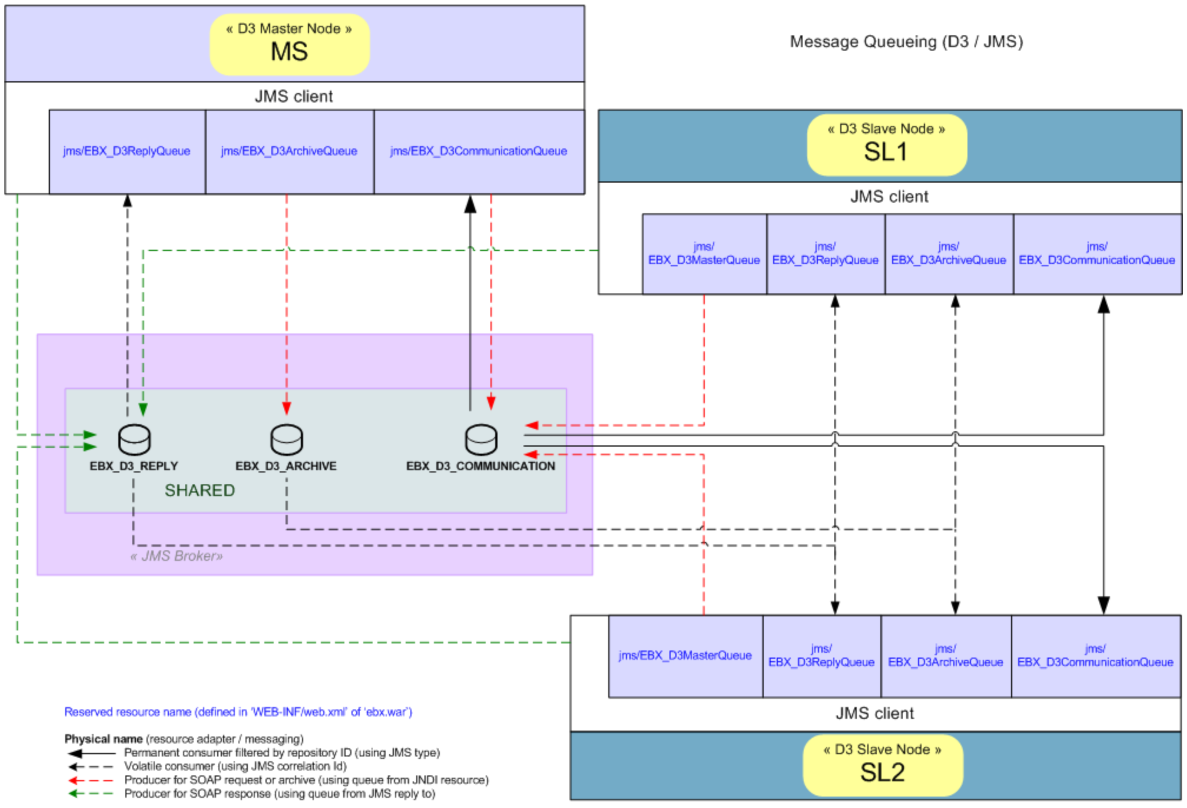
The resource names can be freely named, but the physical names of their associated queue must correspond to the definition on slaves for resources `jms/EBX_D3ArchiveQueue` and `jms/EBX_D3CommunicationQueue`.

> **Note**
>
> Physical queue names matching: On registration, the slave node sends the communication and archive physical queue names. These queues are matched by physical queue name among all resources declared on the master node. If unmatched, the registration fails.

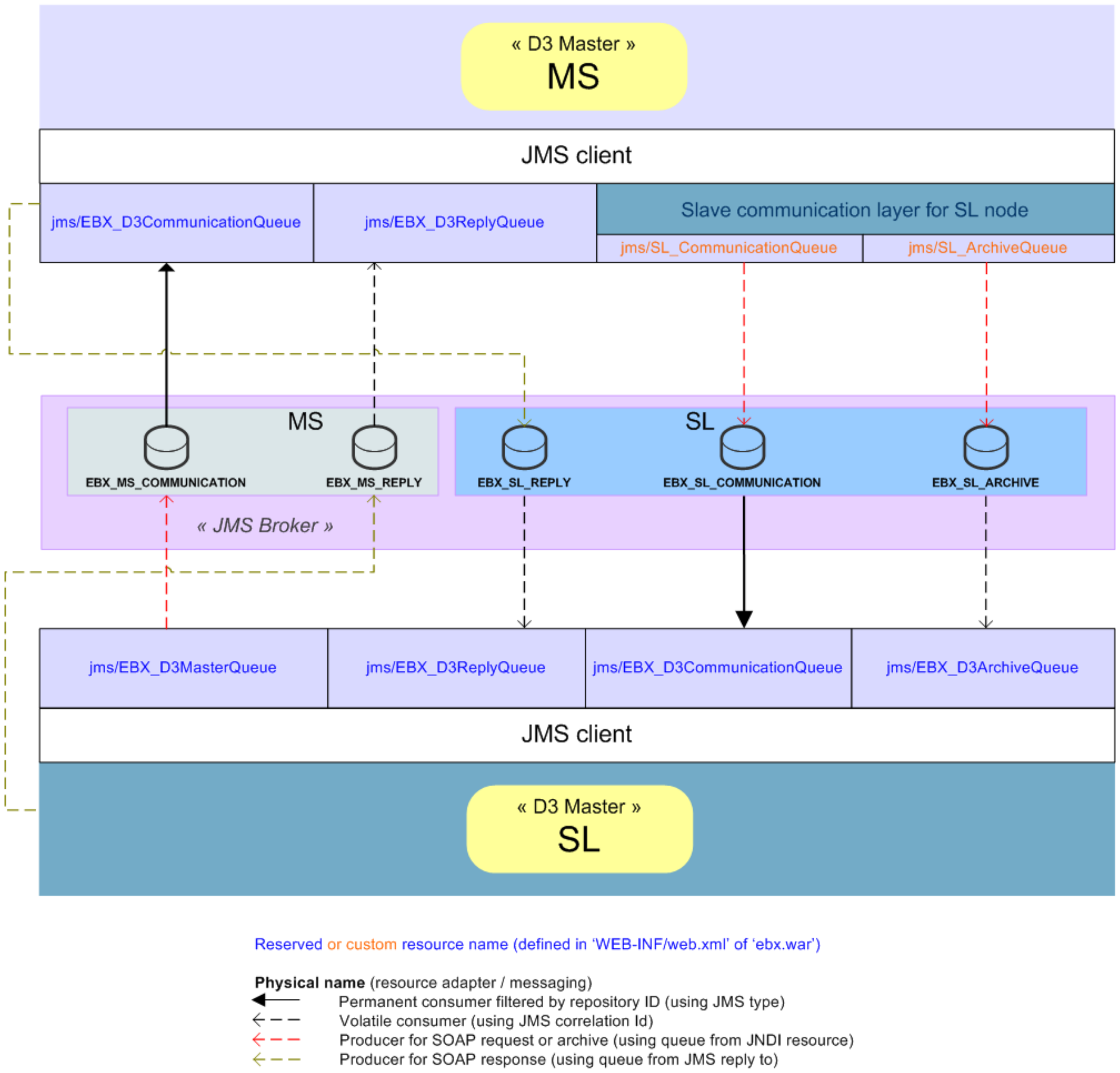
Examples of JMS configuration

	Shared queues	**Specific queues**
Master-Slave architecture	Between a master node and two slave nodes with shared queues [p 342]	Between a master node and a slave node with slave-specific queues [p 343]
Hub-Hub architecture	Between two hub nodes with shared queues [p 344]	Between two hub nodes with slave-specific queues [p 345]

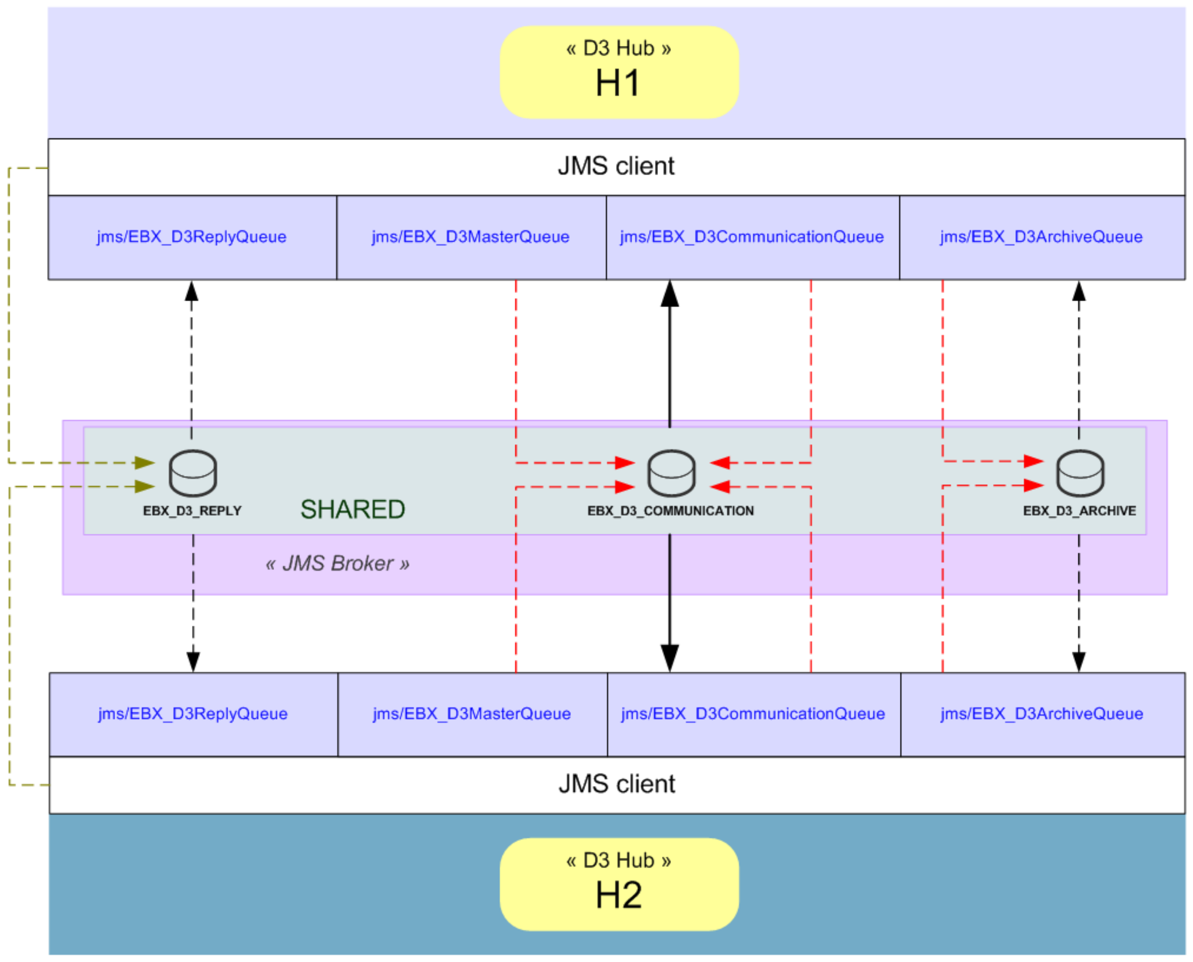
Between a master node and two slave nodes with shared queues


« D3 Master Node »
MS
Message Queueing (D3 / JMS)
JMS client
jms/EBX_D3ReplyQueue
jms/EBX_D3ArchiveQueue
jms/EBX_D3CommunicationQueue
« D3 Slave Node »
SL1
JMS client
jms/EBX_D3MasterQueue
jms/EBX_D3ReplyQueue
jms/EBX_D3ArchiveQueue
jms/EBX_D3CommunicationQueue
EBX_D3_REPLY
EBX_D3_ARCHIVE
EBX_D3_COMMUNICATION
SHARED
« JMS Broker»
jms/EBX_D3MasterQueue
jms/EBX_D3ReplyQueue
jms/EBX_D3ArchiveQueue
jms/EBX_D3CommunicationQueue
JMS client
« D3 Slave Node »
SL2
Reserved resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')
Physical name (resource adapter / messaging)
Permanent consumer filtered by repository ID (using JMS type)
Volatile consumer (using JMS correlation Id)
Producer for SOAP request or archive (using queue from JNDI resource)
Producer for SOAP response (using queue from JMS reply to)

Between a master node and a slave node with slave-specific queues


« D3 Master »
MS
JMS client
jms/EBX_D3CommunicationQueue
jms/EBX_D3ReplyQueue
Slave communication layer for SL node
jms/SL_CommunicationQueue
jms/SL_ArchiveQueue
MS
EBX_MS_COMMUNICATION
EBX_MS_REPLY
SL
EBX_SL_REPLY
EBX_SL_COMMUNICATION
EBX_SL_ARCHIVE
« JMS Broker »
jms/EBX_D3MasterQueue
jms/EBX_D3ReplyQueue
jms/EBX_D3CommunicationQueue
jms/EBX_D3ArchiveQueue
JMS client
« D3 Master »
SL
Reserved or custom resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')
Physical name (resource adapter / messaging)
Permanent consumer filtered by repository ID (using JMS type)
Volatile consumer (using JMS correlation Id)
Producer for SOAP request or archive (using queue from JNDI resource)
Producer for SOAP response (using queue from JMS reply to)

Between two hub nodes with shared queues

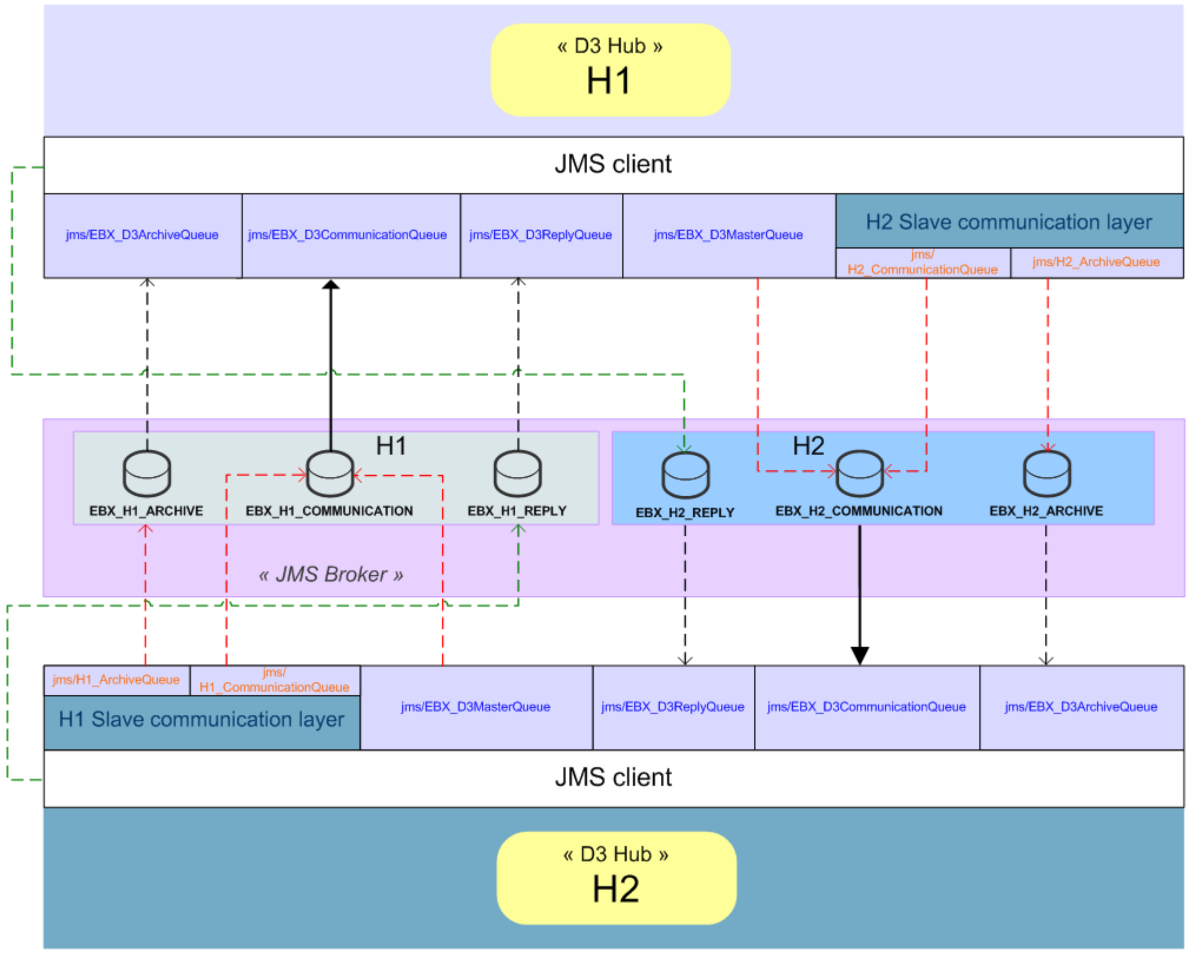


Reserved resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

Physical name (resource adapter / messaging)

◀──	Permanent consumer filtered by repository ID (using JMS type)
← – –	Volatile consumer (using JMS correlation Id)
← – –	Producer for SOAP request or archive (using queue from JNDI resource)
← – –	Producer for SOAP response (using queue from JMS reply to)

Between two hub nodes with slave-specific queues



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

Physical name (resource adapter / messaging)

⟵	Permanent consumer filtered by repository ID (using JMS type)
← – –	Volatile consumer (using JMS correlation Id)
← – –	Producer for SOAP request or archive (using queue from JNDI resource)
← – –	Producer for SOAP response (using queue from JMS reply to)

56.6 **Java EE deployment examples**

EBX5 can be deployed on any Java EE application server that supports Servlet 2.4 or higher. The following documentation on Java EE deployment and installation notes are available:

- *[HTML documentation]* Installation on Tomcat 5.5
- *[HTML documentation]* Installation on WebSphere 6
- *[HTML documentation]* Installation on WebLogic 9.2

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` or other libraries in the class-loading system.

CHAPTER 57

EBX5 main configuration file

This chapter contains the following topics:

57.1 Overview

The EBX5 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 [p 337]. Administrators can also perform further configuration through the user interface, which is then stored in the EBX5 repository.

See also

Deployment details [p 337]

Front end administration *[p 377]*

Location of the file

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

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

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

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

> **Note**
>
> In addition to specifying properties in the main configuration file, you can also set the values of properties directly in the system properties. For example, using the `-D` argument of the `java` command-line command.

Custom properties and variable substitution

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

For example, the default configuration file provided with EBX5 uses the custom property `ebx.home` to set a default common directory, which is then included in other properties.

57.2 Setting an EBX5 license key

See also *Installing a repository using the configuration assistant* *[p 361]*

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

57.3 Setting the EBX5 root directory

The EBX5 root directory contains 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
```

57.4 Configuring the EBX5 repository

Before configuring the persistence properties of the EBX5 repository, carefully read the section Technical architecture [p 366] in the chapter 'Repository administration'.

The required library (driver) for each supported database is described in the chapter Database drivers [p 334].

See also

Repository administration [p 366]

Rules for the access to the database and user privileges [p 367]

Supported databases [p 331]

Data source of the EBX5 repository [p 338]

Database drivers [p 334]

```
################################################################
## 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=yyyyyyyy


################################################################
## 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=yyyyyyyy


## Activate to use VARCHAR2 instead of NVARCHAR2 on Oracle (this should be required only in rare cases).
#ebx.persistence.oracle.useVARCHAR2=false


################################################################
## 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=yyyyyyyy

################################################################
## 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=yyyyyyyy

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

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

57.5 Configuring the user and roles directory

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

See also

Users and roles directory *[p 391]*

*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
```

57.6 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
```

57.7 Activating the XML audit trail

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

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

See also *Audit trail* *[p 395]*

57.8 **Configuring the EBX5 logs**

The most important logging categories are:

ebx.log4j.category.log.kernel	Logs for EBX5 main features, processes, exceptions and compilation results of modules and data models.
ebx.log4j.category.log.workflow	Logs for main features, warnings and exceptions about workflow.
ebx.log4j.category.log.persistence	Logs related to communication with the underlying database.
ebx.log4j.category.log.setup	Logs for compilation results of all EBX5 objects, except for modules and data models.
ebx.log4j.category.log.validation	Logs for data sets validation results.
ebx.log4j.category.log.mail	Logs for the activity related to the emails sent by the server (see Activating and configuring SMTP and e-mails [p 353]). **Note:** This category must not use the Custom SMTP appender [p 353] in order to prevent infinite loops.
ebx.log4j.category.log.d3	Logs for D3 events on EBX5.
ebx.log4j.category.log.dataservices	Logs for data service events in EBX5.
ebx.log4j.category.log.monitoring	Raw logs for monitoring [p 321].
ebx.log4j.category.log.request	The optimization strategy for every `Request`API issued on a semantic table in the EBX5 repository.

Some of these categories can also be written to through custom code using the `LoggingCategory`API interface.

```
 #################################################
## Log4J properties:
##
## We have some specific syntax extensions:
## - Appender ebxFile:<aFileName>
## Defines a file appender with default settings (threshold=DEBUG)
##
## - property log.defaultConversionPattern is set by Java
#################################################
#ebx.log4j.debug=true
#ebx.log4j.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
```

Custom 'ebxFile' appender

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

The property `ebx.log4j.appender.ebxFile.backup.Threshold` allows defining 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
```

Custom 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* [p 353]

57.9 Activating and configuring SMTP and e-mails

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

See also *Mail sessions* [p 338]

```
#################################################
## 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 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
```

57.10 Configuring data services

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

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

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

# Upon WSDL generation, specifies if the target namespace value corresponds to the content before 5.5.0 'ebx-
services'
# or 'urn:ebx:ebx-services' in conformity with the URI syntax.
# If the parameter is set to true, there is no check of the target namespace as URI at the WSDL generation.
# If unspecified, default is false.
#ebx.dataservices.wsdlTargetNamespace.disabledCheck=false
```

57.11 Activating and configuring JMS

See also *JMS for data services* [p 339]

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

## Activates JMS (true or false, default is false).
## If activated, the deployer must ensure that the entry 'jms/EBX_JMSConnectionFactory'
## are bound in the operational environment of the application server.
## The entry 'jms/EBX_QueueIn' should also be bound to enable handling Data Services request
## using JMS.
#ebx.jms.activate=false

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

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

57.12 Configuring distributed data delivery (D3)

See Configuring D3 nodes [p 423] for the main configuration file properties pertaining to D3.

See also

JMS for distributed data delivery (D3) [p 340]

Introduction to D3 [p 412]

57.13 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.

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

In this case, the server name and the port are obtained from the initial request sent to 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. This configuration also allows using SSL security on the front-end server.

The web server sends requests to the application server according to a path in the URL. This `servletAlias` path is specified in the main configuration file.

The web server provides all external resources. These 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 properties are 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 parameter encoding

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

The properties are 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
```

57.14 **Configuring failover**

These parameters are used to configure failover mode and activation key, as well as heartbeat logging in `DEBUG` mode.

See also *Failover with hot-standby* *[p 367]*

```
#################################################
## Mode used to qualify the way in which a server accesses the repository.
## Possible values are: unique, failovermain, failoverstandby.
## Default value is: unique.
#################################################
#ebx.repository.ownership.mode=unique

## Activation key used in case of failover. The backup server must include this
## key in the HTTP request used to transfer exclusive ownership of the repository.
## The activation key must be an alphanumeric ASCII string longer than 8 characters.
#ebx.repository.ownership.activationkey=

## Specifies whether to hide heartbeat logging in DEBUG mode.
## Default value is true.
#ebx.repository.ownership.hideHeartBeatLogForDebug=true
```

57.15 **Tuning the EBX5 repository**

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

The properties are configured as follows:

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

# Specifies whether the validation report should be kept in memory,
# regardless of the loading strategy of the data space.
# Default value is true. However, it is recommended to deactivate it
# when the repository contains a large number of open data spaces and
# data sets.
#ebx.validation.report.keepInMemory=false
```

57.16 Miscellaneous

Activating data workflows

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

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

Log procedure starts

This parameter specifies whether starts of procedure execution are logged.

```
##################################################
## Specifies whether transaction starts are logged. Default is false.
##################################################
ebx.logs.logTransactionStart = true
```

Log validation starts

This parameter specifies whether starts of data sets validation are logged.

```
##################################################
## Specifies whether validation starts are logged. Default is false.
##################################################
ebx.logs.logValidationStart = true
```

Deployment site identification

This parameter allows specifying the email address to which technical log emails are sent.

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

Dynamically reloading the main configuration

Some parameters can be dynamically reloaded, without restarting EBX5. The parameter `thisfile.checks.intervalInSeconds` indicates how frequently the main configuration file is checked.

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

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

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

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 overrided by each EBX5 module.
#################################################
templates.checksIfUpdated=true
```

Debug mode in the EBX5 module web pages (only used 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 be registered. 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, as 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* *[p 437]*

```
#################################################
## 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
```

EBX5 run mode

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

When running in *development* mode, the development tools [p 507] 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 **Validation** 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/*:
```

Downgrading to IE8 display

With Internet Explorer, if EBX5 is included in a portal which is forced to "Compatibility Mode", the IE compatibility mode [p 385] of EBX5 is not applied by Internet Explorer. It is thus impossible to ensure a good display of graphics (such as images or beautiful title fonts). The display must be downgraded to IE8 mode for all Internet Explorer versions starting from 9.

```
##################################################
## Specifies whether the Internet Explorer display must be downgraded to version 8.
## Default value is false.
##################################################
#ebx.browser.ie.forceIE8=true
```

CHAPTER 58

Installing a repository using the configuration assistant

The EBX5 Configuration Assistant helps with 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. See Java EE deployment [p 333].

> **Note**
>
> The EBX5 main configuration file must also be correctly configured. See EBX5 main configuration file [p 347].

This chapter contains the following topics:

1. License key
2. Configuration steps

58.1 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 EBX5 main configuration file, 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.

58.2 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

In order to proceed with the configuration, you must read and accept the product license agreement.

Configuring the repository

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

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

See also *EBX5 main configuration file* *[p 347]*

Defining users in the default directory

If a custom user and roles directory is not defined in the EBX5 main configuration file, the configuration assistant allows you to define default users for the default user and roles directory.

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

See also *Users and roles directory* *[p 391]*

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 this documentation.

This chapter contains the following topics:

59.1 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.

59.2 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

This chapter contains the following topics:

60.1 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 [p 331], accessed through a configured data source [p 349].
- A repository cannot be shared by multiple JVMs at any given time. However, a failover architecture may be used. These aspects are detailed in the sections Single JVM per repository [p 367] and Failover with hot-standby [p 367]. Furthermore, to achieve horizontal scalability, an alternative is to deploy a distributed data delivery (D3) [p 412] environment.
- A single relational database instance can support multiple EBX5 repositories (used by distinct JVMs). It is then required that they specify distinct table prefixes using the property `ebx.persistence.table.prefix`.

See also

Configuring the EBX5 repository [p 349]

Supported databases [p 331]

Data source of the EBX5 repository [p 338]

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 [p 272].

It is recommended for the database user specified by the configured data source [p 349] to have 'create/alter' privileges on tables, indexes and sequences. This allows automatic repository installation and upgrades [p 369]. If this is not the case, it is still possible to perform manual repository installation and upgrades [p 369], however, Relational mode [p 269] and History [p 275] will be more inconvenient to maintain since modification of concerned models will require performing manual steps.

See also

SQL access to history [p 278]

Accessing a replica table using SQL [p 285]

SQL access to data in relational mode [p 272]

Data source of the EBX5 repository [p 338]

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 potentially even corruption of data in the repository.

EBX5 performs checks to enforce this restriction. Before the repository becomes available, the repository must first acquire 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 performed by repeatedly tagging a technical table in the relational database. The shutdown command for the application server ensures that the tag on this technical table is removed. If the server shuts down unexpectedly, the tag may be left in the table. If this occurs, the server must wait several additional seconds upon restart to ensure that the table is not being updated by another live process.

Attention

To avoid an additional wait period upon 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 in an active/passive cluster. To ensure this is the case, the main server must declare the property `ebx.repository.ownership.mode=failovermain`. The main server claims the repository database, as in the case of a single server.

A backup server can still start up, but it will not have access to the repository. It must declare the property `ebx.repository.ownership.mode=failoverstandby` to act as the backup server. Once started, the backup server is registered in the connection log. Its status can be retrieved using the Java

API or through an HTTP request, as described in the section Repository status information and logs [p 368] 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 file. See Configuring failover [p 357].
- Using the Java API, call the method `RepositoryStatus.wakeFromStandby`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 can the backup server start using the repository.

Repository status information and logs

A log of all attempted Java process connections to the repository is available in the Administration area under 'History and logs' > 'Repository connection log'.

The status of the repository may be retrieved using the methods in the `RepositoryStatus`API API.

It is also possible to get repository status information using a HTTP request that includes the parameter `repositoryInformationRequest` with one of following values:

`state`	The state of the repository in terms of ownership registration. • `D`: Java process is stopped. • `O`: Java process has exclusive ownership of the database. • `S`: Java process is started in failover standby mode, but is not yet allowed to interact with the repository. • `N`: Java process has tried to take ownership of the database but failed because another process is holding it.
`heart_beat_count`	The number of times that the repository has made contact since associating with the database.
`info`	Detailed information for the end user regarding the repository's registration status. The format of this information may be subject to modifications in future without explicit warning.

60.2 Repository installation and upgrade

Automatic installation and upgrades

If the specified user for the EBX5 data source has permission to create and alter tables, indexes and sequences in the relational database, then the repository installation or upgrade is done automatically. Otherwise, an administrator must manually carry out the procedure detailed in the next section Manual installation and upgrades [p 369].

> **Note**
>
> Concerning relational mode and history, declaring a table in relational mode or history mode creates a dedicated table in the database. Similarly, in these modes, modifying the structure of an existing table alters its declaration in the database. Modifications may take place at any time for reasons other than repository creation. The EBX5 application database user must have 'create/alter' privileges on tables and indexes, since these actions must be carried out from EBX5.

Manual installation and upgrades

If the user specified for the EBX5 data source does *not* have permission to create or alter tables, indexes or sequences in the relational database, the administrator must carefully execute all the SQL scripts, located in the `ebx.software/files/ddl/` directory (select the folder corresponding to your RDBMS and then execute the scripts in ascending order).

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 modified accordingly in the provided scripts.

See also *Configuring the EBX5 repository [p 349]*

60.3 Repository backup

A global backup of the EBX5 repository must be delegated to the underlying RDBMS. The database administrator must use the standard backup procedures of the underlying database.

60.4 Archives directory

Archives are stored in a sub-directory named `archives` within `ebx.repository.directory` (see configuration [p 347]). This directory is automatically created during the first export from EBX5.

> **Attention**
>
> If manually creating this directory, ensure that the EBX5 process has read-write access to it. Furthermore, the administrator is responsible for cleaning this directory, as EBX5 does not maintain it.

> **Note**
>
> A file transferred between two EBX5 environments must be done outside of the product using tools such as FTP or simple file copies by network sharing.

60.5 Repository attributes

A repository has the following attributes:

repositoryId	Uniquely identifies a repository within the scope of the enterprise). It is 48 bits (6 bytes) and is usually represented as 12 hexadecimal digits. This information is used for generating UUIDs (Universally Unique Identifier) for entities created in the repository, as well as transactions logged in history tables or the XML audit trail. This identifier acts as the 'UUID node' part, as specified by *RFC 4122*.
repository label	Provides a user-friendly label that identifies the purpose and context of the repository. For example, "Production environment".
store format	Identifies the underlying persistence system, including the current version of its structure.

60.6 Monitoring and cleanup of relational database

Some entities accumulate during the execution of EBX5.

> **Attention**
>
> It is the *administrator's responsibility* to monitor and to clean up these entities.

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 [p 269] 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 with 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 suffixed by `HTB` and `BRV`, as well as their indexes. This is especially true for large repositories where many data spaces are created and deleted.

Database statistics

The performance of requests executed by EBX5 requires that the database has computed up-to-date statistics on its tables. Since database engines regularly schedule statistics updates, this should generally not be an issue. Yet, it could be needed to explicitly update the statistics in cases where tables are heavily modified over a short period of time (e.g. by an import which creates many records).

Impact on UI

Some UI component use these statistics to adapt their behavior, in order to prevent user from executing costly requests unwillingly.

For example, the combo box will not automatically search on user input if the table has a large volume of records. This behavior may also occur if the data base's statistics are not up to date, because a table may be considered as having a large volume of records even if it is not the case.

Cleaning up data spaces, snapshots, and history

A full cleanup 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 `Repository.closeHome`API.
- Using the data service close data space and close snapshot operations. See Closing a data space or snapshot [p 235] 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 `Repository.deleteHome`API and `RepositoryPurge.markHomeForHistoryPurge`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 `RepositoryPurge.purgeAll`API.
- Using the task scheduler. See Task scheduler [p 405] for more information.

The purge process is logged into the directory `${ebx.repository.directory}/db.purge/`.

Cleaning up tables having unreadable records

Some data model model evolutions can lead to unreadable data:

- A column containing null values is added to the primary key of a table.
- The type of a column has changed to a different type with no possible conversion.

In these situations, records that do not match the new table definition are no longer visible, but remain persisted in the table. (This allows retrieving the records if switching back to the previous table definition). When such records are encountered, an informative error is recorded in EBX5 logs.

EBX5 provides the option to clean the records that no longer conform to the model, once the new version of the data model is stabilized. This allows recovering space in the database and getting rid of error messages. Please proceed carefully, as this operation permanently removes all unreadable records from the selected table, and cannot be undone.

Cleaning up unreadable records is done by selecting Administration **Actions > Clean up unreadable records** in the navigation pane.

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 will 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 [p 372]. 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. Scheduled tasks constantly add to this table as they are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

User interactions

User interactions are used by the 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 section. It is recommended to regularly monitor the user interactions table, as well as to clean it, if needed.

Workflow history

The workflow events are persisted in the workflow history table, in the 'Workflow' section of the Administration area. Data workflows constantly add to this table as they are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

60.7 Monitoring and cleanup of file system

Attention

In order to guarantee the correct operation of EBX5, the disk usage and disk availability of the following directories must be supervised by the administrator, as EBX5 does not perform any cleanup:

- **XML audit trail**: `${ebx.repository.directory}/History/`
- **Archives**: `${ebx.repository.directory}/archives/`
- **Logs**: ebx.logs.directory [p 352]
- **Temporary directory**: ebx.temp.directory

Attention

For **XML audit trail**, if large transactions are executed with full update details activated (the default setting), the required disk space can be quite high.

Attention

For pagination in the data services `getChanges` operation, a persistent store is used in the **Temporary directory**. Large changes may require a large amount of disk space.

See also

XML audit Trail [p 396]

Tuning the EBX5 repository [p 357]

CHAPTER **61**

Front end administration

Several administrative tasks can be performed from the Administration area of the EBX5 user interface.

This chapter contains the following topics:

61.1 Administration tools

By clicking on 'Actions' under the 'Administration' area, basic administration tools are available.

System information

This page lists information related to the repository, the operating system and EBX5 configuration.

Modules and data models

This tool lists all registered modules as well as existing data models.

User sessions

This tool lists all user sessions and allows terminating active sessions.

61.2 Data spaces

Some data space administrative tasks can be performed from the Administration area of EBX5 by selecting 'Data spaces'.

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* *[p 78]*

From this section, it is also possible to close open data spaces, reopen previously 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* *[p 371]*

Data space permissions

This table lists all the existing permission rules defined on all the data spaces in the repository. You can view the permission rules and modify their information.

See also *Data space permissions* *[p 79]*

Repository history

The table 'Deleted data spacess/napshots' lists all the data spaces that have already been purged from the repository.

61.3 User directory

If the default directory provided and integrated into EBX5 is used, the 'Directory' administration section allows defining which users can connect and what their roles are.

See also *Users and roles directory* *[p 391]*

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 users 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 in this table.

61.4 Global permissions

Global permission rules can be created in EBX5.

The 'Display area' property allows restricting access to areas of the software. To define the access rules, select 'Global permissions' in the 'Administration' area.

Profile	Indicates on which profile the rule will be applied.
Restriction policy	Indicates if the permissions defined here restrict the ones defined for other profiles. See the Restriction policy concept [p 309] for more information.
Data Spaces	Defines permissions for the Data Spaces area.
Data Models	Defines permissions for the Data Models area.
Workflow Models	Defines permissions for the Workflow Models area.
Data Workflows	Defines permissions for Data Workflow.
Data Services	Defines permissions for the Data Services area. And Defines permissions for the WSDL connector http(s) servlet. [p 206].
Administration	Defines permissions for the Administration area.

> **Note**
>
> Permissions can be defined by administrators and by the data space or data set owner.

61.5 Administrative delegation

An administrator can delegate administrative rights to a non-administrator user, either for specific actions or for all activities.

The administrative delegation is defined under 'Administration' in the global permissions profile.

If all necessary administrative rights have been delegated to non-administrator users, it becomes possible to disable the built-in 'ADMINISTRATOR' role.

See also *Configuring the user and roles directory [p 350]*

61.6 Perspectives configuration

An unlimited number of perspectives can be configured. To each perspective corresponds a data set. Data set inheritance is available and allows sharing easily items between perspectives.

See also *Inheritance [p 23]*

A perspective menu is displayed on the left side vertical navigation bar and consists of menu items of different types.

Menu item types:

Section Menu Item	This is a top level menu item. It contains other menu items.
Group Menu Item	This is a container for other menu items.
Action Menu Item	This menu item displays a UI service in the workspace area.

Perspectives available by default

Root perspective

The root perspective can define menu items that are inherited by all other perspectives. It can be made available to end users but this is not recommended.

Advanced perspective

This perspective menus cannot be customized. The advanced perspective is available by default to all end-users but access can be restricted.

Note: Administrators can always access this perspective even when it is deactivated.

Perspective creation

To create a perspective, select the parent (or root) perspective and click on the + sign to create its corresponding child data set.

See also *Creating an inheriting child data set* *[p 93]*

Perspective properties

The available perspective properties are:

Activated	Indicates that the perspective is visible to authorized users.
Allowed profiles	The list of authorized user profiles for the perspective.
Default selection	The menu item that is selected by default. This property is not available for the advanced perspective.

Menu view

This view displays the perspective menu. It is a hierarchical table view.

From this view, a user can create, delete or reorder menu item records.

See also *Hierarchical table view* *[p 23]*

Menu item properties

The available perspective properties are:

Type	The menu item type. **See also** *Menu item types* *[p 380]*
Parent	The parent of the menu item. This property is not available for section menu items.
Label	The menu item label. The label is optional for action menu items. If not specified, the label will be dynamically generated by EBX when the menu item is displayed.
Icon	The icon for the menu item. Icon can be either "standard" (provided by EBX5) or an image, specified by an URL, that can be hosted on any web server. This property is not available for section menu items.
Top separator	Indicates that the menu item section has a top separator. This property is only available for section menu items.
Action	The UI service to execute when the user clicks on the menu item. **See also** *User interface services* *[p 175]* If an end user is authorized to view the perspective but not to execute the UI service, an access denied message will be displayed when the user clicks on the menu item. This property is only available for action menu items.
Selection on close	The menu item that will be selected when the service terminates. Built-in services use this property when the user clicks on the 'Close' button. This property is only available for action menu items.

61.7 User interface configuration

Some options are available in the Administration area for configuring the web interface, in the 'User interface' section.

Attention

Be careful when configuring the 'URL Policy'. If the web interface configuration is invalid, it can lead to the EBX5 application being unusable. If this occurs, use the "rescue mode" by setting `frontEnd.rescueMode.enable=true` in EBX5 main configuration file [p 347], and accessing the following URL in your browser as a built-in administrator user: `http://.../ebx/?onwbpID=iebx-manager-rescue`.

Session configuration

These parameters configure the user session options:

User session default locale	Default session locale.
Session time-out (in seconds)	Maximum duration of user inactivity before the session is considered to be inactive and is terminated. A negative value indicates that the session should never timeout.

Interface configuration

Entry policy

Describes the URL to access the application.

Login URL	If the user is not authenticated, the session is forwarded to this URL.

The entry policy defines an EBX5 login page, replacing the default one.

If defined,

- it replaces an authentication URL that may have been defined using a specific user `Directory`API,
- it is used to build the permalinks in the user interface,
- if the URL is full, that is, starting with `http://` or `https://`, it replaces the URL of the workflow email configuration.

URL policy

Describes the URL and proxy policy. Both dynamic (servlet) and static (resources) URLs can be configured.

HTTP servlet policy	Header content of the servlet HTTP request: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.
HTTPS servlet policy	Header content of the servlet HTTPS request: • if a field is not set, the default value is chosen (in environment configuration), • if a default value is not set, the value in the initial request is used.
HTTP external resource policy	Header content of the external resource URL in HTTP: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.
HTTPS external resource policy	Header content of the external resource URL in HTTPS: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.

Exit policy

Describes how the application is exited.

Normal redirection	Specifies the redirection URL used when exiting the session normally.
Error redirection	Specifies the redirection URL used when exiting the session due to an error.

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, `((127\.0\.0\.1) | (192\.168\.*\.*))` grants access to the local machine and the network IP range `192.168.*.*`.
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 view is applied on a table.
Maximum auto-width for table columns	Defines the maximum width to which a table column can auto-size during table initialization. This is to prevent columns from being too wide, which could occur for very long values, such as URLs. Users will still be able to manually resize columns beyond this value.
Max expanded elements for a hierarchy	Defines the maximum number of elements that can be expanded in a hierarchy when using the action "Expand all". A value less than or equal to '0' disables this parameter.
Default table filter	Defines the default table filter to display in the filters list in tabular views. If modified, users must log out and log in for the new value to take effect.
Display the message box automatically	Defines the message severity threshold for displaying the messages pop-up.
IE compatibility mode	Defines whether or not to compensate for Internet Explorer 8+ displaying EBX5 in compatibility mode. In order to prevent Internet Explorer browsers from using compatibility mode when displaying the repository user interface, the meta-tag *http-equiv="X-UA-Compatible" content="IE=EmulateIE8"* is added to the header of pages. However, in some local environments, this setting may conflict with existing policies, in which case this header must be omitted by setting the parameter to 'No'. The default value is 'Yes'. See Specifying Document Compatibility Modes for more information.
Forms: width of labels	The width of labels in forms.
Forms: width of inputs	The width of form input fields in forms.
Forms: height of text areas	The height of text entry fields in forms.

Forms: aggregated lists	The number of hidden candidate lines to be generated, available to create new instances in the list.
Forms: width of HTML editor	The width of HTML editors in forms.
Forms: height of HTML Editor	The height of HTML editors in forms.
Searchable list selection page size	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 default character encoding to use for CSV file import and export.
CSV : field separator	Specifies the default separator character to use for CSV file import and export.
CSV : list separator	Specifies the default list separator character to use for CSV file import and export.
Import mode	Specifies the default import mode.
Missing XML values as 'null'	If 'Yes', when updating existing records, if a node is missing or empty in the imported file, the value is considered as 'null'. If 'No', the value is not modified.

Colors and themes

Customizes 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.

61.8 **Lineage**

To administrate lineage, three tables are accessible:

- **Authorized profiles**: Profiles must be added to this table to be used for data lineage WSDL generation.
- **History**: Lists the general data lineage WSDLs and their configurations.
- **JMS location**: Lists the JMS URL locations.

61.9 **Event broker**

Overview

EBX5 offers the ability to receive notifications and information related to specific events using the event broker feature. This feature consists in sending notifications related to EBX5 core events to the subscriber according to their chosen topic.

Terminology

Event broker	Notification component for loosely-coupled event handling. Consists of dispatching fired events from EBX5 core to concerned subscribers. The event broker is mainly used for monitoring and statistical purposes.
Topic	Corresponds to the EBX5 event type that contains messages. The number of subscribers registered to a topic is unlimited.
Subscriber	Client implementation in the modules that receive the events related to the subscribed topic(s).

Topics

Repository	Corresponds to operations in the repository, such as: start-up and purge.
Data space and snapshot	Corresponds to operations in the data space and in the snapshot, such as: create, close, reopen, delete, archive export and archive import (only for data space merge).
User session	Corresponds to the operations related to user authentication, such as: login and logout.

Administration

The management console is located under 'Event broker' in the 'Administration' area. It contains three tables: 'Topics', 'Subscribers' and 'Subscriptions'.

All content is read-only, except for the following operations:

- Topic and subscriber can be manually activated or deactivated using dedicated services.
- Subscribers that are no longer registered to the broker can be deleted.

61.10 **Other technical tables**

Several technical tables can be accessed in the 'Administration' area of the EBX5 user interface. These tables are for internal use only and their content should not be edited manually, unless removing obsolete or erroneous data.

Auto-increments	Lists all auto-increment fields in the repository.
Interactions	Lists all interactions in the repository.
Workflows	Lists all data workflows and their tokens in the repository, and the information associated with each, including history.
User preferences	Lists the user preferences of users of the repository.
Views configuration	Allows accessing the Views, Groups of views, User filters and Default views lists defined by users of the repository.

CHAPTER 62

Users and roles directory

This chapter contains the following topics:

62.1 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 another type of enterprise directory.

See also

Configuring the user and roles directory [p 350]

Custom directory [p 393]

62.2 Concepts

In EBX5, a user can be a member of several roles, and a role can be shared by several users. Moreover, a role can be included in another role. The generic term *profile* is used to describe 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. Allows performing general administrative tasks.
Profile.READ_ONLY	Built-in read-only role. A user associated with the read-only role can only view the EBX5 repository, with no rights for performing modifications in the repository.
Profile.OWNER	Dynamic built-in owner role. This role is checked dynamically depending on the current element. It is only activated if the user belongs to the profile defined as owner of the current element.
Profile.EVERYONE	All users belong to this role.

Information related to profiles is primarily defined in the directory.

Attention

Associations between users and the built-in roles *OWNER* and *EVERYONE* are managed automatically by EBX5, and thus must not be modified by the directory.

User permissions are managed separately from the directory. See Permissions [p 303].

See also

profile [p 19]

role [p 20]

user [p 19]

administrator [p 20]

user and roles directory [p 20]

62.3 Default directory

Directory content

The default directory is represented by the data set 'Directory', in the Administration area.

This data set contains tables for users and roles.

Depending on the policies defined, users can modify information related to their own accounts.

Note

It is not possible to delete or duplicate the default directory.

Password recovery procedure

In the default directory, passwords are encrypted (by default with a SHA256-like algorithm), and stored in this state. Consequently, it is impossible to retrieve lost passwords. A new password must be generated and sent to the user.

There are two options for this procedure:

1. A notification email is sent to the administrator, the administrator manually changes the password and sends the new password to the user.
2. A procedure automatically generates a new password and sends it to the user.

By default, the first option is used. To activate the second option, specify the property `ebx.password.remind.auto=true` in the EBX5 main configuration file [p 347].

> **Note**
>
> For security reasons, the password recovery procedure is not available for administrator profiles. If required, use the administrator recovery procedure instead.

Administrator recovery procedure

If all the 'login/password' credentials of the administrators are lost, a special procedure must be followed. A specific directory class redefines an administrator user with login 'admin' and password 'admin'.

To activate this procedure:

- Specify the following property in the EBX5 main configuration file [p 347]:

  ```
  ebx.directory.factory=
  com.orchestranetworks.service.directory.DirectoryDefaultRecoverFactory
  ```

- Start EBX5 and wait until the procedure completes.
- Reset the '`ebx.directory.factory`' property.
- Restart EBX5 and connect using the 'admin' account.

> **Note**
>
> While the '`ebx.directory.factory`' property is set for recovery procedure, authentication of users will be denied.

62.4 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.

See also *`DirectoryFactory`*API

CHAPTER 63 Audit trail

This chapter contains the following topics:

63.1 Overview

XML audit trail is a feature that allows logging updates into XML files. An alternative history feature is also available to record table updates in the relational database; see History [p 275].

Any persistent updates performed in the EBX5 repository is logged to an audit trail XML file. Procedure executions are also logged, even if they do not perform any updates, as procedures are always considered to be transactions. The following information is logged:

- Transaction type, such as data set creation, record modification, record deletion, specific procedure, etc.
- Data space or snapshot on which the transaction is executed.
- Transaction source. If the action was initiated by EBX5, this source is described by the user identity, HTTP session identifier and client IP address. If the action was initiated programmatically, only the user's identity is logged.
- Optional "trackingInfo" value regarding the session
- Transaction date and time (in milliseconds);
- Transaction UUID (conform to the Leach-Salz variant, version 1);
- Error information; if the transaction has failed.
- Details of the updates done. If there are updates and history detail is activated, see next section.

63.2 Update details and disk management

The audit trail is able to describe all updates made into the EBX5 repository, at the finest level. Thus, the XML files can be quite large and the audit trail directory must be carefully supervised. The following should be taken into account:

1. If an archive import is executed in non-interactive mode (without a change set), the audit trail does not detail the updates; it only specifies the archive that has been imported. In this case, if it is important to keep a fine trace of the import-replace, the archive itself must be preserved.
2. If an archive import is executed in interactive mode (with a change set), or if a data space is merged to its parent, the resulting log size will nearly triple the unzipped size of the archive. Furthermore, for consistency concerns, each transaction is logged into a temporary file (in the audit trail directory) before being moved into the main file. Therefore, EBX5 requires *at least six times the unzipped size of the largest archive that may be imported*.
3. In the context of a custom procedure that performs many updates not requiring auditing, it is possible for the developer to disable detailed history using the method `ProcedureContext.setHistoryActivation`API.

See also EBX5 monitoring *[p 375]*

63.3 File organization

All audit trail files are stored in the directory `${ebx.repository.directory}/History`.

"Closed" audit files

Each file is named as follows:

`<yyyy-mm-dd>-part<nn>.xml`

where `<yyyy-mm-dd>` is the file date and `<nn>` is the file index for the current day.

Writing to current audit files

When an audit file is being written, the XML structure implies working in an "open mode". The XML elements of the modifications are added to a text file named:

`<yyyy-mm-dd>-part<nn>Content.txt`

The standard XML format is still available in an XML file that references the text file. This file is named:

`<yyyy-mm-dd>-part<nn>Ref.xml`

These two files are then re-aggregated in a "closed" XML file when the repository has been cleanly shut down, or if 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

This chapter contains the following topics:

64.1 Administrating 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.

64.2 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'.

This chapter contains the following topics:

65.1 Execution of workflows and history

Several tables facilitate the management of the data workflows currently in progress. These tables 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, including those invoked as sub-workflows. A data workflow is a particular execution instance of a workflow model publication. This table provides access to the data context variables for all data workflows. It can be used to access the status of advancement of the data workflow in terms of current variable values, and in case of data workflow suspension, to modify the variable values.

Tokens table

The 'Tokens' table allows managing the progress of data workflows. Each token marks the current step being executed in a running data workflow, as well as the current state of the data workflow.

See also *token* *[p 26]*

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* *[p 26]*

'Waiting workflows' table

The 'Waiting workflows' table contains all the workflows waiting for an event. If needed, a service is available to clean this table: this service deletes all lines associated with a deleted workflow.

See also *wait task* *[p 26]*

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.

65.2 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.

65.3 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.

65.4 Configuration

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

This chapter contains the following topics:

66.1 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.

66.2 Configuration from EBX5

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the Administration area.

- **Schedules**: defines scheduling using "cron expressions".
- **Tasks**: configures tasks, including parametrizing task instances and user profiles for their execution.
- **Scheduled tasks**: current schedule, including task scheduling activation/deactivation.
- **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.

66.3 Cron expression

(An extract of the 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" day-of-week day of the month. For example, the value of "6#3" in the day-of-week field means "the third Friday of the month" (day 6 = Friday and "#3" = the 3rd one in the month). Other examples: "2#1" = the first Monday of the month and "4#5" = the fifth Wednesday of the month. Note that if you specify "#5" and there is not 5 of the given day-of-week in the month, then no firing will occur that month.

Examples

Expression	Meaning
0 0 12 * * ?	Fire at 12pm (noon) every day.
0 15 10 ? * *	Fire at 10:15am every day.
0 15 10 * * ?	Fire at 10:15am every day.
0 15 10 * * ? *	Fire at 10:15am every day.
0 15 10 * * ? 2005	Fire at 10:15am every day during the year 2005.
0 * 14 * * ?	Fire every minute starting at 2pm and ending at 2:59pm, every day.
0 0/5 14 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day.
0 0/5 14,18 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at 6pm and ending at 6:55pm, every day.
0 0-5 14 * * ?	Fire every minute starting at 2pm and ending at 2:05pm, every day.
0 10,44 14 ? 3 WED	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
0 15 10 ? * MON-FRI	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday.
0 15 10 15 * ?	Fire at 10:15am on the 15th day of every month.
0 15 10 L * ?	Fire at 10:15am on the last day of every month.
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month.
0 15 10 ? * 6L 2002-2005	Fire at 10:15am on every last friday of every month during the years 2002, 2003, 2004 and 2005.
0 15 10 ? * 6#3	Fire at 10:15am on the third Friday of every month.
0 0 12 1/5 * ?	Fire at 12pm (noon) every 5 days every month, starting on the first day of the month.
0 11 11 11 11 ?	Fire every November 11th at 11:11am.

> **Note**
>
> Pay attention to the effects of '?' and '*' in the day-of-week and day-of-month fields!
>
> Support for specifying both a day-of-week and a day-of-month value is not complete (you must currently use the '?' character in one of these fields).
>
> Be careful when setting fire times between the hours of the morning when "daylight savings" changes occur in your locale (for US locales, this would typically be the hour before and after 2:00 AM - because the time shift can cause a skip or a repeat depending on whether the time moves back or jumps forward.

66.4 **Task definition**

EBX5 scheduler comes with some predefined tasks.

Custom scheduled tasks can be added by the means of **scheduler** `Package com.orchestranetworks.scheduler`API Java API.

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the Administration area.

66.5 **Task configuration**

A user must be associated with a task definition; this user will be used to generate the **session** `Session`API that will run the task.

> **Note**
>
> The user will not be authenticated, and no password is required. As a consequence, a user with no password set in the directory can only be used to run scheduled tasks.

A custom task can be parametrized 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

This chapter contains the following topics:

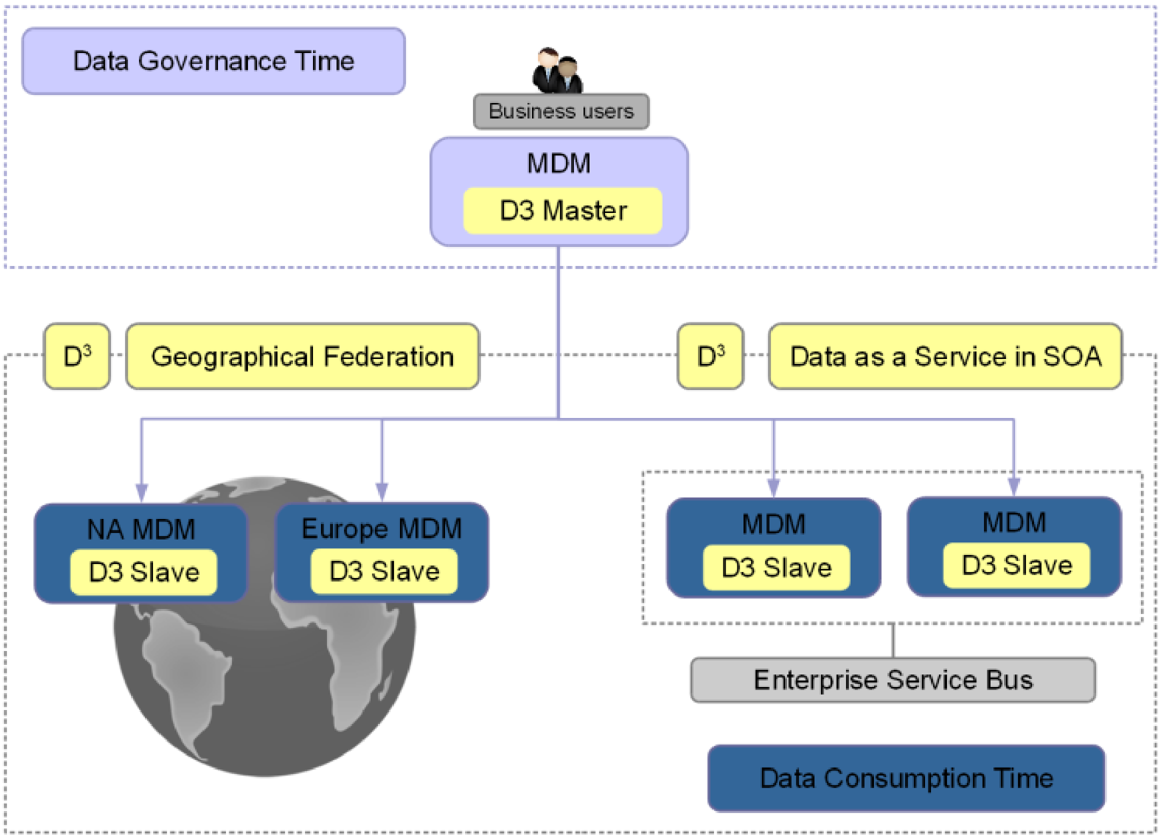
67.1 Overview

EBX5 offers the ability to send data from one EBX5 instance to other instances. Using a broadcast action, it also provides an additional layer of security and control to the other features of EBX5. It is particularly suitable for situations where data governance requires the highest levels of data consistency, approvals and the ability to rollback.

D3 architecture

A typical D3 installation consists of one master instance and multiple slave instances. In the master, a Data Steward declares which data spaces must be broadcast, as well as 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 from which delivery profiles it receives broadcasts.



Involving third-party systems

The features of D3 also allow third-party systems to access 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 broadcast snapshot. This ensures a more controlled and reliable view of the managed data.

Third-party systems can either access data directly through the master node or through a slave node. Thus, a physical architecture consisting of a master node and no slave nodes is possible.

Protocols

If JMS is activated, conversation between a master node and a slave node is based on SOAP over JMS, while archive transfer is based on JMS binary messages.

If JMS is not activated, conversation between a master node and a slave node is based on SOAP over HTTP(S), while binary archive transfer is based on TCP sockets. If HTTPS is used, make sure that the target node connector is correctly configured by enabling SSL with a trusted certificate.

See also *JMS for distributed data delivery (D3)* *[p 340]*

67.2 D3 terminology

broadcast	Send a publication of an official snapshot of data from a master node to slave nodes. The broadcast transparently and transactionally ensures that the data is transferred to the slave nodes.
delivery data space	A delivery data space is a data space that can be broadcast to authenticated and authorized users using a dedicated action. By default, when a data service accesses a delivery data space on any node, it is redirected to the last snapshot that was broadcast. See Data services [p 419].
delivery profile	A delivery profile is a logical name that groups one or more delivery data spaces. Slave nodes subscribe to one or more 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 last broadcast 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 last broadcast snapshots. The synchronization processes are thus independent of one another and replay individual slave nodes are performed for certain broadcast failures.
master node	An instance of EBX5 that can define one or more delivery data spaces, and to which slave nodes can subscribe. A master node can also act as a regular EBX5 server.
slave node	An instance of EBX5 attached to a master node, in order to receive delivery data space broadcasts. Besides update restrictions on delivery data spaces, the slave node acts as a regular EBX5 server.

hub node	An instance of EBX5 acting as both a master node and a slave node. Master delivery data spaces and slave delivery data spaces **must** be disjoint.

67.3 Known limitations

General limitations

- Each slave node must have only one master node.
- 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.
- The compatibility is not assured if at least one slave product version is different from the master.

Broadcast and delivery data space limitations

- Access rights on data spaces are not broadcast, whereas access rights on data sets are.
- Data space information is not broadcast.
- Data spaces defined in relational mode cannot be broadcast.
- If a data space and its parent are broadcast, their parent-child relationship will be lost in the slave nodes.
- 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.
- To guarantee data space consistency between D3 nodes, the data model (embedded or packaged in a module) on which the broadcast contents are based, must be the same between the master node and its slave nodes.
- On a slave delivery data space, if several slave nodes are registered, and if replication is enabled in data models, it will be effective for all nodes. No setting is available to activate/deactivate replication according to D3 nodes.
- Replication on slave nodes does not take part in the distributed transaction: it is automatically triggered after commit.

Administration limitations

Technical data spaces cannot be broadcast, thus the EBX5 default user directory cannot be synchronized using D3.

CHAPTER 68

D3 broadcasts and delivery data spaces

This chapter contains the following topics:

68.1 Broadcast

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 broadcast snapshot and the current 'commit' one 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, if the previous slave commit is not known to the master node, or on demand using the UI service in '[D3] Master Configuration'.

See also *Services on master nodes* *[p 426]*

Performing a broadcast

The broadcast can be performed:

- By the end user, using the action **Broadcast** available in the data space or snapshot (this action is available only if the data space is registered as a delivery data space)
- Using custom Java code that uses `D3NodeAsMaster`API.

Conditions

To be able to broadcast, the following conditions must be fulfilled:

- The authenticated user profile has permission to broadcast.
- The data space or snapshot to be broadcasted has no validation errors.

  Note: Although it is not recommended, it is possible to force a broadcast of a delivery data space that contains validation errors. In order to do this, set the maximum severity threshold allowed in a delivery data space validation report under '[D3] Configuration of master' in the Administration area.
- The D3 master configuration has no validation errors on the following scope: the technical record of the concerned delivery data space and all its dependencies (dependent delivery mappings, delivery profiles and registered slaves).
- The data space or snapshot does not contain any tables in relational mode.
- There is an associated delivery profile.
- If broadcasting a data space, the data space is not locked.
- If broadcasting a snapshot, the snapshot belongs to a data space declared as delivery data space and is not already the current broadcast snapshot (though a rollback to a previously broadcast snapshot is possible).
- The data space or snapshot contains differences compared to the last broadcast snapshot.

Persistence

When a master node shuts down, all waiting or in progress broadcast requests abort, then they will be persisted on a temporary file. On startup, all aborted broadcasts are restarted.

See also *Temporary files* [p 428]

> **Note**
>
> Broadcasts are performed asynchronously. Therefore, no information is displayed in the user interface about the success or failure of a broadcast. Nevertheless, it is possible to monitor the broadcast operations inside '[D3] Master configuration'. See Supervision [p 427].

68.2 Slave registration

Scope and contents

An initialization occurs at the slave level according to the delivery profiles registered in the EBX5 main configuration file of the slave. When the master receives that initialization request, it creates or updates the slave entry, then sends the last broadcast snapshot of all registered delivery data spaces.

> **Note**
>
> If the registered slave repository ID or communication layer already exists, the slave entry in 'Registered slaves' technical table is updated, otherwise a new entry is created.

Performing an initialization

The initialization can be done:

- Automatically on slave node server startup.
- Manually when calling the slave service 'Register slave'.

Conditions

To be able to register, the following conditions must be fulfilled:

- The D3 mode must be 'hub' or 'slave'.
- The master and slave authentication parameters must correspond to the master administrator and slave administrator defined in their respective directories.
- The delivery profiles defined on the slave node must exist in the master configuration.
- All data models contained in the registered data spaces must exist in the slave node. If embedded, the data model names must be the same. If packaged, they must be located at the same module name and the schema path in module must be the same in both the master and slave nodes.
- The D3 master configuration has no validation error on the following scope: the technical record of the registered slave and all its dependencies (dependent delivery profiles, delivery mappings and delivery data spaces).

> **Note**
>
> To set the parameters, see the slave or hub EBX5 properties in Configuring master, hub and slave nodes [p 423].

68.3 **Accessing 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 in the global configuration.

See also *Common parameter 'disableRedirectionToLastBroadcast' [p 215]*

Access restrictions

On the master node, a delivery data space can neither be merged nor closed. Other operations are available depending on permissions, for example, modifying a delivery data space directly, creating a snapshot independent from a broadcast, or creating and merging a child data space.

On the slave node, aside from the broadcast process, no modifications of any kind can be made to a delivery data space, whether by the end user, data services, or a Java program. Furthermore, any data space-related operations, such as merge, close, etc., are forbidden on the slave node.

D3 broadcast Java API

The last broadcast snapshot may change between two calls if a broadcast has taken place in the meantime. If a fully stable view is required for several successive calls, these calls need to specifically refer to the same snapshot.

To get the last broadcast snapshot, see `D3Node.getBroadcastVersion`API.

CHAPTER **69**

D3 administration

This chapter contains the following topics:

69.1 Quick start

This section introduces the configuration of a basic D3 architecture with two EBX5 instances. Before starting, please check that each instance can work fine with its own repository.

> **Note**
>
> Deploy EBX5 on two different web application containers. If both instances are running on the same host, ensure that all communication TCP ports are distinct.

Declare an existing data space on the master node

The objective is to configure and broadcast an existing data space from a *master* node.

This configuration is performed on the entire D3 infrastructure (master [p 414] and slave [p 414] nodes included)

Update `ebx.properties`*master* node configuration file with:

1. Define D3 mode as `master` in key `ebx.d3.mode`.

> **Note**
>
> The *master* node can be started after configuration.

After authenticating as a built-in administrator, navigate inside the administration tab:

1. Prerequisite: Check that node is configured as *master* node (in 'Actions' menu use 'System information' and check 'D3 mode').
2. Open '[D3] Master configuration' administration feature.
3. Add data space to be broadcasted into 'Delivery data spaces' table, and declare the allowed profile.
4. Add delivery profile [p 414] into the 'Delivery profiles' table (it must correspond to a logical name) and declare the delivery mode. Possible values are: cluster mode [p 414] or federation mode [p 414].
5. Map the delivery data space with the delivery profile into 'Delivery mapping' table.

> **Note**
>
> The *master* node is now ready for slave(s) registration on delivery profile.
>
> Check that D3 broadcast menu appears in 'Actions' of the data space or one of its snapshots.

Configure slave node for registration

The objective is to configure and register the *slave* node based on a delivery profile and communications settings.

Update `ebx.properties` slave node configuration file with:

1. Define D3 mode as `slave` in key `ebx.d3.mode`.
2. Define the delivery profile [p 414] set on *master* node in key `ebx.d3.delivery.profiles` (delivery profiles must be separated by a comma and a space).
3. Define *master* node user authentication (must have the built-in administrator profile) for node communications in `ebx.d3.master.username` and `ebx.d3.master.password`.
4. Define HTTP/TCP protocols [p 425] for *master* node communication, by setting a value for the property key `ebx.d3.master.url`

   (for example `http://localhost:8080/ebx-dataservices/connector`).
5. Define *slave* node user authentication (must have the built-in administrator profile) for node communications in `ebx.d3.slave.username` and `ebx.d3.slave.password`.
6. Define HTTP/TCP protocols [p 425] for *slave* node communication, by setting a value for the property key `ebx.d3.slave.url`

   (for example `http://localhost:8090/ebx-dataservices/connector`).

> **Note**
>
> The *slave* node can be started after configuration.

After authenticating as a built-in administrator, navigate inside the administration tab:

1. Prerequisite: Check that node is configured as *slave* node (in 'Actions' menu use 'System information' and check 'D3 mode').
2. Open '[D3] Slave configuration' administration feature.
3. Check the information on screen 'Master information': No field should have value 'N/A'.

> **Note**
>
> Please check that the model is available before broadcasting (from data model assistant, it must be published).
>
> Then *slave* node is now ready for broadcast.

69.2 Configuring D3 nodes

Runtime configuration of master and 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 broadcasted.
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.

> **Note**
>
> The tables above are read-only while some broadcasts are pending or in progress.

Configuring master, hub and slave nodes

This section details how to configure a node in its EBX5 main configuration file.

See also *Overview* *[p 347]*

Master node

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 master, hub and slave
#####################################################################
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=master
```

See also *master node [p 414]*

Hub node

In order to act as a *hub* node (combination of master and slave node 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 master, hub and slave
##################################################################
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=hub

##################################################################
# Configuration dedicated to hub or slave
##################################################################
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

See also *hub node [p 415]*

Slave node

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 master, hub and slave
##################################################################
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=slave

##################################################################
# Configuration dedicated to hub or slave
##################################################################
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

See also *slave node* *[p 414]*

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* *[p 347]*

HTTP(S) and socket TCP protocols

Sample configuration for `ebx.d3.mode=hub` or `ebx.d3.mode=slave` node with HTTP(S) network protocol:

```
##################################################################
# HTTP(S) and TCP socket configuration for D3 hub and slave
##################################################################
# URL to access the data services connector of the master
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: [protocol]://[master_host]:[master_port]/ebx-dataservices/connector
# Where the possible values of 'protocol' are 'http' or 'https'.
ebx.d3.master.url=

# URL to access the data services connector of the slave
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: [protocol]://[slave_host]:[slave_port]/ebx-dataservices/connector
# Where the possible values of 'protocol' are 'http' or 'https'.
ebx.d3.slave.url=

# Minimum port to use to transfer archives on TCP mode.
# Must be a positive integer above zero and below 65535.
# If not set, a random port will be used.
#ebx.d3.slave.socket.range.min=

# Max port to use on TCP mode to transfer archives.
# Must be a positive integer above ebx.d3.slave.socket.range.min and below 65535.
# Mandatory if ebx.d3.slave.socket.range.min is set.
#ebx.d3.slave.socket.range.max=
```

JMS protocol

If JMS is activated, the following properties can be defined in order to enable JMS functionalities for a D3 node.

Sample configuration for all D3 nodes with JMS network protocol:

```
##################################################################
## JMS configuration for D3
##################################################################
# Taken into account only if Data Services JMS is configured properly
##################################################################
# Configuration for master, hub and slave
##################################################################
# Default is false, activate JMS for D3
## If activated, the deployer must ensure that the entries
## 'jms/EBX_D3ReplyQueue', 'jms/EBX_D3ArchiveQueue' and 'jms/EBX_D3CommunicationQueue'
## are bound in the operational environment of the application server.
## On slave or hub mode, the entry 'jms/EBX_D3MasterQueue' must also be bound.
ebx.jms.d3.activate=false

# Change default timeout when use reply queue, default is 10000 milliseconds
#ebx.jms.d3.reply.timeout=10000

# Archive maximum size in KB for the JMS body message. If exceeds, the message
# is transfered into several sequences messages in a same group, where each one does
# not exceed the maximum size defined.
# Must be a positive integer equals to 0 or above 100.
# Default is 0 that corresponds to unbounded.
#ebx.jms.d3.archiveMaxSizeInKB=

##################################################################
# Configuration dedicated to hub or slave
```

```
##################################################################
# Master repository ID, used to set a message filter for the concerned master when sending JMS message
# Mandatory property if ebx.jms.d3.activate=true and if ebx.d3.mode=hub or ebx.d3.mode=slave
#ebx.jms.d3.master.repositoryId=
```

See also *JMS for distributed data delivery (D3)* [p 340]

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:

Relaunch replays	Immediately relaunch all replays for waiting federation deliveries.
Delete slave delivery data space...	Delete the delivery data space on chosen slave nodes and/or unregister it from the configuration of the D3 master node. To access the service, select a delivery data space from the 'Delivery data spaces' table on the master node, then launched the wizard.
Fully resynchronize	Broadcast the full content of the last broadcast snapshot to the registered slaves.
Subscribe a slave node	Subscribe a set of selected slave nodes.
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 node.

Note

The master services above are hidden while some broadcasts are pending or in progress.

Services on slave nodes

Services are available in the Administration area under *[D3] Configuration of slave* to manage its subscription to the master node and perform other actions:

Register slave	Re-subscribes the slave node to the master node if it has been unregistered.
Unregister slave	Disconnects the slave node from the master node.
Close and delete snapshots	Clean up a slave delivery data space. To access the service, select a delivery data space from the 'Delivery data spaces' table on the slave node, then follow the wizard to close and delete snapshots based on their creation dates. **Note:** The last broadcast snapshot is automatically excluded from the selection.

69.3 Supervision

The last broadcast snapshot is highlighted in the snapshot table of the data space, it is represented by an icon displayed in the first column.

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:

Check slave node information	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 node 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`*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 [p 352]*

Temporary files

Some temporary files, such as exchanged archives, SOAP messages, broadcast queue, (...), are created and written to the EBX5 temporary directory. This location is defined in EBX5 main configuration file:

```
##################################################
## Directories for temporary resources.
##################################################
# When set, allows specifying a directory for temporary files different from java.io.tmpdir.
# Default value is java.io.tmpdir
ebx.temp.directory = ${java.io.tmpdir}

# Allows specifying the directory containing temporary files for cache.
# If unset, the used directory is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.cache.directory = ${ebx.temp.directory}/ebx.platform

# When set, allows specifying the directory containing temporary files for import.
# If unset, the used directory is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.import.directory = ${ebx.temp.directory}/ebx.platform
```

Developer Guide

CHAPTER 70

Notes to developers

This chapter contains the following topics:

70.1 Terminology in version 5

In version 5, EBX5 introduced a new vocabulary for users. These terminology changes, however, do not impact the API; Java classes, data services WSDLs and EBX5 components still use version 4 terminology.

See also *Glossary* *[p 19]*

The following table summarizes the mappings between version 5 terminology and previous terminology:

Term in EBX5	Term 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. Data models contribute to EBX5's ability to guarantee the highest level of data consistency and facilitate data management.

Specifically, the data model is a document that conforms to the XML Schema standard (W3C recommendation). Its main features are as follows:

- A rich library of well-defined simple data types [p 441], such as integer, boolean, decimal, date, time;
- The ability to define additional simple types [p 443] and complex types [p 443];
- The ability to define simple lists of items, called aggregated lists [p 447];
- Validation constraints [p 469] (facets), for example, enumerations, uniqueness constraints, minimum/maximum boundaries.

EBX5 also uses the extensibility features of XML Schema for other useful information, such as:

- Predefined types [p 443], for example, locale, resource, html;
- Definition of tables [p 451] and foreign key constraints [p 455];
- Mapping data in EBX5 to Java beans;
- Advanced validation constraints [p 469] (extended facets), such as dynamic enumerations;
- Extensive presentation information [p 487], such as labels, descriptions, and error messages.

> **Note**
>
> EBX5 supports a subset of the W3C recommendation, as some features are not relevant to Master Data Management.

This chapter contains the following topics:

71.1 Model Editor

The data model can be defined using an XML Schema editor or through the data model assistant. The data model assistant has the advantage of being integrated into the EBX5 user interface, abstracting the verbose underlying XML.

71.2 References

For an introduction to XML Schema, see the W3Schools XML Schema Tutorial.

See also

XML Schema Part 0: Primer

XML Schema Part 1: Structures

XML Schema Part 2: Datatypes

71.3 Relationship between data sets and data models

Each root data set is associated with a single data model. At data space creation, an associated data model is selected, on which to base the data set.

See also *Creating a data set* *[p 93]*

71.4 Pre-requisite for XML Schemas

In order for an XML Schema to be accepted by EBX5, it must include a global element declaration that includes the 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>
```

71.5 Conventions

By convention, namespaces are always defined as follows:

Prefix	Namespace
`xs:`	`http://www.w3.org/2001/XMLSchema`
`osd:`	`urn:ebx-schemas:common_1.0`
`fmt:`	`urn:ebx-schemas:format_1.0`

71.6 **Schemas with reserved names**

Several data models in EBX5 have reserved names.

All references to other data models (using the attribute `schemaLocation` for an import, include or redefine) that end with one of the following strings are reserved:

- common_1.0.xsd
- org_1.0.xsd
- coreModel_1.0.xsd
- session_1.0.xsd

These XSD files correspond to the schemas provided for the module `ebx-root-1.0`, at the path `/WEB-INF/ebx/schemas`. The attribute `schemaLocation` can reference the files at this location or a copy, if the file names are the same. This is useful if you want to avoid a module dependency on `ebx-root-1.0`.

For security reasons, EBX5 uses an internal definition for these schemas to prevent any modification.

CHAPTER 72

Packaging EBX5 modules

An EBX5 module allows packaging a data model along with its resources, such as included XML Schema Documents and Java classes.

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 deployment.

This chapter contains the following topics:

72.1 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 [p 438].

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, which are accessible by public URL. This directory is localized and structured by resource type (HTML, images, JavaScript files, stylesheets). External resources in this directory can be referenced by data models using the type `osd:resource`.

72.2 Declaration

A module is declared using the document `/WEB-INF/ebx/module.xml`. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:ebx-schemas:module_2.3"
 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>
```

See the associated schema for documentation about each property. The main properties are as follows:

Element	Description	Required
`name`	Defines the unique identifier of the module in the server instance. The module name usually corresponds to the name of the web application (the name of its directory).	Yes.
`publicPath`	Defines a path other than the module's name identifying the web application in public URLs. This path is added to the URL of external resources of the module, when computing absolute URLs. If this field is not defined, the public path is the module's `name`, defined above.	No.
`locales`	Defines the locales supported by the data models in the module. This list must contain all the locales that are included in the data models within the module, and that are exposed 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 UI services. See Declaration and configuration [p 178] of UI services.	No.
`beans`	Declares reusable Java bean components. See the **workflow package** `Package com.orchestranetworks.workflow`API in the Java API.	No.
`ajaxComponents`	Declares Ajax components. See **Declaring an Ajax component in a module** `UIAjaxComponent`API in the Java API.	No.

72.3 Registration

In order to be identifiable to 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 the code example below.
2. Make a standard declaration of this servlet in the deployment descriptor `/WEB-INF/web.xml`.
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>` .

Deployment descriptor example

The follow is an example of a Java EE deployment descriptor (`/WEB-INF/web.xml`):

```
<?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>
```

Note

- If Java classes are located in the web application (in `/WEB-INF/classes` or `/WEB-INF/lib`) and these classes are used as data model resources in the module, the specific servlet class must also be located in the web application. This is because the servlet class is internally used as a hook to the application's class-loader.
- It is recommended for the servlet to also implement the method `destroy()`, otherwise stopping the web application will not work.

  Once the method `ModulesRegister.unregisterWebApp()` has been executed, the data models and associated data sets become unavailable.

  EBX5 supports the hot-deployment and hot-redeployment operations implemented by Java EE application servers: deployment-start, stop-restart, stop-redeployment-restart, stop of a web application. However, these operations remain sensitive, particularly concerning class-loading and potential complex dependencies, consequently these operations must be carefully monitored.
- All module registrations and unregistrations are logged to the `log.kernel` category.
- If an exception occurs while loading a module, the cause is written to the application server log.

Registration example

Java code example of a servlet that registers/unregisters the module in 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* *[p 359]*

CHAPTER **73**

Data types

This chapter details the data types supported by EBX5.

See also *Tables and relationships* *[p 451]*

This chapter contains the following topics:

73.1 XML Schema built-in simple types

The table below lists all the simple types defined in XML Schema that are supported by EBX5, along with their corresponding Java types.

XML Schema type	Java class	Notes
`xs:string`	`java.lang.String`	
`xs:boolean`	`java.lang.Boolean`	
`xs:decimal`	`java.math.BigDecimal`	Converted to a `double` when displayed in the user interface. Loss of precision and/or rounding problems can occur if the `BigDecimal` exceeds 15 digits. Moreover, the inaccurately displayed value will replace the correct one if the form with the displayed value is submitted.
`xs:dateTime`	`java.util.Date`	
`xs:time`	`java.util.Date`	The date portion of the returned `Date` is always set to '1970/01/01'.
`xs:date`	`java.util.Date`	The time portion of the returned `Date` is always the beginning of the day, that is, '00:00:00'.
`xs:anyURI`	`java.net.URI`	
`xs:Name` (`xs:string` restriction)	`java.lang.String`	
`xs:int` (`xs:decimal` restriction)	`java.lang.Integer`	
`xs:integer` (`xs:decimal` restriction)	`java.lang.Integer`	This mapping does not comply with the XML Schema recommendation. Although the XML Schema specification states that `xs:integer` has no value space limitations, this value space is, in fact, restricted by the Java specifications of the `java.lang.Integer` object.

The mapping between XML Schema types and Java types are detailed in the section Mapping of data types [p 503].

73.2 XML Schema named simple types

Named simple types can be defined when designing a data model for redefining an existing built-in simple type. A *named simple type* can be reused in the data model.

Restrictions:

- In the data model, only the element `restriction` is allowed in a named simple type, and even then only derivation by restriction is supported. Notably, the elements `list` and `union` are not supported.
- Facet definition is not cumulative. That is, if an element and its named type both define the same kind of facet then the facet defined in the type is overridden by the local facet definition. However, this restriction does not apply to programmatic facets defined by the element `osd:constraint`. For `osd:constraint`, if an element and its named type both define a programmatic facet with different Java classes, the definition of these facets will be cumulative. Contrary to the XML Schema Specification, 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 not possible to define different types of enumerations on both an element and its named type. For instance, you cannot specify a static enumeration in an element and a dynamic enumeration in its named type.
- It is not possible to simultaneously define a pattern facet in both an element and its named type.

73.3 **XML Schema complex types**

Complex types can be defined when designing a data model. A *named complex type* can be reused in the data model.

Restrictions:

- In the data model, only the element `sequence` is allowed. Notably, attribute definition is not supported.
- Type extensions are not supported in the current version of EBX5.

73.4 **Extended simple types defined by EBX5**

EBX5 provides pre-defined simple data types:

XML Schema type	**Java class**
`osd:text` (`xs:string` restriction)	`java.lang.String`
`osd:html` (`xs:string` restriction)	`java.lang.String`
`osd:email` (`xs:string` restriction)	`java.lang.String`
`osd:password` (`xs:string` restriction)	`java.lang.String`
`osd:resource` (`xs:anyURI` restriction)	internal class
`osd:locale` (`xs:string` restriction)	`java.util.Locale`

The above types are defined by the internal schema `common-1.0.xsd`. They are defined as follows:

`osd:text`	This type represents textual information. For a basic `xs:string`, its default user interface in 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 formatting. A WYSIWYG editor is provided in EBX5.
	`<xs:simpleType name="html">`
	` <xs:restriction base="xs:string" />`
	`</xs:simpleType>`

`osd:email`	This represents an email address as specified by the RFC822 standard.
	`<xs:simpleType name="email">`
	` <xs:restriction base="xs:string" />`
	`</xs:simpleType>`

`osd:password`	This represents an encrypted password. A specific editor is provided in EBX5.
	`<xs:element name="password" type="osd:password" />`
	The default editor performs an encryption using the SHA-256 algorithm. This encryption function is also available from a Java client using the method `DirectoryDefault.encryptString`API.
	It is also possible for the default editor to use a different encryption mechanism by specifying a class that implements the interface `Encryption`API.
	`<xs:element name="password" type="osd:password">`
	` <xs:annotation>`
	`  <xs:appinfo>`
	`   <osd:uiBean class="com.orchestranetworks.ui.UIPassword">`
	`    <encryptionClass>package.EncryptionClassName</encryptionClass>`
	`   </osd:uiBean>`
	`  </xs:appinfo>`
	` </xs:annotation>`
	`</xs:element>`

`osd:resource`	This represents a resource in a module for EBX5. A resource is a file of type image, HTML, CSS or JavaScript, stored in a module within EBX5. It requires the definition of the facet FacetOResource [p 474].
	`<xs:simpleType name="resource">`
	` <xs:restriction base="xs:anyURI" />`
	`</xs:simpleType>`

`osd:locale`	This represents a geographical, political or cultural location. The locale type is translated into Java by the class `java.util.Locale`.

```
<xs:simpleType name="locale">
 <xs:restriction base="xs:string">
  <xs:enumeration value="ar" osd:label="Arabic" />
  <xs:enumeration value="ar_AE" osd:label="Arabic (United Arab
 Emirates)" />
  <xs:enumeration value="ar_BH" osd:label="Arabic (Bahrain)" />
  <xs:enumeration value="ar_DZ" osd:label="Arabic (Algeria)" />
  <xs:enumeration value="ar_EG" osd:label="Arabic (Egypt)" />
  <xs:enumeration value="ar_IQ" osd:label="Arabic (Iraq)" />
   ...
  <xs:enumeration value="vi_VN" osd:label="Vietnamese (Vietnam)" /
>
  <xs:enumeration value="zh" osd:label="Chinese" />
  <xs:enumeration value="zh_CN" osd:label="Chinese (China)" />
  <xs:enumeration value="zh_HK" osd:label="Chinese (Hong Kong)" />
  <xs:enumeration value="zh_TW" osd:label="Chinese (Taiwan)" />
 </xs:restriction>
</xs:simpleType>
```

73.5 Complex types defined by EBX5

EBX5 provides pre-defined complex data types:

XML Schema type	Description
`osd:UDA`	**User Defined Attribute:** This type allows any user, according to their access rights, to define a value associated with an attribute defined in a dictionary called a UDA Catalog.
`osd:UDACatalog`	**Catalog of User Defined Attributes:** This type consists of a table in which attributes can be specified. This catalog is used by all `osd:UDA` elements declared in the same data model.

`osd:UDA`

A User Defined Attribute (UDA) supports both the `minOccurs` and `maxOccurs` attributes, as well as the attribute `osd:UDACatalogPath`, which specifies the path of the corresponding catalog.

```
<xs:element name="firstUDA" type="osd:UDA" minOccurs="0"
 maxOccurs="unbounded" osd:UDACatalogPath="//insuranceCatalog" />
<xs:element name="secondUDA" type="osd:UDA" minOccurs="1"
 maxOccurs="1"
 osd:UDACatalogPath="/root/userCatalog" />
<xs:element name="thirdUDA" type="osd:UDA" minOccurs="0"
 maxOccurs="1"
 osd:UDACatalogPath="//userCatalog" />
```

In the manager, when working with a UDA, the editor will adapt itself to the type of the selected attribute.

`osd:UDACatalog`

Internally, a catalog is represented as a table. The parameters `minOccurs` and `maxOccurs` must be specified.

Several catalogs can be defined in the same data model.

```
<xs:element name="insuranceCatalog" type="osd:UDACatalog"
 minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:documentation xml:lang="en-US">Insurance Catalog.</
xs:documentation>
  <xs:documentation xml:lang="fr-FR">Catalog assurance.</
xs:documentation>
 </xs:annotation>
</xs:element>
<xs:element name="userCatalog" type="osd:UDACatalog" minOccurs="0"
 maxOccurs="unbounded">
 <xs:annotation>
  <xs:documentation xml:lang="en-US">User catalog.</
xs:documentation>
  <xs:documentation xml:lang="fr-FR">Catalogue utilisateur.</
xs:documentation>
 </xs:annotation>
</xs:element>
```

Only the following types are available for creating new attributes:

- xs:string
- xs:boolean
- xs:decimal

- xs:dateTime
- xs:time
- xs:date
- xs:anyURI
- xs:Name
- xs:int
- osd:html
- osd:email
- osd:password
- osd:locale
- osd:text

Restrictions on User Defined Attributes and Catalogs

The following features are unsupported on UDA elements:

- Facets
- Functions using the `osd:function` property
- UI bean editors using the `osd:uiBean` property
- The `osd:checkNullInput` property
- History features
- Replication
- Inheritance features, using the `osd:inheritance` property

As UDA catalogs are internally considered to be tables, the restrictions that apply to tables also exist for `UDACatalog` elements.

73.6 **Aggregated lists**

In XML Schema, the maximum number of times an element can occur is determined by the value of the `maxOccurs` attribute in its declaration. If this value is strictly greater than 1 or is `unbounded`, the data can have multiple occurrences. If no `osd:table` declaration is included, this element is called an *aggregated list*. In Java, it is then represented as an instance of class `java.util.List`.

The following is an example of an aggregated list that defines the pricing of a loan product, depending on the amount borrowed.

```
<xs:element name="pricing" minOccurs="0" maxOccurs="unbounded"
 osd:access="RW">
 <xs:annotation>
  <xs:documentation>
   <osd:label>Pricing</osd:label>
   <osd:description>Pricing grid </osd:description>
  </xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element name="amount" type="xs:int">
    <xs:annotation>
     <xs:documentation>
      <osd:label>Amount borrowed</osd:label>
     </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="monthly" type="xs:int">
    <xs:annotation>
     <xs:documentation>
      <osd:label>Monthly payment </osd:label>
     </xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="cost" type="xs:int">
    <xs:annotation>
     <xs:documentation>
      <osd:label>Cost</osd:label>
     </xs:documentation>
    </xs:annotation>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Aggregated lists have a dedicated editor in EBX5. This editor allows you to add occurrences or to delete occurrences.

Attention

The addition of an `osd:table` declaration to an element with `maxOccurs > 1` is a very important consideration that must be taken into account during the design process. An aggregated list is severely limited with respect to the many features that are supported by tables. Some features unsupported on aggregated lists that are supported on tables are:

- Performance and memory optimization;
- Lookups, filters and searches;
- Sorting, view and display in hierarchies;
- Identity constraints (primary keys and uniqueness constraints);
- Detailed permissions for creation, modification, delete and particular permissions at record level;
- Detailed comparison and merge.

Thus, *aggregated lists should be used only for small volumes of simple data (one or two dozen occurrences), with no advanced requirements*. For larger volumes of data or more advanced functionalities, it is strongly advised to use an `osd:table` declarations.

For more information on table declarations, see Tables and relationships [p 451].

73.7 Including external data models

Including another data model in your current model allows you to use the reusable types that are defined in that data model. You can thus use the inclusion of external data models to share data types between multiple XML Schema Documents.

To include another XML Schema Document in your model, thereby including the data types that it defines, specify the 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 or 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 defined by EBX5. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
 <xs:include schemaLocation="urn:ebx:publication:myPublication"/>
 ...
</xs:schema>
```

To include a data model packaged in a module, specify the specific URN defined by EBX5. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
 <xs:include schemaLocation="urn:ebx:module:aModuleName:/WEB-INF/ebx/schema/myDataModel.xsd"/>
 ...
</xs:schema>
```

See `SchemaLocation`API for more information about specific URNs supported by EBX5.

Note

If the packaged data model uses Java resources, the class loader of the module containing the data model will be used at runtime for resolving these resources.

CHAPTER **74**

Tables and relationships

This chapter contains the following topics:

74.1 Tables

Overview

EBX5 supports the features of relational database tables, including the handling of large volumes of records, and identification by primary key.

Tables provide many benefits that are not offered by aggregated lists [p 447]. Beyond relational capabilities, some features that tables provide are:

- filters and searches;
- sorting, views and hierarchies;
- identity constraints: primary keys, foreign keys [p 455] and uniqueness constraints [p 471];
- specific permissions for creation, modification, and deletion;
- dynamic and contextual permissions at the individual record level;
- detailed comparison and merge;
- ability to have inheritance at the record level (see data set inheritance [p 298]);
- performance and memory optimization.

See also

Foreign keys [p 455]

Associations [p 458]

Working with existing data sets [p 105]

Simple tabular views [p 98]

Hierarchical views [p 98]

History *[p 275]*

Declaration

A table element, which is an element with *maxOccurs > 1*, is declared by adding the following annotation:

```
<xs:annotation>
 <xs:appinfo>
  <osd:table>
   <primaryKeys>/pathToField1 /pathToField...n</primaryKeys>
  </osd:table>
 </xs:appinfo>
</xs:annotation>
```

Common properties

Element	Description	Required
`primaryKeys`	Specifies the primary key fields of the table. Each field of the primary key must be denoted by its absolute XPath notation that starts just under the root element of the table. If there are multiple fields in the primary key, the list is delimited by whitespace. **Note:** Whitespaces in primary keys of type `xs:string` are handled differently. See Whitespace handling for primary keys of type string [p 481].	Yes.

`defaultLabel`	Defines the end-user display of records. Multiple variants can be specified:
	• A static non-localized expression is defined using the `defaultLabel` element, for example:
	`<defaultLabel>Product: ${./productCode}</defaultLabel>`
	• Static localized expressions are specified using the `defaultLabel` element with the attribute `xml:lang`, for example:
	`<defaultLabel xml:lang="fr-FR">Produit : ${./productCode}</defaultLabel>`
	`<defaultLabel xml:lang="en-US">Product: ${./productCode}</defaultLabel>`
	• A JavaBean that implements the interface `UILabelRenderer`API and/or the interface `UILabelRendererForHierarchy`API. The JavaBean is specified by means of the attribute `osd:class`, for example:
	`<defaultLabel osd:class="com.wombat.MyLabel"></defaultLabel>`
	Note: The priority of the tags when displaying the user interface is the following:
	1. `defaultLabel` tags with a JavaBean (but it is not allowed to define several renderers of the same type);
	2. `defaultLabel` tags with a static localized expression using the `xml:lang` attribute;
	3. `defaultLabel` tag with a static non-localized expression.
	Attention: Access rights defined on associated data sets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
		No.

`index`	Specifies an index for speeding up requests that match this index (see performances [p 317]). The attribute `name` is mandatory. Each field of the index must be denoted by its absolute XPath notation, which starts just under the root element of the table. If there are multiple fields in the index, the list is delimited by whitespace. **Note:** • Indexing only concerns semantic and relational tables. History and replica tables are not affected. • It is possible to define multiple indexes on a table. • It is not possible to define two indexes with the same name.	No.

Element	Description	Required
	• It is not possible to declare two indexes containing the same exact fields. • An indexed field must be terminal. • An indexed field cannot be a list or be under a list. • A field declared as an *inherited field* cannot be indexed. • A field declared as a function cannot be indexed. For performance purposes, the following nodes are automatically indexed: • Primary keys nodes; See primary keys [p 451]. • Nodes defining a foreign key constraint. See foreign key constraint [p 455]. • Nodes declared as being unique. See uniqueness constraint [p 471]. • Auto-incremented nodes. See auto-incremented values [p 484].	

Example

Below is an example of a product catalog:

```
<xs:element name="Products" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
   <xs:documentation>
    <osd:label>Product Table </osd:label>
    <osd:description>List of products in Catalog </osd:description>
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
  <xs:annotation>
   <xs:appinfo>
    <osd:table>
     <primaryKeys>./productRange /productCode</primaryKeys>
     <index name="indexProductCode">/productCode</index>
    </osd:table>
   </xs:appinfo>
  </xs:annotation>
   <xs:sequence>
    <xs:element name="productRange" type="xs:string"/><!-- key -->
    <xs:element name="productCode" type="xs:string"/><!-- key -->
    <xs:element name="productLabel" type="xs:string"/>
    <xs:element name="productDescription" type="xs:string"/>
    <xs:element name="productWeight" type="xs:int"/>
    <xs:element name="productType" type="xs:string"/>
    <xs:element name="productCreationDate" type="xs:date"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Catalogs" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
   <xs:documentation>
    <osd:label>Catalog Table</osd:label>
    <osd:description>List of catalogs</osd:description>
   </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:annotation>
    <xs:appinfo>
     <osd:table>
      <primaryKeys>/catalogId</primaryKeys>
     </osd:table>
    </xs:appinfo>
   </xs:annotation>
   <xs:sequence>
    <xs:element name="catalogId" type="xs:string"/><!-- key -->
    <xs:element name="catalogLabel" type="xs:string"/>
    <xs:element name="catalogDescription" type="xs:string"/>
    <xs:element name="catalogType" type="xs:string"/>
    <xs:element name="catalogPublicationDate" type="xs:date"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

Properties related to data set inheritance

The following properties are only valid in the context of data set inheritance:

Element	Description	Required
`onDelete-deleteOccultingChildren`	Specifies whether, upon record deletion, child records in occulting mode are also to be deleted. Valid values are: `never` or `always`.	No, default is `never`.
`mayCreateRoot`	Specifies whether root record creation is allowed. The expression must follow the syntax below. See definition modes [p 298].	No, default is `always`.
`mayCreateOverwriting`	Specifies whether records are allowed to be overwritten in child data sets. The expression must follow the syntax below. See definition modes [p 298].	No, default is "always".
`mayCreateOcculting`	Specifies whether records are allowed to be occulted in child data sets. The expression must follow the syntax below. See definition modes [p 298].	No, default is "always".
`mayDuplicate`	Specifies whether record duplication is allowed. The expression must follow the syntax below.	No, default is "always".
`mayDelete`	Specifies whether record deletion is allowed. The expression must follow the syntax below.	No, default is "always".

The `may...` expressions specify when the action is possible, though the ultimate availability of the action also depends user access rights. The expressions have the following syntax:

```
expression ::= always | never | <condition>*
condition ::= [root:yes | root:no]
"always": the operation is "always" possible (but user rights may restrict this).
"never": the operation is never possible.
"root:yes": the operation is possible if the record is in a root instance.
"root:no": the operation is not possible if the record is in a root instance.
```

If the record does not define any specific conditions, the default is used.

See also *Data set inheritance [p 297]*

74.2 **Foreign keys**

Declaration

A reference to a table [p 451] 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 using 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' (data space) is defined. Otherwise, default is current data set.
`branch`	Reference of the data space that contains the container data set.	No, default is current data space or snapshot.

`display`	Custom display for presenting the selected foreign key in the current record and the sorted list of possible keys. Two variants can be specified, either pattern-based expressions, or a JavaBean if the needs are very specific:
	• Static expressions are specified using the `display` and `pattern` elements. These static expressions can be localized using the additional attribute `xml:lang` on the `pattern` element, for example:
	`<display>`
	`<pattern>Product : ${./productCode}</pattern>`
	`<pattern xml:lang="fr-FR">Produit : ${./productCode}</pattern>`
	`<pattern xml:lang="en-US">Product: ${./productCode}</pattern>`
	`</display>`
	• A JavaBean that implements the interface `TableRefDisplay`API. It is specified using the attribute `osd:class`. For example:
	`<display osd:class="com.wombat.MyLabel"></display>`
	It is not possible to define both variants on the same foreign key element.
	Attention: Access rights defined on associated data sets are not applied when displaying record labels. Fields that are usually hidden due to access rights restrictions will be displayed in labels.
		No, if the `display` property is not specified, the table's record rendering [p 453] is used.

`filter`	Specifies an additional constraint that filters the records of the target table. Two types of filters are available: • An XPath filter is an XPath predicate in the target table context. It is specified using the `predicate` element. For example: `<filter><predicate>type = ${../refType}</predicate></filter>` A localized validation message can be specified using the element `validationMessage`, which will be displayed to the end-user at validation time if a record is not accepted by the filter. A specific severity level can be defined in a nested `severity` element. The default severity is 'error'. Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute.	No.

Element	Description	Required
	To specify a default message for unsupported locales, define a `message` element with no `xml:lang` attribute. • A programmatic filter is a JavaBean that implements the interface `TableRefFilter`API. It is specified using the attribute `osd:class`. For example: `<filter osd:class="com.wombat.MyFilter"></filter>` Additional validation messages can be specified during the setup of the programmatic filter using the dedicated methods of the interface `TableRefFilterContext`API. **Note:** The attributes `osd:class` and the property `predicate` cannot be set simultaneously.	
`validation`	Specifies localized validation messages for the `osd:tableRef` and error management policy. A specific severity level can be defined in a nested `severity` element. The default severity is 'error'. An error management policy can be defined in a nested `blocksCommit` element. The error management policy that blocks all operations does not apply to filters. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected, and a validation error will be reported. Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a `message` element with no `xml:lang` attribute.	No.

Attention

You can create a data set which has a foreign key to a container that does not exist in the repository. However, the content of this data set will not be available until the container is created. 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 for fields that are not contained in 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.

Example

The example below specifies a foreign key in the 'Products' table to a record of the 'Catalogs' table.

```
<xs:element name="catalog_ref" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:tableRef>
     <tablePath>/root/Catalogs</tablePath>
     <display>
       <pattern xml:lang="en-US">Catalog: ${./catalogId}</pattern>
       <pattern xml:lang="fr-FR">Catalogue : ${./catalogId}</pattern>
     </display>
      <validation>
       <severity>error</severity>
       <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
       <message>A default error message</message>
```

```
        <message xml:lang="en-US">A localized error message</message>
        <message xml:lang="fr-FR">Un message d'erreur localisé</message>
      </validation>
     </osd:tableRef>
    </osd:otherFacets>
   </xs:appinfo>
  </xs:annotation>
</xs:element>
```

See also

Table definition [p 451]

Primary key syntax `PrimaryKey`API

Extraction of foreign keys (XPath predicate syntax) [p 257]

Associations [p 458]

View for advanced selection [p 496]

*`SchemaNode.getFacetOnTableReference`*API

*`SchemaFacetTableRef`*API

74.3 Associations

Overview

An association provides an abstraction over an existing relationship in the data model, and allows an easy model-driven integration of *associated objects* in the user interface and in data services.

Two main types of associations are supported:

- *'By foreign key'* specifies the inverse relationship of an existing foreign key field [p 455].
- *'Over a link table'* specifies a relationship based on an intermediate link table (such tables are often called "join tables"). This link table has to define two foreign keys, one referring to the 'source' table (the table holding the association element) and another one referring to the 'target' table.

For an association it is also possible to:

- Filter associated objects by specifying an additional XPath filter.
- Configure a tabular view to define the fields that must be displayed in the associated table.
- Define how associated objects are to be rendered in forms.
- Hide/show associated objects in data service 'select' operation. See Hiding a field in Data Services [p 496].
- Specify the minimum and maximum numbers of associated objects that are required.
- Add validation constraints using XPath predicates for restricting associated objects.

See also

*`SchemaNode.getAssociationLink`*API

*`SchemaNode.isAssociationNode`*API

*`AssociationLink`*API

Declaration

Associations are defined in the data model using the XML Schema element `osd:association` under `xs:annotation/appInfo`.

Restrictions:

- An association must be a simple element of type *xs:string*.
- An association can only be defined inside a table.

> **Note**
>
> The "official" cardinality constraints (`minOccurs="0" maxOccurs="0"`) are required because, from an instance of XML Schema, the corresponding node is absent. In other words, an association has no value and is considered as a "virtual" element as far as XML and XML Schema is concerned.

The table below presents the elements that can be defined under `xs:annotation/appInfo/osd:association`.

Element	Description	Required
`tableRefInverse`	Defines the properties of an association that is the inverse relationship of a *foreign key*. Element `fieldToSource` defines the foreign key that refers to the source table of the association. Element `fieldToSource` is mandatory and must specify a foreign key field that refers to the table containing the association.	Yes if the association is the inverse relationship of a *foreign key*, otherwise no.
`linkTable`	Defines the properties of an association over a link table. Element `table` specifies the link table used by the association. Element `table` is mandatory and must refer to an existing table. **Important:** In order to be used by an association, a link table must define a primary key that is composed of auto-incremented fields or/and the foreign key to the source or target table of the association. Element `fieldToSource` defines the foreign key that refers to the source table of the association. Element `fieldToSource` is mandatory and must specify a foreign key field that refers to the table containing the association. Element `fieldToTarget` defines the foreign key that refers to the target table of the association. Element `fieldToTarget` is mandatory and must specify a foreign key field.	Yes if the association is over a link table, otherwise no.
`xpathFilter`	Defines an XPath predicate to filter associated objects. It is only possible to use fields from the source and the target table when defining an XPath filter. That is, if it is an association over a link table, it is not possible to use fields of the link table in the XPath filter.	No.

It is possible to refer to another data set. For that, the following properties must be defined either under element `tableRefInverse` or `linkTable` depending on the type of the association:

Element	**Description**	**Required**
`schemaLocation`	Defines the data model containing the fields used by the association. The data model is defined using a specific URN that allows referring to embedded data models and data models packaged in modules. See `SchemaLocation`API for more information about specific URNs supported by EBX5.	Yes.
`dataSet`	Defines the data set used by the association. This data set must use the data model specified by the element `schemaLocation`.	Yes.
`dataSpace`	Defines the data space containing the data set used by the association.	No.

Important: When creating a data set, you can create a data set that defines an association to a container that does not yet exist in the repository. However, the content of this data set will not be available immediately upon creation. After the absent container is created, a data model refresh is required in order 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.

User interface integration

It is possible to define how associated objects are to be rendered in forms, using the element `osd:defaultView/displayMode` under `xs:annotation/appinfo`.

Possible values are:

- `inline`, specifies that associated records are to be rendered in the form at the same position of the association in the data model.
- `tab`, specifies that associated records are to be rendered in a specific tab.

By default, associated records are rendered `inline` if this property is not defined.

The following example specifies that associated objects are to be rendered `inline` in the form:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
  <osd:association>
   <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
  </osd:association>
  <osd:defaultView>
   <displayMode>inline</displayMode>
  </osd:defaultView>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

The following example specifies that associated objects are to be rendered in a specific tab:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
  <osd:association>
   <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
  </osd:association>
  <osd:defaultView>
   <displayMode>tab</displayMode>
  </osd:defaultView>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

Customize view of associated objects

A specific tabular view can be specified to define the fields that must be displayed in the target table. If a tabular view is not defined, all columns that a user is allowed to view, according to the granted access rights, are displayed. A tabular view is defined using the element `osd:defaultView/tabularView` under `xs:annotation/appinfo`.

The table below shows the elements that can be defined under `osd:defaultView/tabularView`.

Element	Description	Required
`column`	Define a field of the target table to display. The specified path must be absolute from the target table and must refer to an existing field. Several `column` elements can be defined to specify the fields that are to be displayed.	No
`sort`	Define a field that can be used to sort associated objects. Several`sort` elements can be defined to specify the fields that can be used to sort associated objects. Element `nodePath` defines the path of the field that can be used to sort associated objects. Element `isAscending` specifies whether the sort order is ascending (true) or descending (false).	No.

The following example shows how to define a tabular view from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
      <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
   </osd:association>
   <osd:defaultView>
    <tabularView>
     <column>/productRange</column>
     <column>/productCode</column>
     <column>/productLabel</column>
     <column>/productDescription</column>
     <sort>
        <nodePath>/productLabel</nodePath>
        <isAscending>true</isAscending>
     </sort>
    </tabularView>
   </osd:defaultView>
  </xs:appinfo>
 </xs:annotation>
```

```
</xs:element>
```

Actions in the user interface

In the user interface, it is possible to perform the following actions:

- **Create**: it allows directly creating an object in the target table of the association. When a new object is created, it is automatically associated with the current record.
- **Associate**: associates an existing object with the current record. In the case of an association over a link table, a record in the link table is automatically created to materialize the link between the current record and the existing object.
- **Delete**: deletes selected associated objects in the target table of the association.
- **Detach**: breaks the semantic link between the current record and the selected associated objects. In the case of an association over a link table, the records in the link table are automatically deleted, to break the links between the current record and associated objects.

Validation

Some controls can be defined on associations, in order to restrict associated objects. These controls are defined under the element `osd:association`.

The table below presents the controls that can be defined under xs:annotation/appInfo/osd:association.

Element	Description	Required
`minOccurs`	Specifies the minimum number of associated objects that are required for this association. This minimum number is defined using the element `value` and must be a positive integer.	No, by default the minimum is not restricted.
`maxOccurs`	Specifies the maximum number of associated objects that are allowed for this association. This maximum number is defined by the element `value` and must be either a positive integer or the raw string `unbounded` which indicates that this maximum is not restricted. The maximum number of associated objects must be greater than the minimum number of associated objects.	No, by default the maximum is not restricted.
`constraint`	Defines an XPath predicate for restricting associated records. It is only possible to use fields from the source and the target table when defining an XPath predicate. That is, if it is an association over a link table, it is not possible to use fields of the link table in the XPath predicate. In associated data sets, a validation message of the specified severity is added and displayed to the end user at validation time when an associated record does not comply with the specified constraint.	No.
`validation`	A validation message can be defined under the elements `minOccurs`, `maxOccurs` and `constraint`, using the element `validation`. The severity of the validation message is specified using the element `severity`. Possible severities are: `error`, `warning` and `info`. If the severity is not specified, then by default, the severity `error` is used. A localized validation message can be specified using the element `message`, which will be displayed to the end-user at validation time if an association does not comply with this constraint. Each localized message variant is defined in a nested message element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a message element with no xml:lang attribute.	No.

Data services integration

It is possible to define whether associated objects must be hidden in the Data Service `select` operation. For this, the property `osd:defaultView/hiddenInDataServices` under `xs:annotation/xs:appinfo` can be set on the association. Setting the property to 'true' will hide associated objects in the Data Service `select` operation. If this property is not defined, then by default, associated objects will be shown in the Data Service `select` operation.

See also

Hiding a field in Data Services *[p 496]*

Association field [p 214]

Examples

For example, the product catalog data model defined previously [p 454] specifies that a product belongs to a catalog (explicitly defined by a foreign key in the 'Products' table). The reverse relationship (that a catalog has certain products) is not easily represented in XML Schema, unless the 'Catalogs' table includes the following association that is the inverse of a foreign key:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
   <osd:association>
   <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
   </osd:association>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

For an association over a link table, we can consider the previous example and bring some updates. For instance, the foreign key in the 'Products' table is deleted and the relation between a product and a catalog is redefined by a link table (named 'Catalogs_Products') that has a primary key made of two foreign keys: one that refers to the 'Products' table (named 'productRef') and another to the 'Catalogs' table (named 'catalogRef'). The following example shows how to define an association over a link table from this new relationship:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
  <osd:association>
   <linkTable>
     <table>/root/Catalogs_Products</table>
     <fieldToSource>./catalogRef</fieldToSource>
     <fieldToTarget>./productRef</fieldToTarget>
   </linkTable>
  </osd:association>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

The following example shows an association that refers to a foreign key in another data set. In this example the 'Products' and 'Catalogs' tables are not in the same data set:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
   <osd:association>
   <tableRefInverse>
    <schemaLocation>urn:ebx:module:aModuleName:/WEB-INF/ebx/schema/products.xsd</schemaLocation>
    <dataSet>Products</dataSet>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
   </osd:association>
   </xs:appinfo>
   </xs:annotation>
</xs:element>
```

The following example defines an XPath filter to associate only products of type 'Technology':

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
   <osd:association>
   <tableRefInverse>
    <fieldToSource>/root/Products/catalog_ref</fieldToSource>
   </tableRefInverse>
   <xpathFilter>./productType = 'Technology'</xpathFilter>
   </osd:association>
   </xs:appinfo>
   </xs:annotation>
```

```
</xs:element>
```

The following example specifies the minimum number of products that are required for a catalog:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
      <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
     <minOccurs>
     <value>1</value>
     <validation>
      <severity>warning</severity>
      <message xml:lang="en-US">One product should at least be associated to this catalog.</message>
      <message xml:lang="fr-FR">Un produit doit au moins être associé à ce catalogue.</message>
     </validation>
    </minOccurs>
   </osd:association>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

The following example specifies that an catalog must contain at most ten products:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
    <maxOccurs>
     <value>10</value>
     <validation>
      <severity>warning</severity>
      <message xml:lang="en-US">Too much products for this catalog.</message>
      <message xml:lang="fr-FR">Ce catalogue a trop de produits.</message>
     </validation>
    </maxOccurs>
   </osd:association>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

74.4 Selection nodes

Attention

From version EBX5 5.5.0, it is recommended to use associations instead of selection nodes. Associations provide more features than selection nodes, and no further evolutions will be made on selection nodes. The only characteristic on selection nodes that is not supported by associations is the way to display the selection of records as a link.

See Associations [p 458] for more information.

An element declaration can define a dynamic contextual XPath selection. In this case, the user interface provides a link so that the user can navigate to the pre-filtered table corresponding to the selection.

A selection node is useful for creating an association between two entities in the user interface, as well as for validation purposes.

See also *Foreign keys [p 455]*

For example, the product catalog data model defined previously [p 454] specifies that a product belongs to a catalog (explicitly defined by a foreign key in the 'Products' table). The reverse relationship (that a catalog has certain products) is not easily represented in XML Schema, unless the 'Catalogs' table includes the following selection node:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
   <xs:annotation>
   <xs:appinfo>
   <osd:select>
   <xpath>/root/Products[catalog_ref =${../catalogId}]</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, to be valid, an catalog must be associated with at least one product.

The element `maxOccurs` specifies that, to be valid, an catalog must be associated with at most ten products.

Note

The "official" cardinality constraints (`minOccurs="0" maxOccurs="0"`) are required because, from an instance of XML Schema, the corresponding node is absent. In other words, a selection node is a "virtual" element as far as XML and XML Schema is concerned.

It is also possible to specify additional constraints on the relationship. In the following example, each relationship between an catalog and a product is valid if the date of creation of the product is prior to the date of publication of the catalog:

```
<osd:select>
 <xpath>/root/Products[catalog_ref =${../catalogId}]</xpath>
 <constraintPredicate>date-less-than(productCreationDate, ${../catalogPublicationDate})</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`	Specifies the selection to be performed, relative to the current node. Examples: `/root/Products[catalog_ref =${../catalogId}]` or `//Products[catalog_ref = ${../catalogId}]` or `../Products[catalog_ref =${../catalogId}]` . The path up to the predicate, for example `../Products`, specifies the target table to be filtered. This part of the path is resolved relative to the current table root. If the selection depends on the local state, the XPath expression predicate must include references to the node on which it depends using the notation `${<relative-path>}` where `relative-path` is a path that identifies the element relative to the node that holds the selection link. See EBX5 XPath supported syntax [p 253].	Yes.
`container`	Reference of the data set that contains the target table.	No, default is current data set.
`minOccurs`	Specifies an additional validation constraint: the selection must be at least the specified size.	No, default is 0.
`minOccursValidationMessage`	Specifies additional localized messages that will be displayed if the selection does not comply with the `minOccurs` constraint. A specific severity level can be defined in a nested `severity` element. The default severity is 'error'. Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute. To specify a default, non-localized message, define a `message` element with no `xml:lang` attribute.	No.
`maxOccurs`	Specifies an additional validation constraint: the selection must be at most the specified size.	No, by default the maximum is not restricted.
`maxOccursValidationMessage`	Specifies an additional localized message that will be displayed if the selection does not comply with the `maxOccurs` constraint. A specific severity level can be defined in a nested `severity` element. The default severity is 'error'. Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute. To specify a default, non-localized message, define a `message` element with no `xml:lang` attribute.	No.
`constraintPredicate`	Specifies an additional validation constraint: each relation of the selection must satisfy the specified	No.

Element	Description	Required
	XPath predicate. The notation `${<relative-path>}` has the same meaning as for the `xpath` element (see above).	
`constraintPredicateValidationMessage`	Specifies a localized message to display when the constraint predicate is not satisfied. A specific severity level can be defined in a nested `severity` element. The default severity is 'error'. Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute. To specify a default, non-localized message, define a `message` element with no `xml:lang` attribute.	No.

Important: When creating a data set, you can create a data set that defines a selection node to a container that does not yet exist in the repository. However, the content of this data set will not be available immediately upon creation. After the absent container is created, 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 data constraints in your data models. EBX5 supports XML Schema facets and provides extended and programmatic facets for advanced data controls.

This chapter contains the following topics:

75.1 XML Schema supported facets

The tables below show the facets that are supported by different data types.

Key:

- **X -** Supported
- **1 -** The `whiteSpace` facet can be defined, but is not interpreted by EBX5
- **2 -** In XML Schema, boundary facets are not allowed on the type `string`. Nevertheless, EBX5 allows such facets as extensions.
- **3 -** The `osd:resource` type only supports the facet `FacetOResource`, which is required. See Extended Facets [p 472].

	length	minLength	max Length	pattern	enumeration	white Space
`xs:string`	**X**	**X**	**X**	**X**	**X**	**1**
`xs:boolean`				**X**		**1**
`xs:decimal`				**X**	**X**	**1**
`xs:dateTime`				**X**	**X**	**1**
`xs:time`				**X**	**X**	**1**
`xs:date`				**X**	**X**	**1**
`xs:anyURI`	**X**	**X**	**X**	**X**	**X**	**1**
`xs:Name`	**X**	**X**	**X**	**X**	**X**	**1**
`xs:integer`				**X**	**X**	**1**
`osd:resource` [p 443]^3						

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
`xs:string`			**2**	**2**	**2**	**2**
`xs:boolean`						
`xs:decimal`	**X**	**X**	**X**	**X**	**X**	**X**
`xs:dateTime`			**X**	**X**	**X**	**X**
`xs:time`			**X**	**X**	**X**	**X**
`xs:date`			**X**	**X**	**X**	**X**

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
xs:anyURI						
xs:Name			2	2	2	2
xs:integer	X	X	X	X	X	X
osd:resource [p 443]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, using the standard XML Schema element `xs:unique`. This constraint indicates that a value or a set of values has to be unique inside a table.

Example:

In the example below, a uniqueness constraint is defined on the 'publisher' table, for the target field 'name'. This means that no two records in the 'publisher' table can have the same name.

```
<xs:element name="publisher">
 ...
 <xs:complexType>
  <xs:sequence>
   ...
   <xs:element name="name" type="xs:string" />
   ...
  </xs:sequence>
 </xs:complexType>
 <xs:unique name="uniqueName">
  <xs:annotation>
   <xs:appinfo>
    <osd:validation>
     <severity>error</severity>
     <message>Name must be unique in table.</message>
     <message xml:lang="en-US">Name must be unique in table.</message>
     <message xml:lang="fr-FR">Le nom doit être unique dans la table.</message>
    </osd:validation>
   </xs:appinfo>
  </xs:annotation>
  <xs:selector xpath="." />
  <xs:field xpath="name" />
 </xs:unique>
</xs:element>
```

A uniqueness constraint has to be defined within a table and has the following properties:

Property	Description	Mandatory
`name` attribute	Identifies the constraint in the data model.	Yes
`xs:selector` element	Indicates the table to which the uniqueness constraint applies using a restricted XPath expression ('..' is forbidden). It can also indicate an element within the table (without changing the meaning of the constraint).	Yes
`xs:field` element	Indicates the field in the context whose values must be unique, using a restricted XPath expression. It is possible to indicate that a set of values must be unique by defining multiple `xs:field` elements.	Yes

> **Note**
>
> Undefined values (null values) are ignored on uniqueness constraints applied to single fields. On multiple fields, undefined values are taken into account. That is, sets of values are considered as being duplicated if they have the same defined and undefined values.

Additional localized validation messages can be defined using the element `osd:validation` under the elements `annotation/appinfo`. If no custom validation messages are defined, a built-in validation message will be used.

Limitations:

1. The target of the `xs:field` element must be in a table.
2. The uniqueness constraint does not apply to fields inside an aggregated list.
3. The uniqueness constraint does not apply to computed fields.

75.2 **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 XML Schema conformance, these extended facets are defined under the element `annotation/appinfo/otherFacets`.

Foreign keys

EBX5 allows to create a reference to an existing table by the meaning of a specific facet. See Foreign keys [p 455] for more information.

Dynamic constraints

Dynamic constraint facets retain the semantics of XML Schema, but the value attribute is replaced with a path attribute that allows fetching the value from another element. The available dynamic constraints are:

- `length`
- `minLength`
- `maxLength`
- `maxInclusive`
- `maxExclusive`
- `minInclusive`
- `minExclusive`

Using these facets, the data model can be modified dynamically.

Example:

```
<xs:element name="amount">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:minInclusive path="/domain/Loan/Pricing/AmountMini/amount" />
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
   ...
</xs:element>
```

In this example, the boundary of the facet minInclusive is not statically defined. The value of the boundary comes from the node */domain/Loan/Pricing/AmountMini/amount*

FacetOResource constraint

This facet must be defined for each resource type. It has the following attributes:

`moduleName`	Indicates, using an alias, the EBX5 module that contains the resource. If the resource is contained in the current module, the alias must be preceded by "wbp". Otherwise, the alias must be one of the values defined in the element `<dependencies>` in the file `module.xml`.
`resourceType`	Represents the resource type using 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 `resourceType`. In the example below, the resource is in the directory `www/common/images/`. Thus, the resource is located at `www/common/images/promotion/` where `www/` is the directory at the same level as the `WEB-INF/` directory. Furthermore, if a resource is defined in a localized directory, for example `www/fr/`, it will only be taken into account if another resource with the same name is defined in the directory `www/common/`.

This facet has the same behavior as an enumeration facet: the values are collected by recursively listing all the files in the local path in the specified resource type directory in the specified module.

Example:

```
<xs:element name="promotion" type="osd:resource">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:FacetOResource osd:moduleName="wbp"
     osd:resourceType="ext-images" osd:relativePath="promotion/" />
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</ xs:element>
```

For an overview of the standard directory structure of an EBX5 module (Java EE web application), see Structure [p 437].

excludeValue constraint

This facet verifies that a value is not the same as the specified excluded value.

In this example, the empty string is excluded from the allowed values.

Example:

```
<xs:element name="roleName">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:excludeValue value="">
     <osd:validation>
      <severity>error</severity>
```

```
       <message>Please select address role(s).</message>
     </osd:validation>
    </osd:excludeValue>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType type="xs:string" />
</xs:element>
```

excludeSegment constraint

This facet verifies that a value is not included in a range of values. Boundaries are excluded.

Example:

In this example, values between 20000 and 20999 are not allowed.

```
<xs:element name="zipCode">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:excludeSegment minValue="20000" maxValue="20999">
     <osd:validation>
      <severity>error</severity>
       <message>Postal code not valid.</message>
     </osd:validation>
    </osd:excludeSegment>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType type="xs:string" />
</xs:element>
```

Enumeration facet defined using another node

By default, an enumeration facet is described statically in XML Schema.

The content of an enumeration facet can also be provided dynamically by a list of simple elements in the data model.

Example:

In this example, the content of an enumeration facet is sourced from the node `CountryList`.

```
<xs:annotation>
 <xs:appinfo>
  <osd:otherFacets>
   <osd:enumeration osd:path="../CountryList" />
  </osd:otherFacets>
 </xs:appinfo>
</xs:annotation>
```

The node `CountryList`:

- Must be an aggregated list, that is, `maxOccurs` > 1.
- Must be a list of elements of the same type as the node with the enumeration facet.

  Must be a node outside a table if the node with the enumeration facet is not inside a table.

  Must be a node outside a table or in the same table as the node with the enumeration facet if the node with this enumeration is inside a table.

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>
```

75.3 **Programmatic facets**

A programmatic constraint can be added to any XML element declaration for a simple type.

In order to guarantee XML Schema conformance, programmatic constraints are specified under the element `annotation/appinfo/otherFacets`.

Programmatic constraints

A programmatic constraint is defined by a Java class that implements the interface `Constraint`API.

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol.

Example:

In the example below, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

```
<xs:element name="amount">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraint class="com.foo.CheckAmount">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraint>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

See also ***JavaBean specifications*** `Package com.orchestranetworks.schema`API

Programmatic enumeration constraints

An enumeration constraint adds an ordered list of values to a basic programmatic constraint. This facet allows selecting a value from a list. It is defined by a Java class that implements the interface `ConstraintEnumeration`API.

Example:

```
<xs:element name="amount">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraintEnumeration class="com.foo.CheckAmountInEnumeration">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraintEnumeration>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Constraint on 'null' values

In some cases, a value is only mandatory if some conditions are satisfied, for example, if another field has a given value. In this case, the standard XML Schema attribute `minOccurs` is insufficient because it is static.

In order to check if a value is mandatory according to its context, the following requirements must be satisfied:

1. A programmatic constraint must be defined by a Java class (see above).
2. This class must implement the interface `ConstraintOnNull`API.
3. The XML Schema cardinality attributes must specify that the element is optional (`minOccurs="0"` and `maxOccurs="1"`).

> **Note**
>
> By default, constraint on 'null' values is not checked upon user input. In order to enable checking at input, the 'checkNullInput' property [p 480] must be set. Also, if the element is terminal, the data set must also be activated.

Example:

```
<xs:element name="amount" minOccurs="0" maxOccurs="1">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraint class="com.foo.CheckIfNull">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraint>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Constraints on table

A constraint on table is defined by a Java class that implements the interface `ConstraintOnTable`API. It can only be defined on table nodes.

As additional parameters can be defined. the implemented Java class must conform to the JavaBean protocol.

Example:

In the example below, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:appinfo>
   <osd:table>
    <primaryKeys>/key</primaryKeys>
   </osd:table>
   <osd:otherFacets>
    <osd:constraint class="com.foo.checkTable">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraint>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
```

```
</xs:element>
```

Attention

For performance reasons, constraints on tables are only checked when getting the validation report of a data set or table. This means that these constraints are not checked when updates, such as record insertions, deletions or modifications, occur on tables. However, the internal incremental validation framework will optimize the validation cost of these constraints if dependencies are defined. For more information, see Validation [p 322].

See also ***JavaBeans Specifications*** *`Package com.orchestranetworks.schema`*API

75.4 Control policy

Blocking and non-blocking constraints

When an update in the repository is performed, and this update adds a validation error according to a given constraint, it is possible to specify whether the new error blocks the update (and cancels the transaction) or it is considered as non-blocking (so that the update can be committed and the error can be corrected later). The element `blocksCommit` within the element `osd:validation` allows this specification, with the following supported values:

`onInsertUpdateOrDelete`	Specifies that the constraint must always remain valid after an operation (data set update, data set deletion, record creation, update or deletion). In this case, any operation that would violate the constraint is rejected and the values remain unchanged. This is the default and mandatory policy for primary key constraints, data type conversion constraints (an integer or a date must be well-written) and also structural and foreign key constraints in relational data models and mapped tables.
`onUserSubmit-checkModifiedValues`	Specifies that the constraint must remain valid whenever a user modifies the associated value and submits a form. In this case, any form input that would violate the constraint is rejected and the values remain unchanged. This is the default policy for all blocking constraints mentioned in the previous case. For example, a foreign key constraint is by default not blocking (a record referred by other records can be deleted, etc.), except in the context of a form submit.
`never`	Specifies that the constraint must never block operations. In this case, any operation that would violate the constraint is allowed. In the context of the user interface this constraint does not block form submission if the user sets a value that violates this constraint.

On foreign key constraints, the control policy that blocks all operations does not apply to filtered records. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected and a validation error occurs.

It is not possible to specify a control policy on structural constraints that are defined on relational data models or in mapped tables. That is, this property is not available for fixed length, maximum length, maximum number of digits, and decimal place constraints due to the validation policy of the underlying RDBMS blocking constraints.

This property does not apply to archive imports. That is, all blocking constraints, except structural constraints, are always disabled when importing archives.

See also

Facet validation message with severity *[p 490]*

Foreign keys *[p 455]*

Relational mode *[p 269]*

XML Schema facet

The control policy is described by the element `osd:validation` in `annotation/appinfo` under the definition of the facet.

Example:

```
<xs:element name="zipCode">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:minInclusive value="1000">
    <xs:annotation>
     <xs:appinfo>
      <osd:validation>
       <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
      </osd:validation>
     </xs:appinfo>
    </xs:annotation>
   </xs:minInclusive>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

XML Schema enumeration facet

The control policy is described by the element `osd:enumerationValidation` in `annotation/appinfo` under the definition of the field.

Example:

```
<xs:element name="Gender">
 <xs:annotation>
  <xs:appinfo>
   <osd:enumerationValidation>
    <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
   </osd:enumerationValidation>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="0" osd:label="male" />
   <xs:enumeration value="1" osd:label="female" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

EBX5 facet

The control policy is described by the element `osd:validation` under the definition of the facet (which is defined in `annotation/appinfo/otherFacets`).

The control policy with values `onInsertUpdateOrDelete` and `onUserSubmit-checkModifiedValues` is only available on `osd:excludeSegment`, `osd:excludeValue` and `osd:tableRef` EBX5 facets.

The control policy with value `never` can be defined on all EBX5 facets. On programmatic constraints, the control policy with value `never` can only be set directly during the setup of the corresponding constraint. See `ConstraintContext.setBlocksCommitToNever`API and `ConstraintContextOnTable.setBlocksCommitToNever`API in the Java API for more information.

Example:

```
<xs:element name="price" type="xs:decimal">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:minInclusive path="../priceMin">
     <osd:validation>
      <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
     </osd:validation>
    </osd:minInclusive>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Check 'null' input

According to the EBX5 default validation policy, in order to allow temporarily incomplete input, a mandatory element is not checked for completion upon user input. Rather, it is verified at data set validation only. If completion must be checked immediately upon user input, the element must additionally specify the attribute `osd:checkNullInput="true"`.

> **Note**
>
> A value is mandatory if the data model specifies a mandatory element, either statically, using `minOccurs="1"`, or dynamically, using a constraint on 'null'. For terminal elements, mandatory values are only checked for an activated data set. For non-terminal elements, the data set does not need to be activated.

Example:

```
<xs:element name="amount" osd:checkNullInput="true" minOccurs="1">
 ...
</xs:element>
```

See also

Constraint on 'null' *[p 477]*

Whitespace management *[p 481]*

Empty string management *[p 482]*

EBX5 whitespace management for data types

According to XML Schema (see http://www.w3.org/TR/xmlschema-2/#rf-whiteSpace), whitespace handling must follow one of the procedures *preserve*, *replace* or *collapse*:

preserve	No normalization is performed, the value is unchanged.
replace	All occurrences of `#x9` (tab), `#xA` (line feed) and `#xD` (carriage return) are replaced with `#x20` (space).
collapse	After the processing according to the replace procedure, contiguous sequences of `#x20` are then collapsed to a single `#x20`, and any leading or trailing `#x20`s are removed.

EBX5 general whitespace handling

EBX5 complies with the XML Schema recommendation:

- For nodes of type `xs:string`, whether a primary key element or not, whitespaces are always preserved and an empty string is never converted to `null`.
- For other nodes (non-`xs:string` type), whitespaces are always collapsed and empty strings are converted to `null`.

Attention

Exceptions:

- For nodes of type `osd:html` or `osd:password`, whitespaces are always preserved and empty strings are converted to `null`.
- For nodes of type `xs:string` that define the property `osd:checkNullInput="true"`, an empty string is interpreted as `null` at user input by EBX5.

Whitespace handling for primary keys 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 whitespace values at validation.

However, if the primary key node specifies its own `xs:pattern` facet, this facet overrides the default EBX5 constraint. For example, the specific pattern `".*"` would accept any strings, although this is not recommended.

The default constraint allows handling certain ambiguities. For example, it would be difficult for a user to distinguish between the following strings: "12 34" and "12 34". For generic values, this would not create conflicts, however, errors would occur for primary keys.

See also *Tables and relationships* *[p 451]*

Empty string management

Default conversion

For nodes of type `xs:string`, no distinction is made at user input between an empty string and a `null` value. That is, an empty string value is automatically converted to `null` at user input.

Distinction between empty strings and 'null' value

There are certain cases where the distinction is made between an empty string and the `null` value, such as when:

- A primary key defines a pattern that allows empty strings.
- An element defines a foreign key constraint and the target table has a single primary key defining a pattern that allows empty strings.
- An element defines a static enumeration that contains an empty string.
- An element defines a dynamic enumeration to another element with one of the aforementioned cases.

If the distinction is made between an empty string and a `null` value, this implies the following behaviors:

- An empty string will not be converted to `null` at user input,
- Input fields for nodes of type `xs:string` will display an additional button for setting the value of the node to `null`,
- At validation time an empty string will be considered to be a compliant value with regard to the `minOccurs="1"` property.

Attention

In relational mode, the Oracle database does not support the distinction between empty strings and `null` values, and these specific cases are not supported.

See also *Relational mode* *[p 269]*

75.5 Triggers and functions

Computed values

By default, data is read and persisted in the XML repository. Nevertheless, data may be the result of a computation and/or external database access, for example, an RDBMS or a central system.

EBX5 allows taking into account other data in the current data set context.

This is made possible by defining *functions*.

A function is specified in the data model using the `osd:function` element (see example below).

- The value of the *class* attribute must be the qualified name of a Java class that implements the Java interface `ValueFunction`API.
- Additional parameters may be specified at the data model level, in which case the JavaBean convention is applied.

Example:

```
<xs:element name="computedValue">
 <xs:annotation>
  <xs:appinfo>
   <osd:function class="com.foo.ComputeValue">
    <param1>...</param1>
    <param...n>...</param...n>
   </osd:function>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Triggers

Data sets or table records can be associated with methods that are automatically executed when some operations are performed, such as creations, updates, or deletes.

In the data model, these triggers must be declared under the `annotation/appinfo` element using the `osd:trigger` element.

For data set triggers, a Java class that extends the abstract class `InstanceTrigger`API must be declared inside the element `osd:trigger`.

In the case of data set triggers, it is advised to define `annotation/appinfo/osd:trigger` tags just under the `root` element of the data model.

Example:

```
<xs:element name="root" osd:access="--">
   ...
   <xs:annotation>
  <xs:appinfo>
   <osd:trigger class="com.foo.MyInstanceTrigger">
    <param1>...</param1>
    <param...n>...</param...n>
   </osd:trigger>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

For the definition of table record triggers, a Java class that extends the abstract class `TableTrigger`API must be defined inside the `osd:trigger` element. It is advised to define the `annotation/appinfo/osd:trigger` elements just under the element describing the associated table or table type.

Examples:

On a table element:

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:appinfo>
   <osd:table>
    <primaryKeys>/key</primaryKeys>
   </osd:table>
   <osd:trigger class="com.foo.MyTableTrigger" />
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

On a table type element:

```
<xs:complexType name="MyTableType">
   ...
   <xs:annotation>
    <xs:appinfo>
    <osd:trigger class="com.foo.MyTableTrigger">
    <param1>...</param1>
    <param...n>...</param...n>
    </osd:trigger>
    </xs:appinfo>
   </xs:annotation>
   ...
</xs:complexType>
```

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol. In the example above, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

Auto-incremented values

It is possible to define auto-incremented values. Auto-incremented values are only allowed inside tables, and they must be of type `xs:int` or `xs:integer`.

An auto-increment is specified in the data model using the element `osd:autoIncrement` under the element `annotation/appinfo`.

Example:

```
<xs:element name="autoIncrementedValue" type="xs:int">
 <xs:annotation>
  <xs:appinfo>
   <osd:autoIncrement />
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Also, there are two optional elements, `start` and `step`:

- The `start` attribute specifies the first value for this auto-increment. If this attribute is not specified, then the value `1` is set by default.
- The `step` attribute specifies the step for the next value to be generated by the auto-increment. If this attribute is not specified, then the value *1* is set by default.

Example:

```
<xs:element name="autoIncrementedValue" type="xs:int">
 <xs:annotation>
  <xs:appinfo>
   <osd:autoIncrement>
    <start>100</start>
    <step>5</step>
   </osd:autoIncrement>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

A field specifying an `osd:autoIncrement` has the following behavior:

- The computation and allocation of the field value are performed whenever a new record is inserted and the field value is undefined.
- No allocation is performed if a programmatic insertion already specifies a non-`null` value. For example, if an archive import or an XML import specifies the value, that value is preserved.

  Consequently, the allocation is not performed for a record insertion in occulting or overwriting modes.
- A newly allocated value is, whenever possible, unique in the scope of the repository. More precisely, the uniqueness of the allocation spans over all the 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 mass update transaction that specifies values is performed at the same time as a transaction that allocates a value on the same field, it is possible that the latter transaction will allocate a value that will be set by the first transaction (there is no locking between different 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 by administrators in the Administration area. This field is automatically updated so that it defines the greatest value ever set on the associated `osd:autoIncrement` field, in any instance or data space in the repository. This value is computed, taking into account the max value found in the table being updated.

In certain cases, for example when multiple environments have to be managed (development, test, production), each with different auto-increment ranges, it may be required to avoid this "max value" check. This particular behavior can be achieved using the `disableMaxTableCheck` property. It is generally not recommended to enable this property unless it is absolutely necessary, as this could generate conflicts in the auto-increment values. However, this property can be set in the following ways:

- Locally, by setting a parameter element in auto-increment declaration: `<disableMaxTableCheck>true</disableMaxTableCheck>`,
- For the whole data model, by setting `<osd:disableMaxTableCheck="true">` in the element `xs:appinfo` of the data model declaration, or
- Globally, by setting the property `ebx.autoIncrement.disableMaxTableCheck=true` in the EBX5 main configuration file.

  See EBX5 main configuration file [p 347].

> **Note**
>
> When this option is enabled globally, it becomes possible to create records in the table of auto-increments, for example, by importing from XML or CSV. If this option is not selected, creating records in the table of auto-increments is prohibited to ensure the integrity of the repository.

CHAPTER **76**

Labels and messages

EBX5 allows you to provide labels and error messages for your data models to be displayed in the interface.

This chapter contains the following topics:

76.1 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.

Two different notations can be used:

Full	The label and description are defined by the elements `<osd:label>` and `<osd:description>` respectively.
Simple	The label is extracted from the text content, ending at the first period (`'.'`), with a maximum of 60 characters. The description uses the remainder of the text.

The description may also have a hyperlink, either a standard HTML `href` to an external document, or a link to another node of the adaptation within EBX5.

- When using the `href` notation or any other HTML, it must be properly escaped.
- EBX5 link notation is not escaped and must specify the path of the target, for example:

  `<osd:link path="../misc1">Link to another node in the adaptation</osd:link>`

Example:

```
<xs:element name="misc1" type="xs:string">
 <xs:annotation>
  <xs:documentation>
```

```
   Miscellaneous 1. This is the description of miscellaneous element #1.
   Click <a href="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 to the description of the node (element `osd:description`).

> **Note**
>
> Regarding whitespace management, the label of a node is always *collapsed* when displayed. That is, contiguous sequences of blanks are collapsed to a single blank, and leading and trailing blanks are removed. In descriptions, however, whitespaces are always *preserved*.

Dynamic labels and descriptions

As an alternative to statically defining the localized labels and descriptions for each node, it is possible to specify a Java class that programmatically determines the labels and descriptions for the nodes of the data model. To define the class, include the element `osd:documentation`, with the attribute `class` in the data model. It is possible to pass JavaBean properties using nested parameter elements.

Example:

```
<xs:schema ...>
 <xs:annotation>
  <xs:appinfo>
   <osd:documentation class="com.foo.MySchemaDocumentation">
    <param1>...</param1>
    <param2>...</param2>
   </osd:documentation>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:schema ...>
```

The labels and descriptions that are provided programmatically take precedence over the ones defined locally on individual nodes.

See also *`SchemaDocumentation`*API

76.2 Enumeration labels

In an enumeration, a simple, non-localized label can be added to each enumeration element, using the attribute `osd:label`.

> **Attention**
>
> Labels defined for an enumeration element are always `collapsed` when displayed.

Example:

```
<xs:element name="Service" maxOccurs="unbounded">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="1" osd:label="Blue" />
   <xs:enumeration value="2" osd:label="Red" />
   <xs:enumeration value="3" osd:label="White" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

It is also possible to fully localize the labels using the standard `xs:documentation` element. If both non-localized and localized labels are added to an enumeration element, the non-localized label will be displayed in any locale that does not have a label defined.

Example:

```
<xs:element name="access" minOccurs="0">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="readOnly">
    <xs:annotation>
     <xs:documentation xml:lang="en-US">
      read only
     </xs:documentation>
     <xs:documentation xml:lang="fr-FR">
      lecture seule
     </xs:documentation>
    </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value="readWrite">
    <xs:annotation>
     <xs:documentation xml:lang="en-US">
      read/write
     </xs:documentation>
     <xs:documentation xml:lang="fr-FR">
      lecture écriture
     </xs:documentation>
    </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value="hidden">
    <xs:annotation>
     <xs:documentation xml:lang="en-US">
      hidden
     </xs:documentation>
     <xs:documentation xml:lang="fr-FR">
      masqué
     </xs:documentation>
    </xs:annotation>
   </xs:enumeration>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

76.3 **Mandatory error message (osd:mandatoryErrorMessage)**

If the node specifies the attribute `minOccurs="1"` (default behavior), then an error message, which must be provided, is displayed if the user does not complete the field. This error message can be defined specifically for each node using the element `osd:mandatoryErrorMessage`.

Example:

```
<xs:element name="birthDate" type="xs:date">
 <xs:annotation>
  <xs:documentation>
   <osd:mandatoryErrorMessage>
    Please give your birth date.
   </osd:mandatoryErrorMessage>
  </xs:documentation>
 </xs:annotation>
</xs:element>
```

The mandatory error message can be localized:

```
<xs:documentation>
 <osd:mandatoryErrorMessage xml:lang="en-US">
  Name is mandatory
 </osd:mandatoryErrorMessage>
 <osd:mandatoryErrorMessage xml:lang="fr-FR">
  Nom est obligatoire
 </osd:mandatoryErrorMessage>
</xs:documentation>
```

> **Note**
>
> Regarding whitespace management, the enumeration labels are always *collapsed* when displayed.

76.4 Conversion error message

For each predefined XML Schema element, it is possible to define a specific error message if the user entry has an incorrect format.

Example:

```
<xs:element name="email" type="xs:string">
 <xs:annotation>
  <xs:documentation>
   <osd:ConversionErrorMessage xml:lang="en-US">
    Please enter a valid e-mail address.
   </osd:ConversionErrorMessage>
   <osd:ConversionErrorMessage xml:lang="fr-FR">
    Saisissez un e-mail valide.
   </osd:ConversionErrorMessage>
  </xs:documentation>
 </xs:annotation>
</xs:element>
```

76.5 Facet validation message with severity

The validation message that is displayed when the value of a field does not comply with a constraint can define a custom severity, a default non-localized message, and localized message variants. If no severity is specified, the default level is `error`. Blocking constraints *must* have the severity `error`.

XML Schema facet (osd:validation)

The validation message is described by the element `osd:validation` in `annotation/appinfo` under the definition of the facet.

Example:

```
<xs:element name="zipCode">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <!--facet is not localized, but validation message is localized-->
   <xs:minInclusive value="01000">
    <xs:annotation>
     <xs:appinfo>
      <osd:validation>
       <severity>error</severity>
       <message>Non-localized message.</message>
       <message xml:lang="en-US">English error message.</message>
       <message xml:lang="fr-FR">Message d'erreur en français.</message>
      </osd:validation>
     </xs:appinfo>
    </xs:annotation>
   </xs:minInclusive>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

XML Schema enumeration facet (osd:enumerationValidation)

The validation message is described by the element `osd:enumerationValidation` in `annotation/appinfo` under the definition of the field.

Example:

```
<xs:element name="Gender">
 <xs:annotation>
  <xs:appinfo>
   <osd:enumerationValidation>
    <severity>error</severity>
    <message>Non-localized message.</message>
    <message xml:lang="en-US">English error message.</message>
    <message xml:lang="fr-FR">Message d'erreur en français.</message>
   </osd:enumerationValidation>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="0" osd:label="male" />
   <xs:enumeration value="1" osd:label="female" />
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

EBX5 facet (osd:validation)

The validation message is described by the element `osd:validation` under the definition of the facet (which is defined in `annotation/appinfo/otherFacets`).

Example:

```
<xs:element name="price" type="xs:decimal">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:minInclusive path="../priceMin">
     <osd:validation>
      <severity>error</severity>
      <message>Non-localized message.</message>
      <message xml:lang="en-US">English error message.</message>
      <message xml:lang="fr-FR">Message d'erreur en français.</message>
     </osd:validation>
    </osd:minInclusive>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

CHAPTER 77

Additional properties

This chapter contains the following topics:

77.1 Default values

In a data model, it is possible to specify a default value for a field using the attribute `default`. This property is used to assign a default value if no value is defined for a field.

The default value is displayed in the user input field at creation time. That is, the default value will be displayed when creating a new record or adding a new occurrence to an aggregated list.

Example:

In this example, the element specifies a default string value.

```
<xs:element name="fieldWithDefaultValue" type="xs:string" default="aDefaultValue" />
```

77.2 Access properties

The attribute `osd:access` defines the access mode, that is, whether the data of a particular data model node can be read and/or written. This attribute must have one of the following values: `RW`, `R-`, `CC` or `--`.

For each XML Schema node, three types of adaptation are possible:

1. *Adaptation terminal node*

   This node is displayed with an associated value in EBX5. When accessed using the method `Adaptation.get()`, it uses the adaptation search algorithm.
2. *Adaptation non-terminal node*

   This node is a complex type that is only displayed in EBX5 if it has one child node that is also an adaptation terminal node. It has no value of its own. When accessed using the method `Adaptation.get()`, it returns `null`.
3. *Non-adaptable node*

   This node is not an adaptation terminal node and has no child adaptation terminal nodes. This node is never displayed in EBX5. When access using the method `Adaptation.get()`, it returns the node default value if one is defined, otherwise returns `null`.

See also *`Adaptation`*API

Access mode	Behavior
`RW`	*Adaptation terminal node*: value can be read and written in EBX5.
`R-`	*Adaptation terminal node*: value can only be read in EBX5.
`CC`	*Cut*: This is not an adaptation terminal node and none of its children are adaptation terminal nodes. This "instruction" has priority over any child node regardless of the value of their *access* attribute.
`--`	If the node is a simple type, it is not adaptable. If the node is a complex type, it is not an adaptation terminal node and does not define any child nodes. The `root` node of a data model must specify this access mode.
Default	If the *access* attribute is not defined: • If the node is a computed value, it is considered to be `R-` • If the node is a simple type and its value is not computed, it is considered to be `RW` • If the node is an aggregated list, it is then a terminal value and is considered to be `RW` • Otherwise, it is not an adaptation terminal node and it does not define anything about its child nodes.

Example:

In this example, the element is adaptable because it is an adaptation terminal node.

```
<xs:element name="proxyIpAddress" type="xs:string" osd:access="RW"/>
```

77.3 **Information**

The element `osd:information` allows specifying additional information. This information can then be used by the integration code, for any purpose, by calling the method `SchemaNode.getInformation`API.

Example:

```
<xs:element name="misc" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
   <osd:information>
    This is the text information of miscellaneous element.
   </osd:information>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

77.4 **Default view**

Hiding a field in the default view

It is possible for a field to be hidden by default in EBX5 by using the element `osd:defaultView/hidden`. This property is used to hide elements from the default view of a data set without defining specific access permissions. That is, elements hidden by default will not be visible in any default forms and views, whether tabular or hierarchical, generated from the structure of the associated data model.

Attention

- If 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 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:

In this example, the element is hidden in the default view of a data set.

```
<xs:element name="hiddenField" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hidden>true</hidden>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

Hiding a field in search tools

To specify whether or not to hide an element in search tools, use the element `osd:defaultView/hiddenInSearch="true|false|textSearchOnly"`.

If this property is set to `true` then the field will not be selectable in the text and typed search tools of a data set.

If this property is set to `textSearchOnly` then the field will not be selectable only in the text search of a data set but it will be selectable in the typed search.

> **Note**
>
> If a group is configured as hidden in search tools or only in the text search, then all the fields nested under this group will not be displayed respectively in the search tools or only in the text search.

Example:

```
<xs:element name="hiddenFieldInSearch" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hiddenInSearch>true</hiddenInSearch>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the text and typed search tools of a data set.

```
<xs:element name="hiddenFieldOnlyInTextSearch" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hiddenInSearch>textSearchOnly</hiddenInSearch>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

In this example, the element is hidden only in the text search tool of a data set.

Hiding a field in Data Services

To specify whether or not to hide an element in data services, use the element `osd:defaultView/hiddenInDataServices`. For more information, see Disabling fields from data model [p 213].

> **Note**
>
> - If a group is configured as being hidden, then all the fields nested under this group will be considered as hidden by data services.

Example:

```
<xs:element name="hiddenFieldInDataService" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hiddenInDataServices>true</hiddenInDataServices>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the Data Service select operation.

Defining a view for the combo box selector of a foreign key

It is possible to specify a published view that will be used to display the target table or the hierarchical view of a foreign key for a smoother selection. If a view has been defined, the selector will be displayed in the user interface in the combo box of this foreign key. The definition of a view can be done by using the XML Schema element `osd:defaultView/widget/viewForAdvancedSelection`.

Note

- This property can only be defined on foreign key fields.
- The published view must be associated with the target table of the foreign key.
- If the published view does not exist then the advanced selection is not available in the foreign key field.

Example:

In this example, the name of a published view is defined to display the target table of a foreign key in the advanced selection.

```
<xs:element name="catalog_ref" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:tableRef>
     <tablePath>/root/Catalogs</tablePath>
    </osd:tableRef>
     </osd:otherFacets>
    <osd:defaultView>
    <widget>
     <viewForAdvancedSelection>catalogView</viewForAdvancedSelection>
    </widget>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

See Combo-box selector [p 51] for more information.

77.5 Comparison mode

The attribute `osd:comparison` can be included on a terminal node element in order to set its comparison mode. This mode controls how differences are detected for the element during comparisons. The possible values for the attribute are:

default	Element is visible during comparisons of its data.
ignored	No changes will be detected when comparing two versions of the content in records or data sets. During a merge, the data values of an ignored element are not merged when contents are updated, even if the values are different. For new content, the values of ignored elements are merged. During an archive import, values of ignored elements are not imported when contents are updated. For new content, the values of ignored elements are imported.

Note

- If a group is configured as being ignored during comparisons, then all the fields nested under this group will also be ignored.
- If a terminal field does not include the attribute `osd:comparison`, then it will be included by default during comparisons.

Restrictions:

- This property cannot be defined on non-terminal fields.
- Primary key fields cannot be ignored during comparison.

Example:

In this example, the first element is explicitly ignored during comparison, the second element is explicitly included.

```
<xs:element name="fieldExplicitlyIgnoredInComparison"
 type="xs:string" minOccurs="0" osd:comparison="ignored"/>
<xs:element name="fieldExplicitlyNotIgnoredInComparison"
 type="xs:string" minOccurs="0" osd:comparison="default"/>
```

77.6 Apply last modifications policy

The attribute `osd:applyLastModification` can be defined on a terminal node element in order to specify if this element must be included or not in the apply last modifications service that can be executed in a table of a data set.

The possible values for the attribute are:

default	Last modifications can be applied to this element.
ignored	This element is ignored from the apply last modifications service. That is, the last modification that has been performed on this element cannot be applied to other records.

Note

- If a group is configured as being ignored by the apply last modifications service, then all fields nested under this group will also be ignored.
- If a terminal field does not include the attribute `osd:applyLastModification`, then it will be included by default in the apply last modifications service.

Restriction:

- This property cannot be defined on non-terminal fields.

Example:

In this example, the first element is explicitly ignored in the apply last modifications service, the second element is explicitly included.

```
<xs:element name="fieldExplicitlyIgnoredInApplyLastModification"
 type="xs:string" minOccurs="0" osd:applyLastModification="ignored"/>
<xs:element name="fieldExplicitlyNotIgnoredApplyLastModification"
 type="xs:string" minOccurs="0" osd:applyLastModification="default"/>
```

77.7 Categories

Categories can be used for "filtering", by restricting 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

Categories do not apply to table record nodes, except the category 'Hidden'.

Java reference

CHAPTER **78**

Mapping to Java

This chapter contains the following topics:

78.1 How to access data from Java?

Read access

Data can be read from various generic Java classes, mainly `Adaptation`API and `ValueContext`API. The getter methods for these classes return objects that are typed according to the mapping rules described in the section Mapping of data types [p 503].

Write access

Updates of data must be performed in a well-managed context:

- In the context of a procedure execution, by calling the methods `setValue...` of the interface `ValueContextForUpdate`API, or
- During user input validation, by calling the method `setNewValue` of the class `ValueContextForInputValidation`API.

Modification of mutable objects

According to the mapping that is described in the section Mapping of data types [p 503], some accessed Java objects are mutable objects. These are instances of `List` , `Date` or any JavaBean. Consequently, these objects can be locally modified by their own methods. However, such modifications will remain local to the returned object unless one of the above setters is invoked and the current transaction is successfully committed.

78.2 **Concurrency and isolation levels**

Highest isolation level

The highest isolation level in ANSI/ISO SQL is `SERIALIZABLE`. Three execution methods guarantee the `SERIALIZABLE` isolation level within the scope of a data space:

- If the client code is run inside a `Procedure`API container. This is the case for every update, for exports to XML, CSV or archive, and for data services.
- If the client code accesses a data space that has been explicitly locked. See `LockSpec`API.
- If the client code accesses data in a snapshot.

> **Note**
>
> For custom read-only transactions that run on a data space, it is recommended to use `ReadOnlyProcedure`API.

Default isolation level

If the client code is run outside the contexts that enable `SERIALIZABLE`, its isolation level depends on the persistence mode:

- In semantic mode, the default isolation level is `READ UNCOMMITTED`.
- In relational mode, the default isolation level is the database default isolation level.

See also *Overview of modes* *[p 269]*

Java access specificities

In a Java application, a record is represented by an instance of the class `Adaptation`. This object is initially linked to the corresponding persisted record, however, unless the client code is executed in a context that enables the SERIALIZABLE [p 503] isolation level, the object can become "disconnected" from the persisted record. If this occurs and concurrent updates have been performed, they will not be reflected in the `Adaptation` object.

Therefore, it is important for the client code to either be in a `SERIALIZABLE` context, or to regularly look up or refresh the `Adaptation` object.

See also

*`AdaptationHome.findAdaptationOrNull`*API

*`AdaptationTable.lookupAdaptationByPrimaryKey`*API

*`Adaptation.getUpToDateInstance`*API

78.3 **Mapping of data types**

This section describes how XML Schema type definitions and element declarations are mapped to Java types.

Simple data types

Basic rules for simple data types

Each XML Schema simple type corresponds to a Java class, the mapping is documented in the table XML Schema built-in simple types [p 442].

See also *`SchemaNode.createNewOccurrence`*API

Multiple cardinality on a simple element

If the attribute `maxOccurs` is greater than 1, the element is an aggregated list and the corresponding instance in Java is an instance of `java.util.List`.

Elements of the list are instances of the Java class that is determined from the mapping of the simple type (see previous section).

Complex data types

Complex type definitions without a class declaration

By default (no attribute `osd:class`), a terminal node of a complex type is instantiated using an internal class. This class provides a generic JavaBean implementation. However, if custom client Java code has to access these values, it is recommended to use a custom JavaBean. To do so, use the `osd:class` declaration described in the next section.

It is also possible to transparently instantiate, read and modify the mapped Java object, with or without the attribute `osd:class`, by invoking the methods `SchemaNode.createNewOccurrence`API, `SchemaNode.executeRead`API and `SchemaNode.executeWrite`API.

Mapping of complex types to custom JavaBeans

It is possible to map an XML Schema complex type to a custom Java class. This is done by adding the attribute `osd:class` to the complex node definition. Unless the element has `xs:maxOccurs > 1`, you must also specify the attribute `osd:access` for the node to be considered a *terminal* node. If the element has `xs:maxOccurs > 1`, it is automatically considered to be terminal.

The custom Java class must conform to the JavaBean protocol. This means that each child of the complex type must correspond to a JavaBean property of the class. Additionally, each JavaBean property must be a read-write property, and its implementation must ensure that the value set by the setter method is returned, as-is, by the getter method. Contextual computations are not allowed in these methods.

Example

In this example, the class Java `com.carRental.Customer` must define the methods `getFirstName()` and `setFirstName(String)`.

A JavaBean can have a custom user interface within EBX5, by using a `UIBeanEditor`API.

```
<xs:element name="customer" osd:access="RW">
  <xs:complexType name="subscriber" osd:class="com.carRental.Customer">
 <xs:sequence>
   <xs:element name="firstName" type="xs:string"/>
   ...
 </xs:sequence>
  </xs:complexType>
</xs:element>
```

Multiple cardinality on a complex element

If the attribute `maxOccurs` is greater than 1, then the corresponding instance in Java is:

- An instance of `java.util.List` for an aggregated list, where every element in the list is an instance of the Java class determined by the mapping of simple types [p 442], or
- An instance of `AdaptationTable`API, if the property `osd:table` is specified.

78.4 Java bindings

Java bindings allow generating Java types that reflect the structure of the data model. The Java code generation can be done in the user interface. See Generating Java bindings [p 507].

Benefits

Ensuring the link between XML Schema structure and Java code provides a number of benefits:

- **Development assistance:** Auto-completion when typing an access path to parameters, if supported by your IDE.
- **Access code verification:** All accesses to parameters are verified at code compilation.
- **Impact verification:** Each modification of the data model impacts the code compilation state.
- **Cross-referencing:** By using the reference tools of your IDE, it is easy to verify where a parameter is used.

Consequently, it is strongly recommended to use Java bindings.

XML declaration

The specification of the Java types to be generated from the data model is included in the main schema.

Each binding element defines a generation target. It must be located at, in XPath notation, `xs:schema/xs:annotation/xs:appinfo/ebxbnd:binding`, where the prefix `ebxbnd` is a reference to the namespace identified by the URI `urn:ebx-schemas:binding_1.0`. Several binding elements can be defined if you have different generation targets.

The attribute `targetDirectory` of the element `ebxbnd:binding` defines the root directory used for Java type generation. Generally, it is the directory containing the project source code, `src`. A relative path is interpreted based on the current runtime directory of the VM, as opposed to the XML schema.

See bindings XML Schema.

Bindings XML example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:ebxbnd="urn:ebx-schemas:binding_1.0">
 <xs:annotation>
  <xs:appinfo>
   <!-- The bindings define how this schema will be represented in Java.
   Several <binding> elements may be defined, one for each target. -->
   <ebxbnd:binding
    targetDirectory="../_ebx-demos/src-creditOnLineStruts-1.0/">
    <javaPathConstants typeName="com.creditonline.RulesPaths">
     <nodes root="/rules" prefix="" />
    </javaPathConstants>
    <javaPathConstants typeName="com.creditonline.StylesheetConstants">
     <nodes root="/stylesheet" prefix="" />
    </javaPathConstants>
   </ebxbnd:binding>
  </xs:appinfo>
 </xs:annotation>
 ...
```

```
</xs:schema>
```

Java constants can be defined for XML schema paths. To do so, generate one or more interfaces from a schema node, including the root node `/`. The example generates two Java path constant interfaces, one from the node `/rules` and the other from the node `/stylesheet` in the schema. Interface names are described by the element `javaPathConstants` with the attribute `typeName`. The associated node is described by the element `nodes` with the attribute `root`

CHAPTER 79

Tools for Java developers

EBX5 provides Java developers with tools to facilitate use of the EBX5 API, as well as integration with development environments.

This chapter contains the following topics:

79.1 Activating the development tools

To activate the development tools, run EBX5 in *development mode*. This is specified in the EBX5 main configuration file EBX5 run mode [p 359] using the property `backend.mode=development`.

79.2 Data model refresh tool

If you edit your data model directly as an XML Schema document without using the data-modeling tool provided by EBX5, you can refresh it without restarting the application server.

In the Administration area, select **Actions** > **Refresh updated data models** (or **Actions** > **Refresh all data models**).

Attention

Since the operation is critical regarding data consistency, refreshing the data models acquires a global exclusive lock on the repository. This means that most of the other operations (data access and update, validation, etc.) will wait until completion of the data model refresh.

79.3 Generating Java bindings

The Java types specified by Java bindings can be generated from a data set or data model, by selecting **Actions** > **Generate Java** in the navigation pane.

See also *Java bindings [p 505]*

79.4 **Path to a node**

The field 'Data path' is displayed in the documentation pane of a node. This field indicates the path to the node, which can be useful when writing XPath formulas.

> **Note**
>
> This field is always available to administrators.