



Documentation Produit

EBX5 Version 5.4.0 Fix 002

Table des matières

Guide utilisateur

Introduction

1. Notions clés.....	11
2. Interface utilisateur.....	15
3. Glossaire.....	19

Modèles de données

4. Introduction aux modèles de données.....	30
5. Utilisation de l'interface utilisateur de la section Modèles de Données.....	33

Implémentation des modèles de données

6. Création du modèle de données.....	37
7. Configuration du modèle de données.....	39
8. Modélisation de la structure des données.....	43
9. Propriétés des éléments du modèle de données.....	49
10. Contrôles sur les éléments du modèle de données.....	61
11. Actions sur les modèles de données existants.....	67

Publication et gestion de versions des modèles de données

12. Publication du modèle de données.....	71
13. Gestion des versions de modèles de données embarqués.....	73

Espaces de données

14. Introduction aux espaces de données.....	78
15. Actions principales.....	81
16. Image.....	85
17. Fusion d'un espace de données.....	89
18. Permissions.....	93

Jeux de données

19. Introduction aux jeux de données.....	98
20. Actions principales.....	101
21. Vues personnalisées.....	107
22. Héritage.....	111
23. Permissions.....	115

Modèles de workflow

24. Introduction aux modèles de workflow.....	118
25. Actions principales.....	121
26. Modélisation du workflow.....	125
27. Tâche utilisateur.....	131
28. Publication d'un modèle de workflow.....	135

Workflows de données

29. Introduction aux workflows de données.....	138
30. Utilisation de l'interface utilisateur de la section Workflow de données.....	141
31. Bons de travail.....	145

Gestion de workflows de données

32. Lancement et monitoring de workflows de données.....	149
33. Administration de workflows de données.....	151

Services de données

34. Introduction aux services de données.....	156
35. Actions principales.....	159

Manuel de référence (en anglais)

Intégration

36. Overview of integration and extension.....	165
37. Using EBX5 as a web component.....	169
38. User interface services (UI services).....	175
39. Built-in UI services.....	185
40. Data services.....	197

Services d'import et export

41. XML import and export.....	223
42. CSV import and export.....	227
43. Supported XPath syntax.....	231

Divers

44. Permissions.....	238
45. Labeling and localization.....	249
46. Extending EBX5 internationalization.....	253
47. Inheritance and value resolution.....	255
48. Criteria Editor.....	261
49. Performance guidelines.....	263

Persistance

50. Overview of persistence.....	272
51. Relational mode.....	277
52. History.....	283
53. Replication.....	291
54. Data model evolutions.....	297

Guide d'administration (en anglais)

Installation & configuration

55. Supported environments.....	303
56. Java EE deployment.....	307
57. EBX5 main configuration file.....	321
58. Installing a repository using the Configuration Assistant.....	335
59. Deploying and registering EBX5 add-ons.....	339

Technical administration

60. Repository administration.....	342
61. Front end administration.....	351
62. Users and roles directory.....	363
63. Audit Trail.....	367
64. Data model administration.....	371
65. Data workflow administration.....	373
66. Task scheduler.....	377

Distributed Data Delivery (D3)

67. Introduction to D3.....	384
68. D3 broadcasts and delivery data spaces.....	389
69. D3 administration.....	393

Guide du développeur (en anglais)

70. Notes to developers.....	400
------------------------------	---------------------

Model design

71. Introduction.....	404
72. Packaging EBX5 modules.....	407
73. Types.....	411
74. Tables, filters and selection nodes.....	421
75. Constraints, triggers and functions.....	429
76. Labels and messages.....	449
77. Additional properties.....	455

Java reference

78. Mapping to Java.....	462
79. Tools for Java developers.....	469

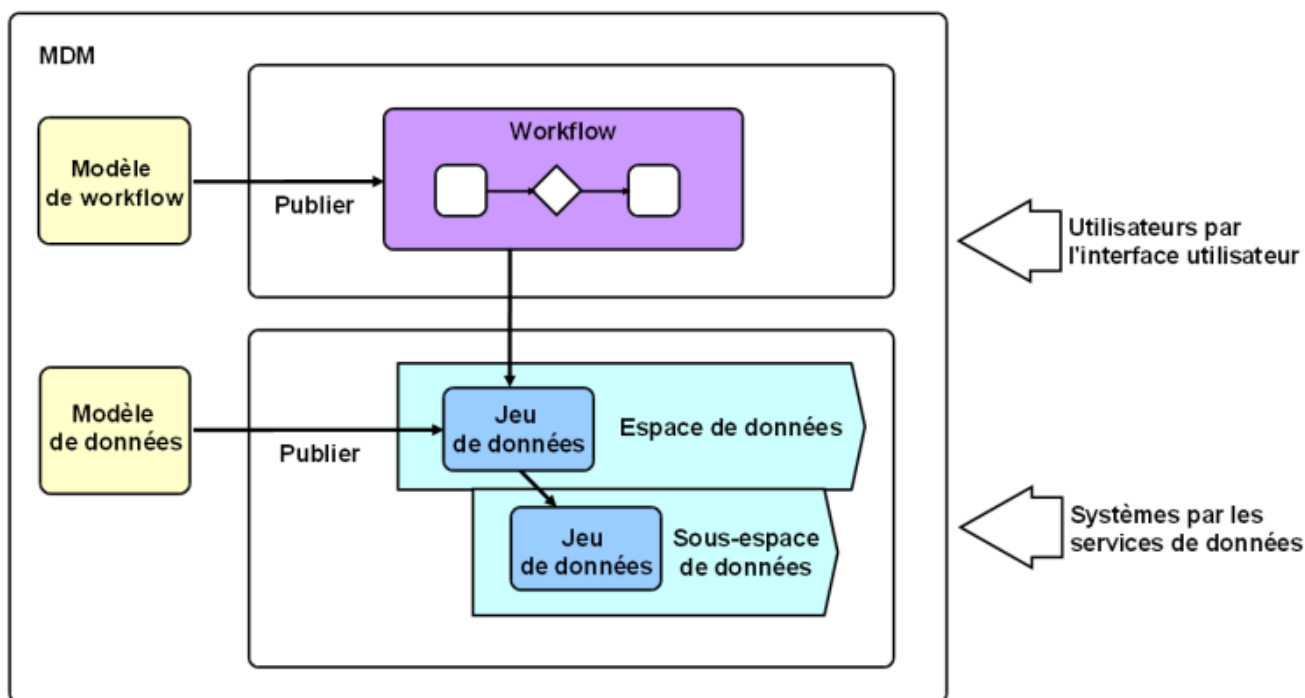
Guide utilisateur

Introduction

CHAPITRE 1

Notions clés

Concepts et outils associés



La gestion des données de référence (plus connue sous l'acronyme MDM pour Master Data Management) est un moyen de mieux modéliser, gérer et ultimement gouverner vos données partagées. La duplication des données au sein de plusieurs systèmes informatiques et son partage par des équipes professionnelles multiples rend critique le besoin d'avoir une version unique et une gouvernance de vos données de référence.

Avec EBX5, les utilisateurs métier et les informaticiens peuvent collaborer sur une seule et unique solution unifiée de manière à concevoir des modèles de données et gérer le contenu des données de référence.

EBX5 est un logiciel de gestion des données de référence qui permet de modéliser tout type de données de référence et d'y appliquer une gouvernance grâce à des outils avancés comme le workflow collaboratif, le contrôle de l'édition des données, la gestion hiérarchique des données, le contrôle de version, et la sécurité.

Un projet MDM sur EBX5 commence par la création d'un modèle de données. C'est là que vous définissez les tables, les champs, les liens et les règles métier permettant de décrire vos données de référence. De bons exemples sont les catalogues de produits, les hiérarchies financières, la liste des fournisseurs ou simplement les tables de référence.

Votre modèle de données peut ensuite être publié en tant que jeu de données, stockant le contenu actuel de vos données de référence. Un jeu de données se trouve dans un espace de données. Un espace de données est un conteneur, qui permet d'isoler des mises à jour ; cela est très utile si vous avez besoin de travailler sur plusieurs versions parallèles de vos données, de réaliser une analyse d'impact ou de travailler en "espaces de démonstration".

Une fois que tout est prêt à l'emploi, vous pouvez définir des processus de gestion des données, que vous allez exprimer sous forme de modèles de workflow dans EBX5. Ces modèles détaillent les tâches à accomplir ainsi que les responsabilités associées. Après publication de ces modèles, il sont disponibles à l'usage sous forme de workflows.

Les workflows sont de grande valeur si vous avez besoin de réaliser une gestion contrôlée du changement ou une validation des données pas à pas impliquant des utilisateurs multiples. Une fois démarrés, des notifications sont envoyées aux utilisateurs sur le nombre de bons de travail mis à leur disposition dans un contexte de travail collaboratif.

Les services de données aide à intégrer EBX5 à des systèmes tiers (middlewares), en leur permettant d'accéder aux données, ou de gérer des espaces de données et/ou des workflows.

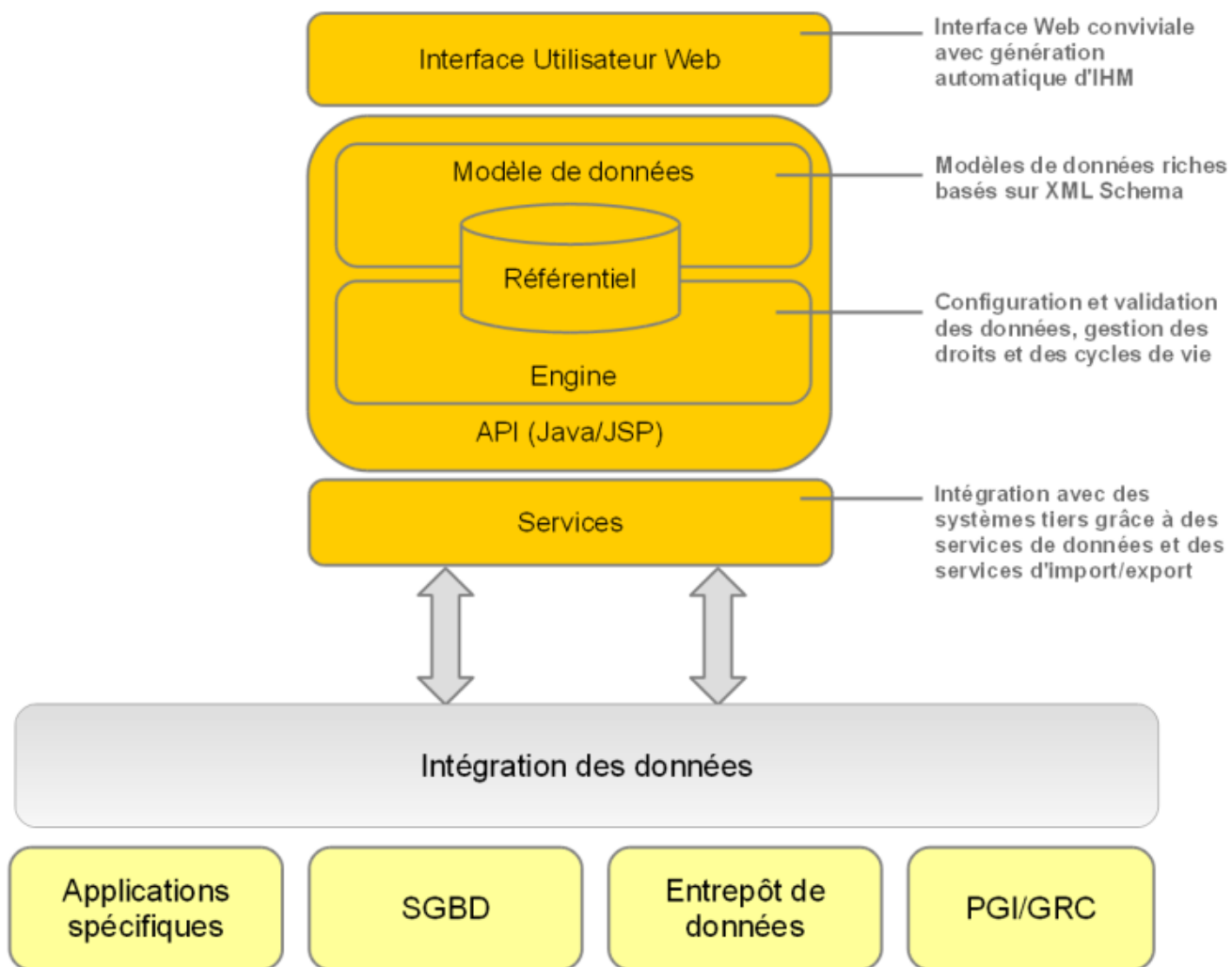
Les mots-clés à comprendre sont :

- [Modèle de données](#)
- [Jeu de données](#)
- [Espace de données](#)
- [Modèle de workflow](#)
- [Workflow de données](#)
- [Service de données](#)

Des définitions détaillées peuvent être trouvées dans notre [glossaire](#).

Architecture

Le schéma suivant présente l'architecture de EBX5.



CHAPITRE 2

Interface utilisateur

Comment vous y retrouver dans EBX5

L'interface de EBX5 est divisée en plusieurs régions générales, référencées dans la documentation en utilisant les termes suivants :

- **Entête** : le nom de l'utilisateur courant s'affiche dans cette zone, ainsi que la sélection de langue, un lien vers la documentation et un bouton pour fermer la session courante.
- **Barre de menu** : cette zone comprend toutes les fonctionnalités accessibles à l'utilisateur et lui permet de naviguer entre elles.
- **Panneau de navigation** : cette zone résume visuellement les diverses possibilités de navigation. Par exemple, sélectionner une table dans un jeu de données ou un bon de travail dans un workflow.
- **Espace de travail** : il s'agit d'une zone de travail. Par exemple, la table sélectionnée dans le panneau de navigation s'affiche dans l'espace de travail ou bien un bon de travail en cours s'y exécute.

Les sections fonctionnelles suivantes sont affichées dans l'interface selon les permissions de l'utilisateur actuel : *Données*, *Espace de données*, *Modélisation*, *Workflow de données*, *Services de données*, et *Administration*.



Où trouver de l'aide sur EBX5

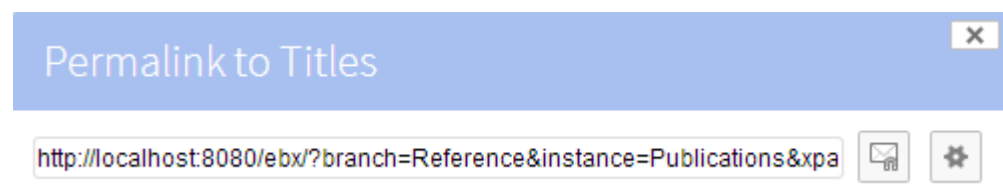
A part de la documentation de produit complète, l'aide est accessible de différentes façons dans l'interface.

Aide contextuelle

Lorsque la souris survole un élément ayant sa propre aide, un point d'interrogation apparaît. Quand vous cliquez sur cet élément, un petit panneau apparaît avec les informations relatives à cet élément.



De plus, lorsque cela est possible, un bouton est disponible à droite du libellé permettant de récupérer un permalien vers l'élément.



CHAPITRE 3

Glossaire

Gouvernance

référentiel

Entité de stockage côté serveur contenant toutes les données gérées par EBX5. Le référentiel est organisé en espaces de données.

Voir [espace de données](#).

profil

Terme générique qui définit soit un utilisateur, soit un rôle. Les profils sont utilisés pour définir des règles de permission et des workflows de données.

Voir [utilisateur](#), [rôle](#).

Java API apparentée **Profile** [Profile]^{API}.

utilisateur

Entité créée dans le référentiel afin de permettre à des personnes physiques ou des systèmes externes de s'authentifier et accéder à EBX5. Un utilisateur peut être affecté à un ou plusieurs rôles, et possède diverses informations de compte tels que nom, prénom, login, email etc.

Voir [annuaire des utilisateurs et des rôles](#), [profil](#).

Concept apparenté [User and roles directory](#).

Java API apparentée **UserReference** [UserReference]^{API}.

rôle

Classification d'utilisateur, utilisée pour les règles de permission et les workflows de données, pouvant être affectée à des utilisateurs. Chaque utilisateur peut appartenir à plusieurs rôles.

Dès qu'un profil de type rôle est configuré dans EBX5, le comportement résultant de cette configuration s'applique à tous les utilisateurs membres de ce rôle. Par exemple, dans un modèle de workflow, un rôle peut être configuré de façon à définir à qui les bons de travail sont proposés. Lors de l'exécution du workflow de données, ce bon de travail pourra être proposé à tous les utilisateurs membres de ce rôle.

Voir [annuaire des utilisateurs et des rôles](#), [profil](#).

Concept apparenté [User and roles directory](#).

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

administrateur

Rôle prédéfini permettant d'accéder à l'administration technique et à la configuration de EBX5.

annuaire des utilisateurs et des rôles

Annuaire définissant les méthodes disponibles pour l'authentification d'accès du référentiel, ainsi que tous les rôles disponibles et les utilisateurs autorisés à accéder au référentiel, avec leur appartenance aux rôles.

Voir [utilisateur](#), [rôle](#).

Concept apparenté [User and roles directory](#).

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

session utilisateur

Contexte d'accès au référentiel associé à un utilisateur (utilisateur ayant été authentifié par rapport à l'annuaire des utilisateurs et des rôles).

Concept apparenté [User and roles directory](#).

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

Modélisation de données

Section de la documentation [Modèles de données](#)

modèle de données


Définition structurée des données à gérer dans le référentiel EBX5. Un modèle de données comprend des descriptions détaillées de toutes les données incluses, en termes d'organisation, de types de données et de relations sémantiques. Le but d'un modèle de données est de définir la structure et les caractéristiques d'un jeu de données, qui est une instance d'un modèle de données qui contient les données gérées par le référentiel.

Voir [jeu de données](#).

Concept apparenté [Modèles de données](#).

champ

Élément de base du modèle de données qui est défini par un nom et un type. Un champ peut être directement défini à la racine du modèle de données, ou en tant que colonne d'une table. Il est possible d'assigner des contraintes de base sur la valeur du champ, par exemple sur sa longueur, ainsi que des règles de validation plus complexes impliquant des calculs. La valeur du champ peut être automatiquement calculée à l'aide du mécanisme d'héritage de données ou de règles de calcul. Un champ peut être défini comme étant une liste agrégée en spécifiant une cardinalité maximum supérieure à 1. Chaque élément de la liste ainsi définie sera du même type que le champ initial. Les champs peuvent être organisés en groupes pour faciliter l'organisation du modèle de données.

Par défaut, les champs sont représentés par l'icône .

Voir [enregistrement](#), [groupe](#), [table \(modèle de données\)](#), [règle de validation](#), [héritage](#).

Concept apparentés [Propriétés des éléments de structure](#), [Contrôles sur les champs de données](#).

Java API apparentée **SchemaNode** [`SchemaNode`]^{API}.

clé primaire

Champ ou composition de plusieurs champs identifiant de manière unique un enregistrement dans une table.

Les clés primaires sont représentées par l'icône .

Concept apparenté [Tables](#).

clé étrangère

Champ ou composition de plusieurs champs référençant un enregistrement d'une autre table, via sa clé primaire.

Les clés étrangères sont représentées par l'icône .

Voir [clé primaire](#).

Concept apparenté [Clé étrangère](#).

table (modèle de données)

Élément du modèle de données composé de champs et/ou de groupes. Une table doit au moins être composée d'un champ défini comme étant une clé primaire. Une table peut être utilisée pour la création d'un type réutilisable, afin de créer d'autres éléments basés sur la structure de cette table.

Les tables sont représentées par l'icône .

Voir [enregistrement](#), [clé primaire](#), [type réutilisable](#).

groupe

Entité de classification utilisée pour organiser les données du modèle. Un groupe peut contenir des champs, d'autres groupes, et des tables. Si un groupe contient des tables, alors celui-ci ne pourra pas être inclus dans une autre table. Un groupe peut être utilisé pour la création d'un type réutilisable, afin de créer d'autres éléments basés sur la structure de ce groupe.

Les groupes sont représentés par l'icône .

Voir [type réutilisable](#).

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

type réutilisable

Définition d'un type simple ou complexe qui peut être partagée entre différents éléments d'un modèle.

règle de validation

Association d'une ou plusieurs règles de contrôle définies sur un champ ou une table. Toute donnée saisie ne respectant pas ces contrôles sera déclarée invalide, selon la sévérité associée à la règle de validation.

assistant de modélisation de données (DMA)

L'interface utilisateur inclut un outil d'aide à la modélisation des données. Il permet de définir la structure d'un modèle, de créer et éditer ses éléments, de configurer et publier le modèle.


Voir [Modèles de données](#).

Gestion des données

Section de la documentation [Jeux de données](#)

enregistrement

Ensemble de données identifié de manière unique par une clé primaire. Un enregistrement correspond à une ligne dans une table. Chaque enregistrement respecte la structure de données définie dans le modèle de données associé. C'est ce modèle de données qui indique les types et les cardinalités des champs qui composent l'enregistrement.

Les enregistrements sont représentés par l'icône .

Voir [table \(jeu de données\)](#), [clé primaire](#).

table (jeu de données)

Ensemble d'enregistrements (lignes) de même structure contenant des données. Chaque enregistrement est identifié de manière unique par sa clé primaire.

Les tables sont représentées par l'icône .

Voir [enregistrement](#), [clé primaire](#).

jeu de données

Instance d'un modèle de données qui contient les données. La structure et le comportement d'un jeu de données sont basés sur les définitions fournies par le modèle de données qu'il implémente. En fonction de son modèle de données, un jeu de données peut contenir des données sous la forme de tables, groupes et champs.

Voir [table \(jeu de données\)](#), [champ](#), [groupe](#), [vue personnalisée](#).

Les jeux de données sont représentés par l'icône .

Concept apparenté [Jeux de données](#).

héritage

Mécanisme par lequel une donnée d'une entité peut être valorisée par défaut à partir d'une autre entité. Dans EBX5, deux types d'héritage sont possibles : le premier entre deux jeux de données, le deuxième entre deux champs.

Lorsqu'il est activé, l'héritage entre jeux de données permet à un jeu de données enfant d'obtenir comme valeur par défaut les données du jeu de données parent. Cette fonctionnalité est utile par exemple lorsqu'on définit un modèle de données où il y a des données au niveau du parent qui doivent avoir la même valeur par défaut au niveau des jeux de données enfant. Il est possible de surcharger les valeurs héritées du parent dans les jeux de données enfants. Par défaut, l'héritage est désactivé. Il peut être activé lors de la définition du modèle de données.

L'héritage depuis le jeu de données parent est représenté par l'icône .

L'héritage de champ fonctionne de manière similaire, mais au grain d'un champ et non plus du jeu de données.

Les champs hérités sont représentés par l'icône .

Concept apparenté [Inheritance and value resolution](#).

vue personnalisée

Configuration d'affichage applicable sur une table. Une vue peut être créée pour un utilisateur ou un rôle, et permet de spécifier l'affichage des enregistrements en mode hiérarchique ou tabulaire, ainsi que des critères de filtrage et de tri.

Voir [vue hiérarchique](#).

Concept apparenté [Vues personnalisées](#).

vue hiérarchique

Vue personnalisée qui affiche les données d'une table sous forme d'arbre. Une vue hiérarchique peut être utile pour montrer les relations entre les données du modèle. Lors de la création d'une vue hiérarchique, une dimension doit être sélectionnée pour déterminer les relations à exploiter. Dans une vue hiérarchique, il est possible de naviguer à travers des relations récursives, ainsi qu'entre plusieurs tables en utilisant des clés étrangères.

Voir [vue personnalisée](#).

Concept apparenté [Hiérarchies](#).

Cycle de vie des données

Section de la documentation [Espaces de données](#)

espace de données

Conteneur de jeux de données. L'espace de données est utilisé pour isoler différentes versions de jeux de données ou pour les organiser. Des espaces de données enfants peuvent être créés à partir d'un espace de données. Un espace de données enfant est initialisé dans le même état que son parent au moment de sa création. Ultérieurement, l'enfant pourra être fusionné sur son parent. A tout moment, une comparaison avec d'autres espaces de données est possible.

Les espaces de données sont représentés par l'icône .

Voir [héritage](#), [référentiel](#), [fusion](#).

Concept apparenté [Data spaces](#).

espace de données de référence

Ancêtre commun de tous les autres espaces de données du référentiel. N'ayant pas de parent, cet espace de données ne peut pas être fusionné.

Voir [espace de données](#), [fusion](#), [référentiel](#).

fusion

Integration des changements réalisés dans un espace de données enfant depuis sa création dans son espace de données parent. L'espace de données enfant est fermé après que la fusion ait été réalisée avec succès. Pour effectuer cette fusion, un passage en revue des différences entre les deux espaces de données est requis afin de résoudre les éventuels conflits. En effet, des conflits peuvent survenir en cas de modifications sur les mêmes données tant sur l'enfant que le parent. Une décision doit être prise pour chacun de ces conflits afin de déterminer quelle modification doit prendre le pas sur l'autre.

Concept apparenté [Fusion](#).

image

Copie statique d'un espace de données qui capture son état et tout son contenu à un moment donné, afin d'être utilisée comme référence. Une image peut être consultée, exportée, et comparée à d'autres espaces de données, mais jamais modifiée directement.

Les images sont représentées par l'icône .

Concept apparenté [Image](#)

Historique

Section de la documentation [History](#)

historisation

Mécanisme qui peut être activé au niveau d'une table afin de suivre les modifications dans le référentiel. Deux vues d'historique sont disponibles quand l'historisation est activée : la vue historique de table et la vue historique des transactions. Dans toutes les vues d'historique, les fonctionnalités classiques des tables comme l'export, la comparaison, les filtres, sont disponibles.

L'activation de l'historique nécessite la configuration d'un profil d'historisation. L'historisation des tables n'est pas activée par défaut.

Voir [vue historique de table](#), [vue historique des transactions](#), [profil d'historisation](#).

profil d'historisation

Ensemble de préférences qui spécifient d'une part les espaces de données dont les modifications doivent être enregistrées dans l'historique de table, et d'autre part si les transactions doivent échouer quand l'historisation n'est pas disponible.

Voir [profil d'historisation](#).

vue historique de table

Vue contenant la trace de toutes les modifications effectuées sur une table donnée, notamment les créations, les mises à jour et les suppressions. Chaque entrée présente les informations transactionnelles comme l'estampillage et l'utilisateur ayant effectué l'action, ainsi que les données à la fin de la transaction. Ces informations peuvent aussi être consultées au niveau d'un enregistrement ou d'un jeu de données.

Référence technique apparentée [History](#).

vue historique des transactions

Vue présentant les données techniques et d'authentification des transactions au niveau du référentiel ou d'un espace de données. Etant donné qu'une transaction peut effectuer de multiples opérations/actions

et peut affecter/toucher plusieurs tables dans un ou plusieurs jeux de données, cette vue montre toutes les opérations qui se sont effectuées dans le périmètre en question pour chaque transaction.

Référence technique apparentée [History](#).

Modélisation de workflow

Section de la documentation [Modèles de workflow](#)

modèle de workflow

Définition de la succession des opérations à effectuer sur les données.


Un modèle de workflow de données décrit la totalité du parcours que doivent suivre les données pour être traitées, que ce soit en terme d'états ou d'actions associées à effectuer par des utilisateurs et des tâches automatiques.

Concept apparenté [Modèles de workflow](#).

tâche automatique

Tâche de workflow de données effectuée par une procédure automatique, sans intervention humaine.

Les tâches automatiques les plus communes sont la création d'espace de données, la fusion d'espace de données et la création d'image.


Les tâches automatiques sont représentées par l'icône .

Voir [modèle de workflow](#).

tâche utilisateur

Tâche de workflow qui est décomposée en un ou plusieurs bons de travail réalisés en parallèle par des utilisateurs (intervention humaine).

Les bons de travail sont proposés ou assignés aux utilisateurs, en fonction du modèle de workflow de données. L'avancement du workflow de données positionné sur une tâche utilisateur dépend de la satisfaction du critère de fin de tâche défini dans le modèle de workflow de données.

Les tâches utilisateur sont représentées par l'icône .

Voir [modèle de workflow](#).

condition de workflow

Etape de décision dans le workflow de données.

Une condition de workflow de données décrit le critère utilisé pour déterminer quelle sera la prochaine étape à exécuter.

Les conditions de workflow sont représentées par l'icône .

contexte des données

Ensemble de données qui peuvent être partagées entre les étapes pendant toute la durée de vie d'un workflow afin de garantir la communication entre les étapes.

Workflows de données

Section de la documentation [Workflows de données](#)

publication de workflow

Version particulière d'un modèle de workflow de données qui est mise à disposition des utilisateurs qui ont les permissions nécessaires afin de pouvoir exécuter des workflows.

workflow de données

Instance particulière d'un modèle de workflow qui exécute les étapes définies dans le modèle de workflow de données (les tâches utilisateur, les tâches automatiques et les conditions).

Voir [modèle de workflow](#).

Concept apparenté [Workflows de données](#).

corbeille

Liste des workflows de données publiés affichés en fonction des permissions de l'utilisateur. Les utilisateurs qui ont la permission de lancer des workflows peuvent le faire à partir de leur corbeille. Tous les bons de travail nécessitant une action de l'utilisateur sont affichés sous la publication de workflow associée dans la corbeille. De plus, si l'utilisateur est administrateur de workflows de données, il a la possibilité de voir leur état dans sa corbeille et ainsi d'intervenir, si nécessaire, sur les workflows qu'il supervise.

Voir [workflow de données](#)

bon de travail

Action unitaire d'une tâche utilisateur qui doit être réalisée par un utilisateur.

Les bons de travail alloués sont représentés par l'icône .

Voir [tâche utilisateur](#).

jeton

Repère de position d'un workflow qui indique quelle étape courante est actuellement exécutée par un workflow de données. Les jetons sont utilisés durant l'avancement d'un workflow de données, et sont uniquement visibles par les administrateurs du référentiel.

Services de données

Section de la documentation [Services de données](#)

service de données

Méthode par laquelle des systèmes externes peuvent interagir programmatiquement avec EBX5 et les données gérées dans le référentiel EBX5. Les services de données sont des services web standard, qui peuvent être utilisés pour accéder à une partie des fonctionnalités disponibles par le biais de l'interface utilisateur.

Concept apparenté [Services de données](#).

lignage

Mécanisme grâce auquel des profils de droit d'accès peuvent être mis en place pour des utilisateurs non humains, nommés services de données. Les profils de droit d'accès ainsi définis sont utilisés lors de l'accès aux données via des interfaces WSDL.

Voir [service de données](#).

Concept apparenté [Générer un WSDL pour un lignage](#).

Modèles de données

CHAPITRE 4

Introduction aux modèles de données

Présentation

La fonction d'un modèle de données

La première étape de toute gouvernance de données dans EBX5 est de développer un modèle de données. Le but d'un modèle de données est de définir la structure des données gérées dans le référentiel, en termes d'organisations, de types de données, et de relations sémantiques. Une fois qu'un modèle de données a été défini et publié, vous pouvez créer des jeux de données à partir de celui-ci.

Afin de définir un modèle de données dans le référentiel, vous devrez d'abord créer un nouveau modèle de données, puis définir sa structure et les propriétés de ses éléments (tables, champs et groupes). Lorsque vous aurez défini votre modèle de données, vous le publierez pour le rendre disponible et ainsi vous et les autres utilisateurs pourrez créer des jeux de données basés sur cette publication pour contenir les données qui sont gérées par le référentiel EBX5.

Concepts de base utilisés dans la modélisation des données

Une compréhension des termes suivants est nécessaire pour commencer la création de modèles de données :

- [champ](#)
- [clé primaire](#)
- [clé étrangère](#)
- [table](#)
- [groupe](#)
- [type réutilisable](#)
- [règle de validation](#)

Concepts apparentés :

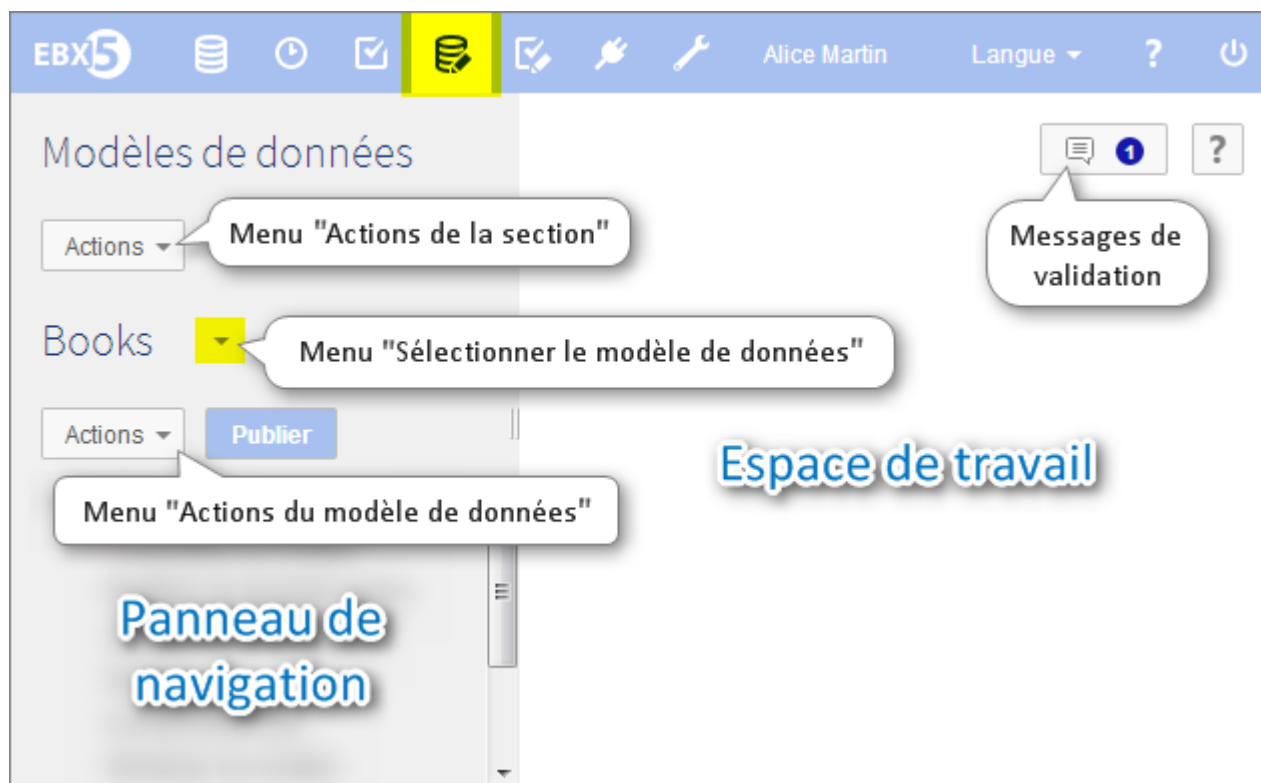
- [*Espaces de données*](#)
- [*Jeux de données*](#)

CHAPITRE 5

Utilisation de l'interface utilisateur de la section Modèles de Données

Navigation dans le Data Model Assistant

Les modèles de données peuvent être importés, édités, et publiés dans la section **Modélisation > Modèles de données**. Le Data Model Assistant de EBX5 aide le développement des modèles de données.



Le panneau de navigation est organisé selon les sections suivantes :

Configuration	La configuration technique du modèle de données.
Propriétés du modèle	Les propriétés techniques du modèle de données.
Modèle de données inclus	Les modèles de données inclus dans le modèle courant. Les types de données définis dans les modèles inclus peuvent être réutilisés dans le modèle de données courant.
Librairie de composants	Les composants Java utilisables dans le modèle de données.
Services	Les services disponibles dans le modèle de données.
Ajax components	Les composants Ajax disponibles dans le modèle de données.
Java bindings	Les propriétés des types Java générés depuis le modèle de données.
Répliquations	Les unités de répllication disponibles dans le modèle de données.
Structure de données	Structure du modèle de données. Définit les relations entre les éléments du modèle de données et permet d'accéder à la définition de chaque élément.
Types de données simples	Types simples réutilisables définis dans le modèle de données courant.
Types de données complexes	Types complexes réutilisables définis dans le modèle de données courant.
Types de données simples inclus	Types simples réutilisables définis dans un modèle de données inclus dans le modèle courant.
Types de données complexes inclus	Types complexes réutilisables définis dans un modèle de données inclus dans le modèle courant.

Icônes des éléments du modèle de données

 [champ](#)

 [clé primaire](#)

 [clé étrangère](#)

 [table](#)

[groupe](#)

Voir aussi :

- [*Modélisation de la structure des données*](#)
- [*Configuration du modèle de données*](#)
- [*Types réutilisables*](#)

CHAPITRE 6

Création du modèle de données

Création d'un modèle de données

Pour créer un modèle de données, utilisez le menu "[Sélectionner le modèle de données](#)" dans le panneau de navigation, cliquez sur le bouton **Créer** dans la pop-up, puis suivez les instructions de l'assistant de création de modèle de données.

Sélection d'un type de modèle de données

Si vous êtes un utilisateur avec le rôle "Administrator", vous devez sélectionner le type de modèle de données. Un utilisateur avec le rôle "Administrator" peut créer un modèle de données *sémantique* ou *relationnel*.

Modèles sémantiques

Les modèles sémantiques permettent l'utilisation de toutes les fonctionnalités de gestion des données, comme la gestion du cycle de vie en utilisant des espaces de données, fournies par EBX5. Par défaut, les modèles de données dans EBX5 sont sémantiques.

Modèles relationnels

Les modèles relationnels sont utilisés lorsque les tables du modèle doivent être accessibles à travers d'un système de gestion de base de données (SGBD). Le principal avantage des modèles relationnels est la possibilité d'interroger les tables avec des requêtes SQL externes. Cependant, les modèles relationnels perdent certaines fonctionnalités de gestion des données, par exemple l'héritage, les champs multi-valués, et la gestion de cycle de vie disponible par l'utilisation des espaces de données.

Note

Un modèle de données relationnel peut être utilisé par un seul jeu de données, et l'espace de données qui contient le jeu de données doit aussi être déclaré comme relationnel.

Voir aussi :

- [*Mode relational*](#)
- [*Espaces de données*](#)

CHAPITRE 7

Configuration du modèle de données

Informations associées au modèle de données

Pour visualiser et éditer les informations concernant le propriétaire et la documentation du modèle de données, sélectionnez "Informations" dans le menu "[Actions](#)" du modèle de données dans le panneau de navigation.

Nom unique	Le nom unique du modèle de données. Ce nom ne peut pas être modifié après la création du modèle.
Propriétaire	Spécifie le propriétaire du modèle de données, qui a le droit de modifier les informations du modèle et ses permissions.
Documentation localisée	Libellés et descriptions localisés pour le modèle de données.

Permissions

Pour définir les permissions d'accès du modèle de données, sélectionnez "Permissions" dans le menu "[Actions](#)" du modèle de données dans le panneau de navigation.

La configuration des permissions d'un modèle de données est identique aux options des permissions d'un jeu de données, expliquée dans la section [Permissions](#).

Propriétés du modèle de données

Dans le panneau de navigation, sous Configuration > Propriétés du modèle, vous pouvez accéder aux propriétés techniques suivantes :

Nom du module	Définit le module contenant les ressources qui seront utilisées par ce modèle de données. Celui-ci est également le module utilisé par la publication du modèle de données s'il est publié dans un module.
Chemin du module	Localisation physique du module sur le système de fichiers du serveur.
Chemin des sources	Les codes sources utilisés pour configurer les composants Java dans la "Librairie de composants". Si le chemin est relatif, il sera résolu à partir du "Chemin du module".
Mode de publication	<p>Les deux modes possibles sont la publication du modèle dans un document XML Schema dans un module ou en tant que modèle de données embarqué dans le référentiel EBX5. Les modèles de données embarqués permettent d'avoir des fonctionnalités additionnelles, comme la gestion des versions et la restauration des versions précédentes du modèle.</p> <p>Voir Modes de publication pour plus d'informations</p> <p>Chemin du modèle dans le module: Définit l'emplacement du document XML Schema pour la publication du modèle de données dans un module. Le chemin doit commencer avec "/".</p>
Héritage des données	<p>Spécifie si l'héritage des données entre jeux de données est activé pour ce modèle de données. L'héritage des données est désactivé par défaut.</p> <p>Voir héritage pour plus d'informations.</p>
Documentation	<p>Documentation du modèle de données définie à l'aide d'une classe Java. Cette classe Java définit les libellés et descriptions des éléments du modèle de données. Les libellés et descriptions définis par cette classe Java sont affichés, dans les jeux de données associés, en priorité par rapport aux libellés et descriptions définis localement par les éléments du modèle de données.</p> <p>Voir Dynamic labels and descriptions pour plus d'informations.</p>
Extensions spéciales	Permissions définies à l'aide de règles programmatiques.

Désactiver les contrôles de l'auto incrément Indique si le contrôle de la valeur d'un champ auto incrémenté par rapport à la valeur maximale trouvée dans la table en cours de mise à jour doit être désactivé.

Voir [Valeurs auto-incrémentées](#) pour plus d'informations.

Modèles de données inclus

Vous pouvez utiliser les types de données définis dans un autre modèle de données dans le modèle de données courant en ajoutant une entrée pour l'autre modèle de données dans la table Configuration > Modèles de données inclus.

En accédant l'enregistrement du modèle inclus depuis cette table, vous trouverez des informations techniques liées à ce modèle sous l'onglet **Information**. Car des erreurs de validation peuvent se produire dans un modèle de données inclus plus tard, par exemple à cause de ressources Java supprimés, cette vue vous donnera les informations sur les erreurs.

Seuls les modèles de données sans erreurs de validation qui ont été définis et publiés comme modèle "embarqué" ou dans un module peuvent être inclus.

Les noms des types de données doivent être unique pour tous les définitions de types de données définis en local et inclus. Autrement dit, les types de données inclus ne peuvent pas avoir des noms qui correspondent avec ceux des types de données définis dans le modèle de données courant ou un autre modèle de données inclus.

Voir aussi : [Including external data models](#)

Réplication des données vers tables relationnelles

Dans n'importe quel type de modèle de données, il est possible de définir des *unités de réplication* afin que les données dans le référentiel soient copiées dans des tables relationnelles dédiées. Ainsi, ces tables relationnelles permettent d'accéder directement aux données en utilisant des requêtes et des vues SQL.

Pour définir une unité de réplication en utilisant l'interface utilisateur, créez un nouvel enregistrement dans la table 'Réplifications' située dans la section configuration du modèle de données dans le panneau de navigation. Chaque unité de réplication concerne un jeu de données spécifique dans un espace de


EBX5 documentation > Guide utilisateur > Modèles de données > Implémentation des modèles de données > Configuration du modèle de données particulier. Une unité de réplication peut inclure plusieurs tables, tant qu'elles sont dans le même jeu de données. Une unité de réplication définit les informations suivantes :

Nom	Nom de l'unité de réplication. Ce nom identifie l'unité de réplication dans le modèle de données. Ce nom doit être unique dans le modèle de données.
Espace de données	Indique l'espace de données concerné par la réplication. Cet espace de données ne peut ni être une version ni être relationnel.
Jeu de données	Indique le jeu de données concerné par la réplication.
Mode de rafraichissement	<p>Définit le mode de synchronisation. Les différents modes de synchronisation sont les suivants :</p> <ul style="list-style-type: none">• Sur 'commit' : les données répliquées contenues dans la base de données sont toujours à jour par rapport à la table source. Chaque transaction de mise à jour de la table source produit les mises à jour correspondantes dans la table contenant les données répliquées dans la base de données.• Sur demande : les données répliquées contenues dans la base de données sont mises à jour uniquement lorsqu'une opération manuelle de rafraichissement est effectuée.
Tables	<p>Indique les tables du modèle de données qui doivent être répliquées dans la base de données.</p> <p>Chemin de la table: Indique le chemin de la table dans le modèle de données qui doit être répliquée dans la base de données.</p> <p>Nom dans la base de données: Indique le nom de la table dans la base de données qui contiendra les données répliquées. Ce nom doit être unique par rapport à toutes les unités de réplication.</p>

Voir aussi : [Replication](#)

CHAPITRE 8

Modélisation de la structure des données

Pour définir la structure du modèle de données, sélectionnez le modèle de données avec lequel vous voulez travailler en utilisant le menu "[Sélectionner le modèle de données](#)"  dans le panneau de navigation.


La structure du modèle de données est accessible dans le panneau de navigation dans la section "Structure de données". Cette section permet de visualiser et de définir la structure des champs, groupes, et tables du modèle de données.

Actions et propriétés communes

Créer des éléments

Les éléments suivants peuvent être ajoutés à un modèle de données :

- champs
- groupes
- tables
- clés primaires
- clés étrangères

Ajoutez un de ces éléments sous un élément existant en cliquant sur la flèche  située à la droite de l'élément existant et en sélectionnant une option de création parmi les options présentées dans le menu. Suivez l'assistant de création afin de créer un élément.

Note


L'élément `root` est ajouté par défaut lors de la création d'un modèle de données. Cet élément représente la racine de la structure du modèle de données.

Noms, libellés, descriptions, et informations

Le nom de l'élément à créer est obligatoire. Ce nom doit être unique au sein d'un même niveau dans la structure de données. En effet, sous un même groupe deux éléments ne peuvent avoir le même nom. Une fois l'élément créé son nom ne pourra plus être modifié.


Vous avez la possibilité de définir des libellés localisés qui seront affichés dans l'interface utilisateur au lieu du nom unique de l'élément. Il est aussi possible de définir une description localisée de l'élément. Contrairement au nom de l'élément, les libellés et descriptions sont modifiables après la création de l'élément. Selon la préférence de langue de chaque utilisateur, EBX5 affichera le libellé et la description localisé de l'élément.

Supprimer des éléments

Tous les éléments sauf le noeud `root` peuvent être supprimés de la structure de données en utilisant la flèche  située à la droite de l'élément à supprimer.

Si un groupe ou une table n'utilisant pas un type réutilisable est supprimé, la suppression est effectuée récursivement sur tous les éléments situés sous le groupe ou la table.

Dupliquer des éléments existants

Pour dupliquer un élément, cliquez sur la flèche  située à la droite de l'élément à dupliquer. Il est nécessaire de spécifier le nom de l'élément dupliqué. Ce nom doit être unique au sein d'un même niveau dans la structure de données. Toutes les propriétés de l'élément source sont dupliquées.

L'élément dupliqué est rajouté dans le modèle de données au même niveau que l'élément d'origine, et en dernière position. Si vous dupliquez un élément qui contient autres éléments, tous les sous éléments sont dupliqués avec leurs noms originaux.

Note

Si vous dupliquez un champ appartenant à une clé primaire, les propriétés du champ sont dupliquées, mais le nouveau champ n'est pas ajouté à la clé primaire de la table parente.

Déplacer des éléments

Il est possible de déplacer un élément en utilisant les flèches orientées vers le haut et le bas situées à côté de l'élément à déplacer.

Note

Le déplacement d'un élément est uniquement possible au sein d'un même niveau dans la structure de données du modèle.



Types réutilisables


Les types réutilisables sont des éléments partagés qui sont créés une fois, puis réutilisés par différents éléments du modèle de données.

Note

En modifiant la définition d'un type réutilisable, vous modifiez la structure de tous les éléments basés sur ce type réutilisable. L'arborescence "Structure de données" affiche la structure en lecture seule d'un groupe ou d'une table qui utilise un type réutilisable. Pour éditer la structure du type réutilisable associé, vous devez accéder au type dans la section "Types de données simples" ou "Types de données complexes".

Définition d'un type réutilisable

En utilisant le menu avec la flèche  associé aux sections "Types de données simples" ou "Types de données complexes" dans le panneau de navigation, vous pouvez créer des types simples et des types complexes réutilisables. Vous avez aussi la possibilité de créer un type réutilisable lors de la création d'un élément complexe (table ou groupe). Dans ce cas, après la création de l'élément, le type réutilisable sera disponible pour la création d'autres éléments avec la même structure et les mêmes propriétés. Vous pouvez également convertir les tables et groupes existants en types réutilisables en utilisant le menu avec la flèche  situé à côté de l'élément à convertir.

Il est possible de visualiser les éléments du modèle utilisant un type réutilisable en éditant ce type et en cliquant sur le lien "Références vers ce type". Ce lien affiche une table listant tous les éléments utilisant ce type. Si le type est utilisé par aucun élément, vous pouvez le supprimer en sélectionnant "Supprimer type" en utilisant le menu avec la flèche  situé à droite du type à supprimer.

Utilisation d'un type réutilisable

Les structures et les propriétés de nouveaux éléments peuvent être définies par des types réutilisables en sélectionnant un type réutilisable lors de la création d'un élément. L'élément créé utilisera la structure et les propriétés du type réutilisable.

Inclusion des types de données définis dans autres modèles de données

Les types réutilisables peuvent aussi être partagé entre plusieurs modèles de données. En configurant l'inclusion d'un modèle de données externe, vous pouvez utiliser les types de données inclus pour créer des nouveaux éléments dans la structure de données de la même façon que les types réutilisables définis en local.

Note

Car les noms de types de données doivent être unique pour tous les types définis en local et inclus, vous ne pouvez pas créer un nouveau type réutilisable qui a le même nom qu'un type de données dans un modèle de données inclus. Egalement, ce n'est pas possible d'inclure un modèle de données externe qui définit un type de données avec le même nom qu'un type réutilisable défini en local ou dans un autre modèle de données inclus.

Les types de données inclus apparaît dans les sections "Types de données simples inclus" et "Types de données complexes inclus" dans le panneau de navigation. Vous pouvez consulter les détails de ces types réutilisables, mais il ne peuvent être édités que dans leurs modèles de données d'origine.

Voir [Modèles de données inclus](#) pour plus d'informations.

Détails de la création des éléments du modèle de données

Création de champs

Quand vous créez un champ, vous êtes obligé de sélectionner un type de données. Ce type définira le type de données associé aux valeurs saisies dans un jeu de données basé sur ce modèle. Le type de données du champ ne peut pas être modifié après la création du champ.

Création de tables

Lors la création d'une nouvelle table, vous avez la possibilité de créer un type réutilisable, basé sur la définition de cette table. Cela permet de réutiliser la même structure de table à différents endroits du modèle de données. Vous pouvez également utiliser un type réutilisable existant pour la définition de la nouvelle table. Voir [Types réutilisables](#) pour plus d'informations.

A la fin de l'assistant de création de table, vous pouvez créer directement une clé primaire qui sera associée à la nouvelle table. Chaque table nécessite la désignation d'au moins un champ clé primaire. Si vous décidez de ne pas créer un champ clé primaire lors de la création de la table, il sera possible de créer une clé primaire ultérieurement à partir du menu d'actions disponible sur la table dans la section "Structure de données " du panneau de navigation.


Création de groupes

Lors de la création d'un groupe, vous avez la possibilité de créer un type réutilisable, ou d'utiliser un type réutilisable existant. Voir [Types réutilisables](#) pour plus d'informations.

Création de clés primaires

Une clé primaire est nécessaire pour chaque table. Après la création d'une table, vous avez la possibilité de créer directement des champs appartenant à la clé primaire de la table à partir de l'assistant de création de tables.

Il est toutefois possible de créer une clé primaire à n'importe quel moment, à partir du menu d'actions disponible sur la table dans la section "Structure de données " du panneau de navigation.

Il est aussi possible d'ajouter un champ existant à la définition de la clé primaire en utilisant le menu avec la flèche  situé à droite de ce champ.

Création ou définition de clés étrangères

Les champs associés à une clé étrangère ont le type de données "Chaîne de caractères". Vous pouvez créer une clé étrangère vers une table située dans le même modèle directement depuis la structure de données, ou vous pouvez définir directement les propriétés d'une clé étrangère en éditant un champ de type "Chaîne de caractères".

Pour convertir un champ existant de type "Chaîne de caractères" en clé étrangère, activez la propriété "Contrainte de clé étrangère" dans les "Contrôles avancés" du champ et définissez les propriétés associées.

Création d'un catalogue d'attributs utilisateurs


Un catalogue d'attributs utilisateurs est nécessaire pour créer des champs de type "attribut utilisateur". Lorsque vous créez un nouveau catalogue d'attributs utilisateurs, vous devez définir son nom. Une fois créé, ce catalogue pourra être utilisé lors de la création de champs de type "attribut utilisateur".

Création d'un attribut utilisateur

Lorsque vous créez un attribut utilisateur, vous devez spécifier un catalogue existant de d'attributs utilisateurs. Ce catalogue contient les attributs qui seront associés avec le nouvel attribut utilisateur. Après la création de ce nouvel attribut, le catalogue référencé pourra être modifié en modifiant la propriété "Chemin du catalogue UDA" dans la section "Propriétés avancées".

Modification des éléments existants

Suppression d'un champ de la clé primaire

Tous champs appartenant à la clé primaire peut être supprimé de la clé primaire d'une table en utilisant le menu avec la flèche  situé à droite du champ. Le champ est supprimé de la définition de la clé primaire mais n'est pas supprimé de la structure de données.

Voir [clé primaire](#) dans le glossaire.

CHAPITRE 9

Propriétés des éléments du modèle de données

Après la création d'un élément, vous pouvez définir des propriétés supplémentaires pour compléter sa définition.

Voir aussi : [*Contrôles sur les éléments du modèle de données*](#)

Propriétés basiques des éléments

Propriétés basiques communes

Les propriétés basiques suivantes sont disponibles pour plusieurs types d'éléments :

Information	Informations supplémentaires non internationalisées associées à l'élément.
Nombre minimum de valeurs	<p>Nombre minimum de valeurs pour cet élément.</p> <p>Les clés primaires ne pouvant être multi-valuées, cette propriété doit être égale à "1" ou être "non définie". Pour les éléments de type "Noeud de sélection" le nombre minimum de valeurs est automatiquement défini à "0".</p>
Nombre maximum de valeurs	<p>Nombre maximum de valeurs pour cet élément. Si cette propriété est supérieure à "1", l'élément est considéré comme multi-valué.</p> <p>Les clés primaires ne pouvant être multi-valuées, cette propriété doit être égale à "1" ou être "non définie".</p> <p>Pour une table, le nombre maximum d'éléments est défini à "unbounded" par défaut lors de sa création.</p> <p>Pour les éléments de type "Noeud de sélection", le nombre maximum d'éléments est défini à "0" par défaut lors de la définition des propriétés du noeud de sélection.</p>
Règles de validation	<p>Cette propriété est disponible pour les champs situés dans une table, sauf pour les champs de type <code>Mot de passe</code>, les types réutilisables, les champs dans les types complexes réutilisables, et les noeuds de sélection. Cette propriété est utilisée pour définir des règles de validation riches et complexes avec l'aide d'un éditeur de prédicats XPath 1.0.</p> <p>Une règle de validation peut être utile lorsque la validation de la valeur dépend de critères complexes ou des valeurs des autres champs.</p> <p>En utilisant l'assistant associé, vous pouvez définir des libellés localisés pour la règle de validation, ainsi qu'un message localisé avec sévérité qui sera affiché si le critère n'est pas satisfait.</p> <p>Si une règle de validation est définie sur un champ de type table alors cette règle sera considérée comme une "contrainte</p>

sur table" et sera exécutée sur chaque enregistrement de la table. Voir [Contraintes sur table](#) pour plus d'informations.

Propriétés basiques des champs

Les propriétés basiques suivantes sont spécifiques aux champs simples :

Valeur par défaut	<p>Définit une valeur par défaut pour ce champ. Cette valeur sera insérée automatiquement dans le champ de saisie dans les formulaires de création de nouveaux enregistrements. Le type de la valeur par défaut doit être en accord avec le type du champ courant.</p> <p>Voir Valeur par défaut pour plus d'informations.</p>
Message d'erreur de conversion	<p>Messages d'erreur internationalisés affichés aux utilisateurs lors de la saisie d'une valeur qui n'est pas en accord avec le type du champ courant.</p>
Règle de calcul	<p>Cette propriété est disponible pour les champs dans les tables sauf pour les types réutilisables. Définit une règle de calcul pour la valeur du champ avec l'aide d'un éditeur de prédicats XPath 1.0.</p> <p>Une règle de calcul peut être utile lorsqu'une valeur dépend d'autres valeurs dans le même enregistrement, mais qu'un calcul programmatique n'est pas nécessaire.</p> <p>Les limitations suivantes existent pour les règles de calcul :</p> <ul style="list-style-type: none">• Les règles de calcul peuvent être utilisées seulement avec les champs simples dans une table.• Les règles de calcul ne peuvent pas être définies sur les champs du type <code>OResource</code> ou <code>Password</code>.• Les règles de calcul ne peuvent pas être définies sur les noeuds de sélection et les champs clés primaires.• Les règles de calcul ne peuvent pas être définies en accédant à l'élément depuis le rapport de validation.

Propriétés avancées des éléments

Propriétés avancées communes

Les propriétés avancées suivantes sont disponibles pour plusieurs types d'éléments :

UI bean	<p>Cette propriété est disponible pour tous les types d'éléments sauf les tables. Spécifie une classe Java permettant de personnaliser l'interface utilisateur associée à cet élément dans un jeu de données. Un UI bean peut afficher l'élément différemment et/ou modifier sa valeur en étendant la classe UIBeanEditor [<code>UIBeanEditor</code>]^{API} dans l'API Java.</p>
Transformation à l'export	<p>Cette propriété est disponible pour les champs et pour les groupes qui sont des noeuds terminaux. Elle spécifie une classe Java définissant des opérations de transformation à effectuer lors de la création d'une archive à partir d'un jeu de données associé. La transformation à effectuer porte sur la valeur de l'élément courant.</p> <p>Voir NodeDataTransformer [<code>NodeDataTransformer</code>]^{API} pour plus d'informations.</p>
Propriétés d'accès	<p>Définit le mode d'accès pour l'élément courant, à savoir : s'il peut être écrit et/ou lu.</p> <ul style="list-style-type: none">• 'Lecture & Ecriture' correspond au mode <code>RW</code> dans le XSD du modèle de données.• 'Lecture seule' correspond au mode <code>R-</code> dans le XSD du modèle de données.• 'Element hors d'un jeu de données' correspond au mode <code>CC</code> dans le XSD du modèle de données.• 'Noeud non terminal' correspond au mode <code>--</code> dans le XSD du modèle de données. <p>Voir Access properties pour plus d'informations.</p>
Affichage par défaut	<p>Cette propriété permet d'indiquer :</p> <ul style="list-style-type: none">• Si cet élément doit être affiché dans la vue par défaut d'un jeu de données. Lorsque la propriété "Caché" est sélectionnée cet élément ne sera pas affiché dans la vue par défaut d'un jeu de données. Si cette propriété n'est pas renseignée, alors cet élément sera affiché par défaut.

- Si cet élément doit être affiché dans l'outil de recherche textuelle d'un jeu de données. Lorsque la propriété "Caché dans une recherche textuelle" est sélectionnée cet élément ne sera pas affiché dans l'outil de recherche textuelle d'un jeu de données. Si cette propriété n'est pas renseignée, alors cet élément sera affiché par défaut dans l'outil de recherche textuelle d'un jeu de données.

Voir [Default view](#) pour plus d'informations.

Catégorie du noeud

Définit les catégories permettant de restreindre la visualisation des données. Un noeud de catégorie "Hidden" est caché par défaut dans un jeu de données.

Restriction : la définition de catégorie ne s'applique pas aux noeuds d'enregistrements de tables, à l'exception de la catégorie "Hidden".

Voir [Categories](#) pour plus d'informations.

Mode de comparaison

Définit un mode de comparaison pour l'élément. Ce mode permet de restreindre la comparaison des données associées à cet élément.

- "Défaut" implique que l'élément est pris en compte lors de la comparaison de données.
- "Ignoré" implique qu'aucune modification ne sera détectée lors de la comparaison de deux entités modifiées (jeux de données ou enregistrements).

Lors de la fusion de données les valeurs des éléments ignorés dans des jeux de données ou enregistrements modifiés ne sont pas fusionnées. Les valeurs contenues dans de nouveaux jeux de données ou enregistrements sont fusionnées.

Lors de l'importation d'une archive les valeurs des éléments ignorés dans des jeux de données ou enregistrements modifiés ne sont pas importées. Les valeurs contenues dans de nouveaux jeux de données ou enregistrements sont importées.

Voir [Comparison mode](#).

Propriétés avancées des champs

Les propriétés avancées suivantes sont spécifiques aux champs.

Contrôle de saisie sur valeur nulle

Implémente la propriété `osd:checkNullInput`. Cette propriété est utilisée pour activer et vérifier une contrainte sur valeur `null` lors de la saisie utilisateur.

Par défaut, pour permettre une saisie temporairement incomplète, la validation des éléments obligatoire n'est pas effectuée lors de la saisie utilisateur, mais pendant la validation du jeu de données. Si un élément obligatoire doit être vérifié immédiatement lors de la saisie utilisateur, cette propriété doit avoir la valeur "oui".

Note

Une valeur est considérée comme obligatoire si le "Nombre minimum de valeurs" est égale ou supérieure à "1". Pour les éléments terminaux, les valeurs obligatoires sont seulement vérifiées dans les jeux de données activés. Pour les éléments non terminaux, les valeurs sont vérifiées indépendamment de l'état d'activation du jeu de données.

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

UI bean

Voir [Propriétés avancées communes](#).

Fonction (valeur calculée)

Cette propriété est disponible pour les champs qui ne sont pas des clés primaires. Elle spécifie une classe Java permettant de calculer la valeur de ce champ programmatiquement. Peut être utile si la valeur de ce champ dépend d'autres valeurs dans le référentiel, ou si le calcul de la valeur doit récupérer des données depuis un système externe.

Une fonction peut être créée en implémentant l'interface **ValueFunction** `[ValueFunction]API`.

Transformation à l'export

Voir [Propriétés avancées communes](#).

Propriétés d'accès

Voir [Propriétés avancées communes](#).

Noeud de sélection

Cette propriété est disponible uniquement pour les champs de type "chaîne de caractères". Elle définit une sélection d'enregistrements en utilisant une expression XPath sur une table dans un jeu de données. Peut être utile pour associer des enregistrements qui sont liés au champ courant. Les enregistrements qui répondent au critère seront affichés quand l'utilisateur clique sur le lien généré. Lorsque cette propriété est renseignée les propriétés "Nombre minimum de valeurs" et "Nombre maximum de valeurs" sont automatiquement définies à "0".

Voir [Noeuds de sélection](#) pour plus d'informations.

Auto-incrément

Cette propriété est disponible uniquement pour les champs de type "entier" ou "décimal" contenus dans une table. Si elle est activée, la valeur du champ est calculée automatiquement lors de la création d'un nouvel enregistrement. Peut être utile pour les clés primaires, car l'auto-incrément génère un identifiant unique pour chaque enregistrement. Deux attributs peuvent être spécifiés :

Valeur de démarrage	Valeur initiale de l'auto-incrément. Si aucune valeur n'est définie, la valeur par défaut est "1".
Pas de l'incrément	La valeur ajoutée à la valeur précédente de l'auto-incrément. Si aucune valeur n'est définie, la valeur par défaut est "1".
Désactiver les contrôles de l'auto incrément	Indique si le contrôle de la valeur d'un champ auto incrémenté par rapport à la valeur maximale trouvée dans la table en cours de mise à jour doit être désactivé.

Les valeurs auto-incrémentées ont le comportement suivant :

- Le calcul et l'allocation de la valeur de ce champ sont effectués dès lors qu'un nouvel enregistrement est inséré et que la valeur du champ est indéfinie.
- Aucune allocation ne peut être réalisée si l'insertion programmatique spécifie déjà une valeur non `null`. Par conséquent, cette allocation n'est pas effectuée à l'insertion d'un enregistrement en mode occultation ou réécriture.
- Si une archive importée spécifie cette valeur, alors elle est préservée.
- Une valeur nouvellement allouée est, autant que possible, unique au sein du référentiel.

Plus précisément, le caractère unique de l'allocation s'étend sur tous les jeux de données basés sur le modèle de données, et également sur tous les espaces de données. Ce dernier cas de figure permet de fusionner un espace de données à son parent avec une garantie raisonnable qu'il n'y aura pas de conflits lorsque la valeur auto-incrémentée fait partie des enregistrements d'une clé primaire.

Ce principe a une limitation très spécifique : quand une opération majeure de mise à jour spécifiant des valeurs est réalisée en simultané d'une opération allouant une valeur au même champ, il est possible que cette dernière opération alloue une valeur qui sera fixée par la première opération (il n'y a pas de verrouillage entre plusieurs espaces de données).

Voir [Valeurs auto-incrémentées](#) pour plus d'informations.

Affichage par défaut

Voir [Propriétés avancées communes](#).

Catégorie du noeud

Voir [Propriétés avancées communes](#).

Champ hérité

Définit une relation entre le champ courant et un champ dans une autre table afin de chercher sa valeur automatiquement.

Enregistrement source	Une clé étrangère ou une séquence de clés étrangères, séparées par des espaces, permettant de trouver l'enregistrement dont hérite le champ courant. Si cette propriété n'est pas renseignée, alors l'élément courant sera utilisé comme source d'héritage de champ.
Élément source	Chemin XPath définissant l'élément à hériter. L'élément source doit être terminal, se trouver dans "Enregistrement source", et son type doit être le même que le type de ce champ. Cette propriété est obligatoire pour l'héritage de champ.

Voir [héritage](#) dans le glossaire et la section [Inherited fields](#) pour plus d'informations.

Propriétés avancées des tables

Les propriétés avancées suivantes sont spécifiques aux tables.

Table

Clé primaire	<p>Une liste des champs composant la clé primaire de la table. Vous pouvez ajouter ou supprimer des champs de la clé primaire.</p> <p>Chaque champ de la clé est dénoté par un chemin XPath absolu qui débute sous la table.</p>
Index	<p>Définit la liste des champs à indexer dans la table. L'indexation accélère les accès à la table pour les requêtes impliquant les champs indexés. Il n'est pas possible de définir deux index portant exactement sur la même liste de champs.</p> <p>Nom de l'index: Nom unique pour cet index.</p> <p>Champs à indexer: Les champs à indexer, où chaque champ d'index est dénoté par un chemin XPath absolu qui débute sous la table.</p>
Profil d'historisation	<p>Spécifie si l'historisation est activée, et le niveau de garantie demandé. Les profils d'historisation peuvent être modifiés dans la section Administration > Historique et journaux.</p> <p>Voir Configuration de l'historique dans le référentiel pour plus d'informations.</p>
Libellé	<p>Définit les champs composant le libellé par défaut et les libellés localisés pour les enregistrements de la table.</p> <p>Peut aussi définir une classe Java pour assigner le libellé programmatiquement, ou définir le libellé dans une hiérarchie. Cette classe Java doit implémenter l'interface UILabelRenderer [UILabelRenderer]^{API} ou UILabelRendererForHierarchy [UILabelRendererForHierarchy]^{API} de l'API Java.</p>
Filtres spécifiques	<p>Définit des filtres pour afficher uniquement les enregistrements correspondant à certains critères.</p>
Présentation	<p>Spécifie la politique d'affichage par défaut des groupes contenus dans cette table. Si cette propriété n'est pas définie, alors la politique par défaut sera utilisée pour afficher les groupes.</p> <p>Voir Ergonomie & comportement pour plus d'informations.</p> <p>Présentation activée des groupes : spécifie un mode de présentation autorisé en plus de "Développé" et "Réduit", qui sont toujours disponibles. Les liens doivent être activés sur la</p>

table afin de définir spécifiquement le mode de présentation des groupes en tant que liens.

Présentation par défaut des groupes : spécifie la présentation par défaut des groupes contenus dans cette table. Si un groupe ne spécifie aucune vue par défaut, alors le mode de présentation par défaut défini par cette table sera utilisé. En fonction des performances du réseau et du navigateur, ajustez la manière d'afficher chaque groupe du formulaire. En ce qui concerne le poids de la page téléchargée, le mode lien est léger, les modes "Développé" et "Réduit" sont plus lourds.

Note : Quand les onglets sont activés dans une table, tous les groupes qui utiliseraient les liens dans les autres cas sont convertis automatiquement en mode "Réduit". Ceci est fait afin d'éviter les complexités d'affichage qui se posent quand les liens et les onglets sont dans la même interface utilisateur.

Contraintes d'unicité

Indique les champs dont les valeurs doivent être uniques dans la table.

Trigger à la mise à jour

Spécifie une classe Java définissant des méthodes qui sont automatiquement exécutées lorsque des opérations sont effectuées : création, mise à jour, suppression, etc.

Un trigger natif est inclus par défaut pour lancer les workflows de données.

Voir [Triggers](#) pour plus d'informations.

Propriétés d'accès

Voir [Propriétés avancées communes](#).

Affichage par défaut

Voir [Propriétés avancées communes](#).

Catégorie du noeud

Voir [Propriétés avancées communes](#).

Propriétés avancées des groupes

Les propriétés avancées suivantes sont spécifiques aux groupes.

Classe du conteneur de valeurs (JavaBean)

Définit une classe Java pour contenir les valeurs des éléments situés sous le groupe. La classe Java spécifiée doit être conforme à la spécification JavaBean. Cela signifie que chaque élément enfant de ce groupe doit correspondre à une propriété de la classe Java. Toutes les propriétés de la classe Java doivent avoir des méthodes d'accès (getters et setters).

UI bean

Voir [Propriétés avancées communes](#).

Transformation à l'export

Voir [Propriétés avancées communes](#).

Propriétés d'accès

Voir [Propriétés avancées communes](#).

Affichage par défaut

Visibilité	Voir Propriétés avancées communes .
Présentation	<p>Définit la manière dont ce groupe sera présenté dans un jeu de données. Si cette propriété n'est pas définie, alors la vue par défaut définie par la table parente sera utilisée. Les options "Onglet" et "Lien" sont disponibles uniquement si la table parente a activé ces options de présentation.</p> <p>Propriétés de l'onglet</p> <p>Position: Cet attribut spécifie la position de l'onglet par rapport à tous les onglets définis dans le modèle. Cette position est utilisée pour ordonnancer la liste des onglets au moment de l'affichage d'un formulaire. Si cette propriété n'est pas définie, alors l'onglet sera positionné selon la position du groupe dans le modèle de données.</p>

Catégorie du noeud

Voir [Propriétés avancées communes](#).

Concepts apparentés : [Contrôles sur les éléments du modèle de données](#)

CHAPITRE 10

Contrôles sur les éléments du modèle de données

Après la création d'un élément, vous pouvez définir des contrôles supplémentaires pour compléter sa définition.

Voir aussi : [*Propriétés des éléments du modèle de données*](#)

Contrôles simples

Il est possible d'établir des contrôles simples sur le contenu d'un champ en définissant des contrôles statiques. Les contrôles disponibles dépendent du type du champ.

Longueur fixe	Le nombre exact de caractères requis pour ce champ.
Longueur minimale	Nombre minimum de caractères requis pour ce champ.
Longueur maximum	Nombre maximum de caractères autorisé pour ce champ.
Pattern	Une expression régulière à laquelle la valeur du champ doit être conforme. Il est interdit de définir un pattern à la fois sur un champ et sur son type de donnée.
Partie décimale	Nombre maximum de chiffres autorisé dans la partie décimale de la valeur d'un champ de type décimal.
Nombre total de chiffres	Nombre maximum de chiffres composant la valeur d'un champ de type entier ou décimal.
Enumération	Définit une liste de valeurs possibles pour ce champ. Si une énumération est définie à la fois sur ce champ et sur son type de donnée alors une énumération représentant l'intersection entre ces deux énumérations sera utilisées dans les jeux de données associés au modèle de données.
Supérieur à [constante]	Définit la valeur minimale autorisée pour ce champ.
Inférieur à [constante]	Définit la valeur maximale autorisée pour ce champ.

Voir [Facettes XML schema supportées](#).

Contrôles avancés

Il est possible d'établir des contrôles avancés sur le contenu d'un élément en définissant des contrôles dynamiques et contextuels. Les contrôles disponibles pour un élément dépendent de sa nature (table, groupe, etc.) et de son type de données.

Voir aussi : [Dynamic constraints](#)

Contrainte de clé étrangère	
Table	Définit la table ciblée par la clé étrangère. Une clé étrangère référence une table dans le même jeu de données par défaut. Elle peut aussi référencer une table dans un autre jeu de données dans le même espace de données, ou un jeu de données dans un autre espace de données.
Mode	<p>Emplacement de la table ciblée par la clé étrangère.</p> <p>"Défaut" : le modèle courant.</p> <p>"Autre jeu de données" : un jeu de données qui se trouve dans le même espace de données.</p> <p>"Autre espace de données" : un jeu de données appartenant à un espace de données différent.</p>
Table référencée	Expression XPath indiquant l'emplacement de la table. Par exemple, /racine/MaTable.
Jeu de données référencé	Obligatoire si la table est dans un autre jeu de données. Le nom unique du jeu de données contenant la table référencée.
Espace de données référencé	Obligatoire si la table est dans un autre espace de données. Le nom unique de l'espace de données contenant la table référencée.
Libellé	<p>Définit les champs composant un libellé par défaut et des libellés localisés pour présenter les enregistrements de la table cible.</p> <p>Peut aussi spécifier une classe Java qui définit le libellé programmatiquement quand "Expression XPath" a la valeur "Non". Cette classe Java doit implémenter l'interface TableRefDisplay [TableRefDisplay]^{API} de l'API Java.</p>
Filtre	<p>Définit un filtre de clé étrangère en utilisant une expression XPath.</p> <p>Peut aussi spécifier une classe Java qui implémente l'interface TableRefFilter [TableRefFilter]^{API} de l'API Java.</p>
Supérieur à [variable]	Définit un champ qui spécifie la valeur minimale autorisée pour ce champ.
Inférieur à [variable]	Définit un champ qui spécifie la valeur maximale autorisée pour ce champ.

Longueur fixe [dynamique]		Définit un champ qui spécifie le nombre exact de caractères requis pour ce champ.
Longueur minimale [dynamique]		Définit un champ qui spécifie le nombre minimum de caractères requis.
Longueur maximale [dynamique]		Définit un champ qui spécifie le nombre maximum de caractères autorisés.
Valeurs exclues		Définit une liste de valeurs non autorisées pour ce champ
Plage exclue de valeurs		Définit une plage de valeurs non autorisées pour ce champ. Valeur maximale exclue : La valeur minimale non autorisée pour ce champ. Valeur minimale exclue : La valeur maximale non autorisée pour ce champ.
Contrainte (composant)	spécifique	Spécifie une ou plusieurs classes Java qui implémentent l'interface Constraint [Constraint] ^{API} de l'API Java. Voir Programmatic constraints pour plus d'informations.
Enumération (composant)	spécifique	Spécifie une classe Java pour définir une énumération. La classe doit définir une liste ordonnée de valeurs, en implémentant l'interface ConstraintEnumeration [ConstraintEnumeration] ^{API} de l'API Java.
Enumération alimentée par un autre noeud		Spécifie un champ qui définit les valeurs autorisées pour cette énumération. Le champ spécifié doit être une liste ou doit définir une énumération.

Messages de validation

Chaque contrainte excepté celles utilisant une classe Java peut définir des messages de validation. Il est possible d'associer une sévérité à ces messages de validation et ces messages peuvent être localisés en utilisant les propriétés suivantes :

Validation	Spécifie le message de validation et la sévérité associée à la contrainte.
Sévérité	Spécifie la sévérité de la contrainte. Les valeurs possibles sont 'Erreur', 'Avertissement', et 'Information'.
Message	Le message à afficher lorsque la valeur de ce champ dans un jeu de données ne respecte pas cette contrainte. Ce message peut être localisé.

Concepts apparentés : [*Propriétés des éléments du modèle de données*](#)

CHAPITRE 11

Actions sur les modèles de données existants

Lorsque votre modèle de données est créé, vous pouvez effectuer des actions qui sont disponibles dans le menu **"Actions" du modèle de données** dans le panneau de navigation.

Validation du modèle de données

Il est possible de valider un modèle de données en sélectionnant **"Actions" du modèle de données > Valider** dans le panneau de navigation. Les éventuels messages issus de la validation du modèle de données sont présentés dans un rapport. Depuis le rapport de validation, vous pouvez cliquer sur le bouton **Revalider** pour mettre à jour ce rapport, ou vous pouvez cliquer sur le bouton **Réinitialiser le rapport de validation** pour supprimer tous les messages de validation actuellement associés au modèle de données afin de pouvoir relancer une validation complète.

Voir [Validation](#) pour obtenir plus d'informations concernant la validation incrémentale de données.

Import et export de fichiers XML Schema Document (XSD)

EBX5 fournit des services intégrés pour importer et exporter des fichiers XML Schema Document (XSD). Les services d'import et d'export sont accessibles depuis le menu **"Actions" du modèle de données** dans le panneau de navigation. Un import ou export est toujours effectué sur l'intégralité du modèle de données. Lors d'un import, la structure du modèle de données courant est entièrement remplacée par le contenu du document XML Schema importé. Lors d'un export le modèle de données complet est exporté dans le document XML Schema cible.

Pour importer un fichier XSD, le fichier doit être valide et doit être conforme aux règles de validation du référentiel EBX5. Si le document déclare de ressources situées dans un module, le module doit être déclaré aussi dans la configuration du modèle de données. Si le module n'a pas été déclaré, vous ne pourrez pas importer le fichier XSD. Voir [Propriétés du modèle de données](#) pour plus d'informations sur la déclaration des modules.

Pour importer un modèle, sélectionnez *Importer XSD* dans le menu **Actions** situé dans le panneau de navigation du modèle de données.

Vous pouvez importer un document XML Schema (XSD) à partir du système de fichier local. Pour cela sélectionnez *Importer à partir d'un document local* :

- **Nom du document** : chemin du document XSD à importer dans le système de fichiers local.

Vous pouvez importer un document XML Schema (XSD) contenu dans le référentiel. Pour cela sélectionnez *Importer à partir du référentiel de modèles* :

- **Modèle** : nom du modèle de données à importer.

Vous pouvez aussi importer un modèle de données XSD dans un module. L'import d'un modèle de données XSD depuis un module utilise les propriétés suivantes :

Module	Module qui contient le modèle de données.
Chemin du module	Localisation physique du module sur le système de fichier du serveur.
Chemin des sources	<p>Les codes sources utilisés pour configurer les "Règles et objets métier". Si le chemin est relatif, il sera résolu à partir du "Chemin du module".</p> <p>Cette propriété est obligatoire si le modèle définit des éléments programmatique.</p>
Modèle	Le modèle de données à importer.

Note

Les fichiers XSD à importer doivent être encodés en "UTF-8". Les fichiers XSD exportés sont toujours encodés en "UTF-8".

Duplication d'un modèle de données

Pour dupliquer un modèle de données, sélectionnez "Dupliquer" dans le menu **Actions** du modèle. Vous devez donner un nom au nouveau modèle de données. Ce nom doit être unique dans le référentiel.

Suppression d'un modèle de données

Pour supprimer un modèle de données, sélectionner "Supprimer" dans le menu [Actions du modèle de données](#). Quand vous supprimez un modèle de données, toutes les publications associées au modèle restent disponibles. Si vous créez un nouveau modèle de données avec le même nom que celui que vous avez supprimé, le nouveau modèle de données sera reassocié avec toutes les publications existantes dans le référentiel. Au moment de la publication, vous aurez l'opportunité de confirmer le remplacement d'une publication existante.

Note

Seul un administrateur peut supprimer les publications d'un modèle de données dans la section "Administration".

Voir [Publication des modèles de données](#) pour plus d'informations sur le processus de publication.

CHAPITRE 12

Publication du modèle de données

A propos des publications

Chaque jeu de données dans le référentiel EBX5 basé sur un **modèle de données embarqué** est associé à une publication d'un modèle de données, et non directement au modèle de données défini dans **Data Model Assistant**. Une publication est créée la première fois que vous publiez un modèle de données en utilisant le bouton **Publier** dans le panneau de navigation. Il est possible à partir d'une publication de créer des jeux de données dont la structure sera basée sur la structure du modèle de données publié.

Note

Le bouton **Publier** est uniquement affiché pour les utilisateurs qui ont le droit de publier le modèle de données. Voir [Permissions du modèle de données](#) pour plus d'informations.

Les jeux de données étant basés sur des publications, toutes modifications que vous faites sur le modèle de données n'impacteront les jeux de données existants. Les jeux de données seront impactés uniquement lors d'une mise à jour d'une publication existante.

Modes de publication

Vous pouvez publier un modèle de données en mode "Embarqué" ou en mode "Dans un module". Le mode de publication "Embarqué" génère une publication qui est gérée et persistée dans le référentiel EBX5 et possède des fonctionnalités spécifiques dédiées à la gestion de version. Le mode de publication "Dans un module" crée un fichier XML Schema Document (XSD) à l'intérieur d'un module qui n'est pas géré par le référentiel EBX5.

Selon la configuration du modèle de données, EBX5 détermine automatiquement le processus de publication à utiliser quand vous cliquez sur le bouton **Publier** dans le panneau de navigation. Quand un modèle de données spécifie le mode de publication "Dans un module" ainsi qu'un fichier XSD à cibler, le processus de publication génère le fichier XSD dans le module défini dans la configuration.

Mode de publication "Embarqué"

La première fois que vous publiez un modèle de données en mode embarqué, une nouvelle publication avec le même nom que le modèle est automatiquement créée dans le référentiel. Si différentes publications ont été créées à partir du modèle vous devrez sélectionner la publication à mettre à jour.

Voir [Consultation et création de publications](#) pour plus d'informations sur l'utilisation de différentes publications.

Pendant le processus de publication, si le modèle de données a été déjà publié, vous avez l'opportunité de visualiser les différences structurelles introduites par la nouvelle publication dans une interface de comparaison.

Le processus de publication permet aussi de créer une image en lecture seule de l'état actuel du modèle de données. Cette image sera utile si un état précédent du modèle doit être restauré après plusieurs modifications et publications du modèle de données.

Note

Les images sont des archives statiques du modèle de données et ne doivent pas être confondues avec les *versions* du modèle de données, qui sont des éditions du modèle évoluant en parallèle. Voir [Gestion de versions du modèle de données embarqué](#) pour plus d'informations sur les versions des modèles de données.

Consultation et création des publications

Pour accéder aux publications existantes du modèle de données courant, sélectionnez "Gérer les publications" dans son menu ["Actions" du modèle de données](#) dans le panneau de navigation. Vous pourrez consulter les détails des publications et créer de nouvelles publications.

Dans certains cas, il faudra utiliser plusieurs publications du même modèle de données, afin de permettre à différents jeux de données d'être basés sur des états différents du modèle. L'utilisation de plusieurs publications doit être réalisée avec précaution. En effet, les utilisateurs devront sélectionner la publication à mettre à jour lors de la publication du modèle de données et doivent donc avoir connaissance de l'utilisation de ces différentes publications. La création d'une nouvelle publication est disponible uniquement aux utilisateurs ayant le rôle "Administrator".

Pour créer une nouvelle publication, sélectionnez "Gérer les publications" dans le menu ["Actions" du modèle de données](#) dans le panneau de navigation, puis cliquez sur le bouton **Créer une publication**. Le nom que vous donnez à la publication doit être unique dans le référentiel. Après sa création la nouvelle publication est vide et ne représente pas un modèle de données. Pour pouvoir être utilisée par des jeux de données la publication doit donc être mise à jour en publiant le modèle de données.

CHAPITRE 13

Gestion des versions de modèles de données

A propos des versions

Vous pouvez créer des *versions* pour les modèles de données afin d'avoir différents états du modèle qui évoluent en parallèle. Les versions ne doivent pas être confondues avec les images des modèles de données, qui sont prises au moment de publication, et sont sauvegardées uniquement pour consultation d'historique en lecture seule.

Accès aux versions

Pour voir les versions existantes de votre modèle de données, sélectionnez "Gérer les versions" dans le menu **"Actions" du modèle de données**.

Les versions existantes sont représentées dans une arborescence selon leurs relations parent-enfant. Chaque modèle de données a une version racine par défaut.

Actions sur les versions

Dans l'espace de travail, en utilisant le menu avec la flèche vers le bas ▼ à côté de chaque version, vous pouvez effectuer les actions suivantes :

Accéder à la version du modèle	Pour visualiser la version correspondante du modèle de données.
Créer une nouvelle version	Crée une nouvelle version basée sur le contenu de la version sélectionnée. La nouvelle version est créée comme enfant de la version sélectionnée mais leurs contenus évoluent indépendamment.
Définir en tant que version par défaut	Définit la version par défaut qui est sélectionnée lorsqu'un utilisateur accède au modèle de données.
Exporter une archive	<p>Exporte le modèle de données sélectionné dans une archive contenant le contenu de la version, ses permissions et ses informations. L'archive exportée est localisée dans le répertoire d'archives, qui est accessible aux administrateurs du référentiel.</p> <p>Voir Dossier archives pour plus d'informations sur le répertoire d'archives.</p>
Importer une archive	Importe le contenu d'une archive dans la version sélectionnée. L'archive à importer doit contenir un modèle de données avec le même nom que le modèle associé à cette version.

Une version peut être supprimée en cliquant le bouton **X** situé à la droite d'une version. Une version ne peut pas être supprimée si elle est liée à une publication ou si elle a des sous versions. La version racine du modèle de données ne peut pas être supprimée.

Deux versions du même modèle peuvent être comparées dans l'espace de travail en sélectionnant leurs cases à cocher, puis **"Actions" du modèle de données > Comparer les versions sélectionnées**. La vue de comparaison côte-à-côte affiche les différences structurelles entre les versions du modèle de données, avec la plus ancienne à la gauche et la plus récente à la droite.

Limitations connues sur la gestion de versions du modèle de données

- Il est impossible de fusionner deux versions d'un modèle de données.
- L'interface de comparaison n'affiche pas les mises à jours des champs, uniquement les ajouts et suppressions.
- Il n'est pas possible de gérer des versions de modèles de données publiés dans un module.
- Les ressources contenues dans un module utilisées par un modèle de données embarqué ne sont pas versionnées lorsqu'une version est créée. En effet, seul les références des ressources sont sauvegardées, et c'est la responsabilité des développeurs de s'assurer que les ressources référencées restent compatibles avec les différentes versions les utilisant.

Espaces de données

CHAPITRE 14

Introduction aux espaces de données

Un espace de données est un récipient de données, dont le contenu peut être mis à jour en toute isolation, c'est-à-dire sans affecter les données situées à l'extérieur de celui-ci. Après avoir décidé comme vous souhaitez organiser vos données, vous pouvez être amené à :

- créer des espaces de données (voir [création](#)),
- les étiqueter et en décrire le contenu et l'utilité (voir [information](#)),
- exporter ou importer un jeu de données dans l'espace de données, puis valider ce changement (voir [import d'archive](#) ou [export archive](#)),
- comparer le contenu de deux espaces de données (voir [actions](#)),
- prendre une image de l'espace de données avant d'y appliquer tout autre changement (voir [image](#)),
- appliquer les mêmes changements d'un espace de données fils à son parent (voir [fusion](#)),
- gérer les droits d'accès à un espace de données (voir [permissions](#)),
- fermer un espace de données, qui n'est plus requis (voir [fermeture](#)).

Un espace de données est toujours créé à partir d'un autre espace de données, à l'exception de l'espace de données de référence, qui est la racine de tous les autres espaces de données.

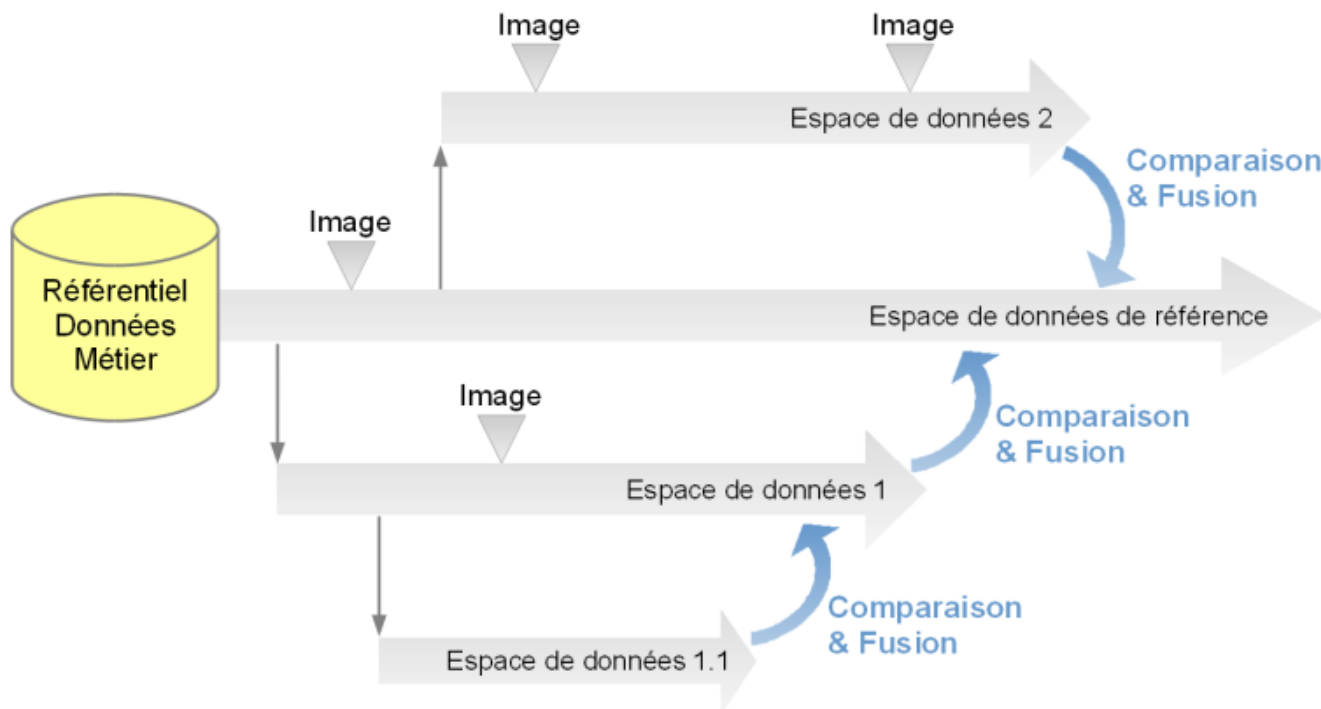
Pour une meilleure compréhension, n'hésitez pas à chercher un mot dans le [glossaire](#).

Concepts

Le cycle de vie des données est souvent complexe. Par exemple, une entreprise a besoin d'avoir une version courante de ses données tout en travaillant sur des évolutions futures. De plus, cette entreprise a besoin de conserver une trace de ses évolutions.

EBX5 permet de créer et gérer plusieurs espaces de données ainsi que des images. Il est possible, en utilisant des espaces de données, de faire des modifications simultanées dans un même référentiel, de les comparer et de les fusionner.

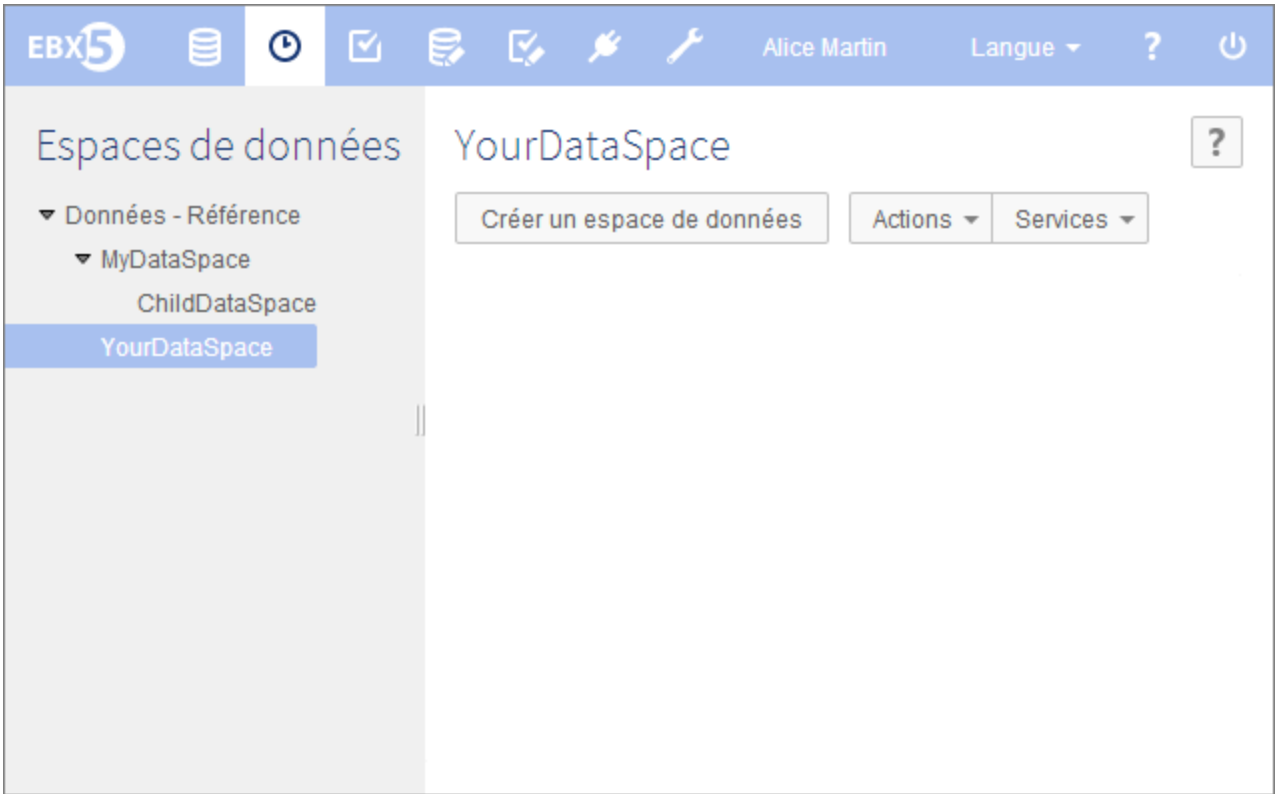
Une image permet de prendre un instantané d'un espace de données afin de conserver une version des données et de pouvoir détecter des modifications ultérieures.



Vue d'ensemble d'un espace de données

Modalités d'accès dans l'interface

Le panneau de navigation affiche l'organisation hiérarchique des espaces de données existants, tandis que l'espace de travail est utilisé pour interagir avec l'espace de données sélectionné, dont il liste les images prises. Pour gérer vos espace de données, sélectionner *Espace de données* dans la barre de menu.



Concepts et outils associés

Instantané	Photographie du contenu d'un espace de données à un instant t.
Espace de données Référence	Ancêtre de tous les autres espaces de données, n'ayant pas de parent et ne pouvant être fusionné.
Fusion	Copie de changements fait sur un espace de données fils vers son parent, qui requiert l'arbitrage de l'utilisateur en cas de conflits.
Espace de données relationnel	Un espace de données relationnel est réservé aux modèles de données en mode relationnel. Dans ce mode, la gestion des données est d'avantage déléguée à la base de données et certaines fonctionnalités ne sont pas disponibles : par exemple, il n'est pas possible de créer une image ou un espace de données fils.

Voir aussi : [Mode relationnel \(en anglais\)](#)

CHAPITRE 15

Actions principales

Création d'un espace de données

Vous pouvez créer un nouvel espace de données en utilisant le bouton *Créer un espace de données* situé dans l'espace de travail. L'espace de données nouvellement créé devient un sous-espace de données de celui qui était sélectionné et le contenu de son parent est virtuellement dupliqué dans le nouvel espace de données. Les informations suivantes sont requises :

Identifiant	Identifiant unique de l'espace de données.
Propriétaire	Utilisateur possédant l'espace de données et étant autorisé à en modifier les informations et les permissions, qui n'est pas obligatoirement son créateur.
Libellé	Libellé et description associés à l'espace de données en plusieurs langues.

Edition de l'espace de données

Informations

Vous pouvez modifier les informations associées à l'espace de données à l'aide du bouton *Actions*.

Propriétaire	Personne possédant l'espace de données et étant autorisée à en modifier les informations et les permissions, qui n'est pas obligatoirement son créateur.
Documentation	Libellé et description associés à l'espace de données en plusieurs langues.
Changement de propriétaire	Modification de l'attribut <i>Propriétaire</i> par le propriétaire d'un espace de données. En son absence, seul l'administrateur a le droit d'effectuer cette modification.
Changement des permissions	Modification des permissions par le propriétaire d'un espace de données. En son absence, seul l'administrateur a le droit d'effectuer cette modification.
Stratégie de chargement	<p>Seul l'administrateur a le droit de modifier cette option.</p> <ul style="list-style-type: none"> • Chargement et déchargement sur demande : est le mode par défaut. Le principal avantage est la capacité de libérer de la mémoire si nécessaire. L'inconvénient est que cela implique un coût de chargement, quand une ressource accédée n'a pas encore été chargée depuis le démarrage du serveur ou si elle a été déchargée entre-temps. • Chargement forcé : ce mode est particulièrement recommandé pour les espaces de données à longue durée de vie et les images, qui sont fréquemment utilisées. • Chargement forcé et pré-validation : ce mode est particulièrement recommandé pour les espaces de données à longue durée de vie et les images, qui sont fréquemment utilisées et pour lesquelles le processus de validation peut être long. <p>Note : Toute modification de la stratégie de chargement requiert un redémarrage du serveur.</p>
Politique de fusion	La politique de fusion des espaces de données enfants ne s'applique qu'au fusion initié par un utilisateur. Elle ne s'applique aux fusions programmatiques (ex: script de workflow).

Les politiques possibles sont :

- **Autoriser des erreurs de validation dans le résultat** : c'est la politique par défaut. Un espace de données enfant peut être fusionné quelque soit la validité du résultat de cette fusion.
- **Fusion pré-validante** : un espace de données enfant peut être fusionné si et seulement si le résultat de la fusion est valide.

Tri des espaces enfants

Définit l'ordre d'affichage des espaces de données enfants dans l'arbre des espaces de données. Si non défini, l'ordre défini par le parent est pris en compte. La valeur par défaut est "par libellé".

Validation

Le contenu d'un espace de données peut être validé d'un coup en utilisant le service validation au niveau espace de données. Ce service est accessible par le bouton *Actions* dans l'espace de travail.

Note : Pour utiliser ce service, l'utilisateur doit avoir la permission de valider chaque jeu de données contenus dans l'espace de données.

Exporter une archive

Le contenu d'un espace de données peut être exporté au sein d'une archive en utilisant le service accessible via le bouton *Actions*.

Note : L'administrateur doit récupérer l'archive exportée. Voir [Archives directory](#) dans le Guide d'administration.

Pour réaliser un export, trois informations sont requises:

- **Nom du fichier archive** : champ texte permettant de renseigner le nom souhaité.
- **Type d'export** : Obligatoire.

Le type d'export par défaut est **Le contenu complet de l'espace de données**, celui ci exporte l'ensemble des données sélectionnées dans l'archive. Si vous travaillez dans un espace de données en fait, il peut être intéressant de comparer l'espace de données avec son père et d'inclure dans l'archive les différences, sous la forme d'un delta.

Deux types d'export permettent de le réaliser : **Mises à jour avec leur contenu complet** et **Mises à jour seules** . Le premier exporte tout les données et le delta tandis que le second n'exporte les données que si une différence est définie dans le delta. La granularité d'export est le niveau table. L'écran de comparaison permet de sélectionner les différences à inclure dans le delta.

- **Jeu de données à exporter** : la table permet de sélectionner les jeux de données qui doivent être exportés. Pour chaque jeu de donnée, il est possible de choisir si les données, les permissions et les informations doivent être exportés.

Importer une archive

Le contenu d'une archive peut être importé dans un espace de données en utilisant le service accessible via le bouton *Actions*.

Si l'archive sélectionnée contient un delta, vous pouvez choisir de l'utiliser. L'écran de comparaison vous permet de sélectionner les différences du delta qui doivent être importées.

Fusion

Des changements à un espace de données peuvent être appliqués à son parent en utilisant la fonction *fusionner*, grâce au bouton *Actions*. Le processus compare les différences entre les deux espace de données à fusionner, puis demande à l'utilisateur d'arbitrer les conflits éventuels.

Pour plus de détails, sur le déroulement des fusions, lisez la page [Fusion](#).

Image

Créer une image est un moyen de sauvegarder un état exact et en lecture seule de votre espace de données à un instant t , avant d'y appliquer tout changement majeur. L'image figée de votre espace de référence devient un point de repère pour vous, au cas où vous auriez besoin d'annuler des changements.

Pour trouver comment faire une image de votre espace de données, lire la page [Image](#).

Permissions

Il est possible pour un propriétaire ou un administrateur de définir des droits d'accès à un espace de données (lecture seule, écriture ou non visible) et de les préciser action par action de manière à protéger l'espace de données et son contenu de tout accès ou modification par des utilisateurs non autorisés. Plus de précisions sur les permissions et leur gestion, peuvent être trouvées à la page [Permissions](#).

Fermeture d'un espace de données

Si vous souhaitez abandonné un espace de données sans le fusionner, ce dernier peut tout simplement être fermé par son propriétaire ou tout utilisateur habilité. Une fois fermé, plus personne ne peut y accéder. L'espace de données peut toutefois être réouvert par un administrateur, sauf si celui-ci a déjà été purgé.

Cela peut se faire à l'aide du bouton *Actions*.

CHAPITRE 16

Image

Une image est une image exacte et en lecture seule d'un espace de données à un instant t.

Création d'une image

Une image peut être obtenue en utilisant le bouton *Créer une image* situé dans l'espace de travail. Les informations suivantes sont requises.

Identifiant	Identifiant unique pour l'image.
Libellé	Libellé et description associés à l'image en différentes langues.

Information

Il est possible de modifier les informations associées à une image grâce au bouton *Actions*.

Propriétaire	Utilisateur possédant l'image et étant autorisé à en modifier les informations et les permissions, qui n'est pas obligatoirement son créateur.
Documentation	Libellé et description associés à l'image en différentes langues
Changement de propriétaire	Modification de l'attribut <i>Propriétaire</i> par le propriétaire d'une image. En son absence, seul l'administrateur a le droit d'effectuer cette modification.
Stratégie de chargement	<p>Seul l'administrateur a le droit de modifier cette option.</p> <ul style="list-style-type: none"> • <i>Charger et décharger à la demande</i> : mode par défaut. Le principal avantage est la capacité de libérer de la mémoire si nécessaire. L'inconvénient est que cela implique un coût de chargement, quand une ressource accédée n'a pas encore été chargée depuis le démarrage du serveur ou si elle a été déchargée entre-temps. • <i>Chargement forcé</i> : ce mode est particulièrement recommandé pour les espaces de données à longue durée de vie et les images, qui sont fréquemment utilisées. • <i>Chargement forcé et prévalidation</i> : ce mode est particulièrement recommandé pour les espaces de données à longue durée de vie et les images, qui sont fréquemment utilisées et pour lesquelles le processus de validation peut être long.

Note : Toute modification de la stratégie de chargement requiert un redémarrage du serveur.

Visualiser le contenu

Il est possible de visualiser le contenu d'une image à l'aide du bouton *Actions*.

Services

Plusieurs autres services sont accessibles à l'aide du bouton *Actions*.

Validation

Le contenu d'une image peut être validé directement en utilisant le service validation au niveau de l'image.

Note : pour pouvoir utiliser ce service, l'utilisateur doit avoir la permission de valider chaque jeu de données contenus dans l'image.

Comparer

Ce service permet de sélectionner un autre espace de données ou une image et de comparer leurs contenus. Un écran de comparaison, similaire à celui qui s'affiche en cas de fusion d'espace de données, résume les différences.

Export

Ce service permet d'exporter le contenu d'une image vers une archive.

Quand on exporte une archive, les informations à renseigner sont les suivantes :

- **Nom de l'archive à créer**: nom du fichier à créer
- **Jeu de données à exporter**: sélectionner les jeux de données à exporter. Pour chaque jeu de données, il est possible de décider, ce qui doit être inclus dans l'archive (données, permissions de jeux de données et/ou informations sur les jeux de données).

Fermeture d'une image

Si vous souhaitez supprimer une image, cela peut être fait en la fermant. Le propriétaire de l'image, ou un utilisateur autorisé, peuvent fermer une image, à l'aide du bouton *Actions*.

Une fois fermée, l'image devient inaccessible. L'image peut toutefois être réouverte par un administrateur, sauf si celle-ci a déjà été purgée.

CHAPITRE 17

Fusion d'un espace de données

Quand le travail dans un espace de données est terminé, vous pouvez effectuer une fusion unidirectionnelle de l'espace de données vers son espace de données parent. Le processus de fusion est le suivant :

1. l'espace de données parent et l'espace de données enfant sont verrouillés pour tous les utilisateurs. Les verrous restent pendant la durée de la fusion. C'est à dire que les contenus des deux espaces de données peuvent être lus, mais ne peuvent pas être modifiés.

Note : cette restriction sur l'espace de données parent s'applique aussi aux fusions des autres espaces de données enfants et non seulement aux modifications directes.

2. les changements dans l'espace de données qui ont été faits depuis sa création sont intégrés dans l'espace de données parent ;
3. l'espace de données enfant est fermé ;
4. l'espace de données parent est déverrouillé.

Initiation d'une fusion

Suivez ces étapes pour initier une fusion d'un espace de données dans son espace de données parent :

1. sélectionnez l'espace de données à fusionner dans le panneau de navigation de la section Espaces de données ;
2. dans l'espace de travail, sélectionnez **Fusionner l'espace de données** dans le menu **Actions**.

Revue et acceptation des changements

Avant la fusion définitive, vous devez décider quels changements survenus dans l'espace de données enfant (source) doivent être fusionnés dans l'espace de données parent (cible).

Note: Cette étape de revue n'existe pas pour une fusion faite par l'intermédiaire d'un service de données ou d'un service programmatique. Pour les fusions automatisées, tous les changements dans l'espace de données enfant remplacent les données dans l'espace de données parent.

Pendant la fusion, un écran de comparaison récapitule tous les changements qui doivent être revus. Deux colonnes de *listes de changements* sont affichées, prenant en compte les changements des comparaisons d'espaces de données suivantes :

- l'espace de données enfant par rapport à son image initiale ;
- l'espace de données parent par rapport à l'image initiale de l'espace de données enfant.

Par défaut, tous les changements détectés sont sélectionnés pour fusion. Vous pouvez désélectionner les changements pour les exclure de la fusion. Vous pouvez voir les changements relatifs aux différentes étendues en sélectionnant des éléments dans le panneau de navigation.

La *fusion à trois points* implique l'espace de données courant, l'image initiale et l'espace de données parent. Elle permet de détecter les conflits. Parfois, les données ont été modifiées à la fois dans l'espace de données courant et dans son parent.

Le processus de fusion concerne aussi les droits d'accès décrits dans une table. Il faut également passer en revue les modifications des permissions pour décider si elles seront incluses dans la fusion.

Lorsque vous avez choisi les changements à fusionner, vous devez cliquer sur le bouton intitulé *Marquer les différences comme revues* pour indiquer que vous avez vérifié les changements dans l'étendue actuelle. Tous les changements doivent être revus pour effectuer la fusion.

Types de modifications

Le processus de fusion considère comme modifications à évaluer :

- la création d'un enregistrement ou d'un jeu de données ;
- la modification de toute entité ;
- la suppression d'un enregistrement, d'un jeu de données, ou de la valeur d'un noeud ;
- la modification des permissions d'une table.

Types de conflits

Cette interface de revue affiche également les conflits trouvés. Des conflits peuvent survenir quand une étendue contient des modifications dans l'espace de données source et l'espace de données cible.

Les conflits sont catégorisés comme suit :

- *un conflit de création d'un enregistrement ou d'un jeu de données,*
- *un conflit de modification de toute entité,*
- *un conflit de suppression d'un enregistrement ou d'un jeu de données,*
- *tout autre conflit.*

Finalisation d'une fusion

Lorsque vous avez vérifié tous les changements et décidé lesquels sont à inclure dans le résultat de la fusion, cliquez sur le bouton **Fusionner >>** dans le panneau de navigation.

En fonction de la politique de fusion de l'espace de données parent, le processus de finalisation peut différer. Par défaut, une fusion peut être effectuée même si le résultat de la fusion contient des erreurs de validation. L'administrateur de l'espace de données parent peut configurer la politique de fusion pour que les fusions avec ses espaces de données enfants soient finalisées uniquement si le résultat n'a pas d'erreur de validation.

Si la politique de fusion par défaut est utilisée, la fusion est achevée quel que soit l'état de validation de l'espace de données résultant.

Si la politique de fusion pré-validante est utilisée, un espace de données dédié est d'abord créé pour contenir le contenu de la fusion. S'il est valide, cet espace de données est automatiquement fusionné avec l'espace de données parent.

Dans le cas contraire, si des erreurs de validation sont détectées dans l'espace de données dédié, seuls l'espace de données d'origine et l'espace de données dédié contenant le résultat de la fusion - nommé "[fusion] <nom de l'espace de données enfant>" - seront accessibles. Les options suivantes sont alors proposées dans le menu de l'espace de travail **Actions > Fusion en cours** :

- **Abandonner la fusion** : cette action abandonne la fusion en cours et récupère l'espace de données enfant dans l'état d'avant fusion.
- **Poursuivre la fusion** : cette action permet de retenter une fusion après avoir effectué les corrections nécessaires dans l'espace de données de fusion dédié.

Configuration de la politique de fusion d'un espace de données

En tant qu'administrateur d'un espace de données, vous pouvez interrompre la finalisation des fusions avec ses espaces de données enfant à travers de l'interface utilisateur si le résultat contient des erreurs de validation. Pour cela, cliquez sur **Actions > Informations** dans l'espace de travail de l'espace de données parent. Sur la page des informations de cet espace de données, positionnez la **Politique de fusion des enfants** à **Fusion pré-validante**. Cette politique de fusion sera appliquée pour toutes les fusions des espaces de données enfant avec l'espace de données parent ainsi paramétré.

Note

En mode composant web, le comportement pour la politique de fusion est le même ; la politique définie par l'espace de données parent sera automatiquement utilisée lors de la fusion des modifications de l'espace de données enfant. Cependant, cette politique n'est pas appliquée pour les fusions programmatiques, donc ignorée pour les tâches automatiques des workflows de données.

Voir aussi : [Politique de fusion](#)

Abandon d'une fusion

Une fusion est effectuée dans le contexte d'une session utilisateur, et doit être achevée en une seule opération. Si vous décidez de ne pas poursuivre la fusion après l'avoir initiée, vous pouvez cliquer sur le bouton **Annuler** afin d'abandonner l'opération.

Si vous naviguez vers une autre page pendant une fusion, la fusion sera abandonnée, mais les verrous sur l'espace de données parent et l'espace de données enfant resteront. Il faudra les déverrouiller dans la section Espaces de Données.

Pour déverrouiller un espace de données, sélectionnez l'espace de données dans le panneau de navigation, et ensuite cliquez sur le bouton **Déverrouiller** dans l'espace de travail. Si vous effectuez le déverrouillage depuis l'espace de données enfant, les deux espaces de données seront déverrouillés. Si vous effectuez le déverrouillage depuis l'espace de données parent, seulement l'espace de données parent sera déverrouillé, donc vous devrez également effectuer un déverrouillage de l'espace de données enfant.

CHAPITRE 18

Permissions

Les permissions de l'espace de données peuvent être accédées à l'aide du bouton *Actions* dans l'espace de travail.

Une permission est toujours attachée à un profil.

Permissions sur un espace de données

Permissions générales

- **Identifiant de l'espace de données** : Indique l'espace de données sur lequel les permissions vont être appliquées.
- **Sélection du profil** : indique le profil affecté par la permission définie.
- **Restriction d'accès** : indique si la permission définir ici restreint celles définies sur pour d'autres profils. Voir aussi la page [Restriction d'accès](#).
- **Permissions d'accès à l'espace de données** : Indique la permission d'accès globale sur l'espace de données (lecture seule, écriture ou non visible). Voir ci-dessous.

Lecture seule

- Permet de visualiser l'espace de données et son image et de voir les espaces de données fils, selon les droits s'appliquant à chacun d'eux.
- Permet de visualiser le contenu de l'espace de données sans pouvoir le modifier, à condition que les droits s'appliquant au contenu vous en autorise l'accès.

Ecriture

- Permet de voir l'espace de données.
- Permet d'accéder aux jeux de données selon les droits que l'on possède sur eux.

Non visible

- Ni l'espace de données, ni ses images ne peuvent être vus.
 - A partir d'un espace de données fils, on peut voir l'espace de données courant sans toutefois pouvoir le sélectionner.
 - Absence d'accès au contenu de l'espace de données.
 - Aucune action ne peut être effectuée sur l'espace de données.
-

Actions permissibles à la carte

- **Créer un espace de données fils** : indique si le profil peut créer un espace de données fils.
- **Créer une image** : indique si le profil peut créer une image de l'espace de données.
- **Initier une fusion** : indique si le profil peut fusionner un espace de données avec son parent.
- **Exporter une archive** : indique si le profil peut accéder au service export.
- **Importer une archive** : indique si le profil peut accéder au service import.
- **Fermer un espace de données** : indique si le profil peut fermer un espace de données.
- **Fermer une image** : indique si le profil peut fermer l'image d'un espace de données.
- **Droits sur les services** : spécifie les permissions d'accès aux services. Un service n'est pas accessible à un profil s'il est barré.
- **Permissions d'un espace de données fils à la création** : spécifie les permissions d'accès, qui seront présentes dans un espace de données fils nouvellement créé.

Permissions sur les images

Permissions sur l'image initiale d'un espace de données

Sont appliquées les permissions définies pour l'espace de données.

Permissions sur une image d'un espace de données

Sont appliquées les permissions définies pour l'espace de données.

Jeux de données

CHAPITRE 19

Introduction aux jeux de données

Une fois que vos données de référence ont été modélisées, et ces modèles publiés, vous pouvez gérer vos données de référence au travers des actions suivantes :

- créer un nouveau jeu de données dans un data space (voir [création](#)),
- sélectionner un espace de données où travailler, un jeux de données dans celui-ci, puis consulter l'arborescence des groupes, tables et champs (voir [actions](#)),
- lire, créer, modifier ou supprimer des enregistrements dans une table d'un jeu de données (voir [tables](#)),
- définir des vues personnalisées et/ou des hiérarchies sur les tables (voir [modes de vue avancés](#)),
- commencer à travailler à partir d'un jeu de données existant en en copiant les données (voir [héritage](#)),
- définir qui peut, ou ne peut pas, accéder à certaines données (voir [permissions](#)).

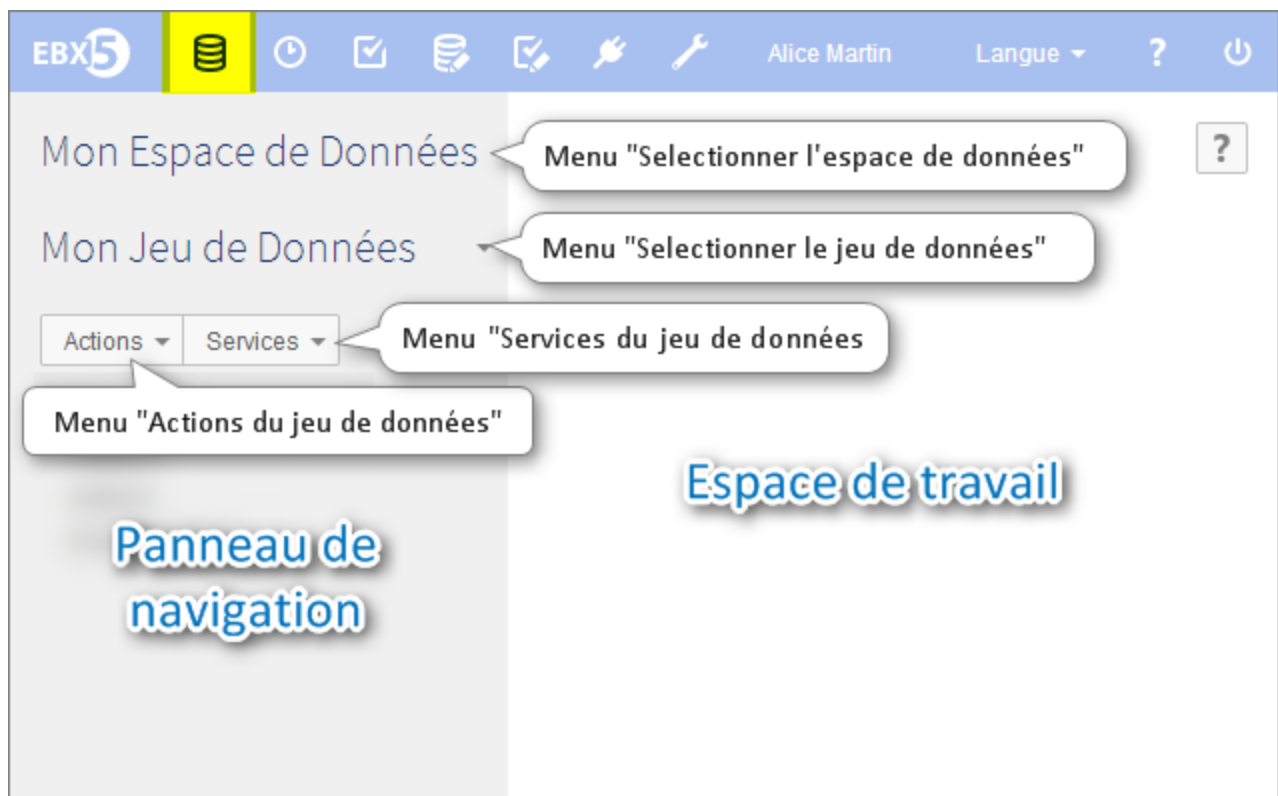
Un jeu de données correspond aux données présentées sous forme de tables ou hiérarchies, et pouvant être filtrées en vue multi-critères. L'accès au contenu d'un jeu de données peut être restreint à l'aide de règles de permissions.

Pour une meilleure compréhension de ces concepts, prenez la liberté de regarder notre [glossaire](#).

Vue d'ensemble d'un jeu de données

Modalités d'accès dans l'interface

Pour interagir avec vos données, sélectionnez *Données* dans la barre de menu. Puis ouvrez un espace de données dans la partie haute du panneau de navigation.



Les utilisateurs peuvent sélectionner un jeu de données en utilisant le panneau de navigation. Deux boutons s'affichent, le premier permet de sélectionner un espace de données, tandis que le second permet de sélectionner un jeu de données. Une fois le jeu de données sélectionné, sa structure apparaît dans le panneau de navigation, et peut être utilisée pour sélectionner un groupe ou une table. Les valeurs de ses éléments (tables, champs, enregistrements) s'affichent alors dans l'espace de travail.

Concepts et outils associés

Une fois dans la section *Données*, vous allez aborder les outils et les notions suivantes :

Espace de données	Conteneur de données, dont le contenu peut être mis à jour en toute isolation des autres éléments à proximité.
Jeu de données	Groupe de données prédéfini dans le modèle de données, et ayant une utilité ou un but commun
Arborescence	Moyen de visualiser le contenu d'un jeu de données dans le panneau de navigation
Table cible	Table étudiée, dont on souhaite visualiser les dépendances aux autres tables. C'est le dernier et le plus spécifique des niveaux de la hiérarchie, à l'image des feuilles d'un arbre.
Enregistrement	Groupe de champs formant une unité d'information entrée par un utilisateur dans un certain jeu de données, apparaissant comme le contenu d'un rang dans une table
Hiérarchie	Arborescence représentant l'enchaînement des dépendances entre les tables. Elle peut être équilibrée, déséquilibrée, irrégulière ou en réseau.
Relation récursive	Occurrence d'un lien de dépendance entre deux entités du même niveau de dimension
Dimension	Axe possible d'analyse d'une table cible, incluant différents niveaux de dimension (exemple : produits, familles, catégories, etc.)

Voir aussi :

- [Modèles de données](#)
- [Espaces de données](#)
- [Modèles de workflow](#)
- [Workflows de données](#)
- [Service de données](#)

CHAPITRE 20

Actions principales

Création d'un jeu de données

Un jeu de données peut être créé à l'aide du bouton *Créer un jeu de données*, accessible à partir du sélecteur de jeu de données. S'il n'existe aucun jeu de données dans l'espace de données sélectionné, le bouton s'affiche directement dans le panneau de navigation. Un assistant vous permet alors de créer le jeu de données.

Information

Les informations associées au jeu de données peuvent être éditées à l'aide du bouton *Actions* dans le panneau de navigation. Les informations éditables sont les suivantes :

- **Propriétaire** : Utilisateur pouvant éditer ses informations et définir des règles de permissions sur le jeu de données, qui n'est pas obligatoirement son créateur.
- **Documentation** : Libellé et description associés au jeu de données dans différentes langues.
- **Activé** : Activation du jeu de données nécessaire à la sélection de règles de validation.

Édition du jeu de données

Navigation dans le jeu de données

Dans un jeu de données, vous pouvez trouver des tables, des champs et des groupes de l'un et de l'autre. Pour y accéder, utilisez l'arborescence du panneau de navigation et cliquez sur le jeu de données qui vous intéresse. Son contenu s'affiche dans l'espace de travail.

Paramétrage des permissions

Pour savoir comment personnaliser les permissions pour chaque utilisateur potentiel, lisez attentivement la page [permissions](#).

Validation du jeu de données

Il est possible de valider un jeu de données en sélectionnant **Actions > Valider** depuis le panneau de navigation. Les éventuels messages issus de la validation du jeu de données sont présentés dans un rapport. Depuis le rapport de validation, vous pouvez cliquer sur le bouton **Revalider** pour mettre à jour ce rapport, ou vous pouvez cliquer sur le bouton **Réinitialiser le rapport de validation** pour supprimer tous les messages de validation actuellement associés au jeu de données afin de pouvoir relancer une validation complète.

Dans la section Données, vous pouvez également valider une table en sélectionnant une table dans le panneau de navigation et en utilisant l'action **Actions > Valider** dans l'espace de travail.

Voir [Validation](#) pour obtenir plus d'informations concernant la validation incrémentale de données.

Table de jeux de données

Le contenu d'une table s'affiche dans l'espace de travail, qui est aussi l'endroit où l'utilisateur peut interagir avec ce contenu.

Édition d'un enregistrement

Créer	Un nouvel enregistrement peut être créé à l'aide du bouton "+" situé en haut à gauche de la table. Un formulaire s'affiche, permettant d'entrer des données. Les données obligatoires sont repérées par une astérisque rouge.
Éditer	Un enregistrement peut être édité en double-cliquant dessus. Le formulaire s'affichant permet d'éditer l'enregistrement, tandis que le bouton <i>Rétablir</i> permet de recharger le formulaire sans soumettre aucun des changements effectués.
Dupliquer	<p>Un enregistrement qui a été sélectionné à l'aide d'une case à cocher peut être dupliqué grâce au bouton <i>Actions</i>.</p> <p>Un formulaire apparaît avec les valeurs pré-remplies à l'image de l'enregistrement copié. La clé primaire doit ensuite être modifiée pour pouvoir créer ce nouvel enregistrement, sauf si celle-ci est générée (par exemple une valeur auto-incrémentée).</p>
Supprimer	Un ou plusieurs enregistrements, ayant été sélectionnés à l'aide des cases à cocher, peuvent être supprimés grâce au bouton <i>Actions</i> .
Comparer	<p>Deux enregistrements, ayant été sélectionnés à l'aide des cases à cocher, peuvent être comparés à l'aide du bouton <i>Actions</i>.</p> <p>Note : Le contenu des noeuds terminaux complexes, comme les listes agrégées ou les attributs utilisateurs, ne sont pas comparés pendant ce processus. Le service de comparaison ignore toutes les différences entre les valeurs des noeuds terminaux complexes dans les enregistrements.</p>

Import/export d'enregistrement

Dans une table, les enregistrements peuvent être importés ou exportés, depuis ou vers des formats CSV ou XML.

Vous pouvez soit sélectionner manuellement certains enregistrements à exporter, grâce aux cases à cocher, ou exporter l'ensemble de la table.

Voir aussi :

- [Services CSV](#)
- [Services XML](#)


Tri des données

Un critère de tri contrôle l'ordre dans lequel les enregistrements sont présentés. Ils peuvent être définis grâce au bouton *Vue*. L'ordre par défaut est par clé primaire ascendante. L'option *Rétablir* permet d'annuler toute modification.

Il est défini par un nom de colonne et un ordre (ascendant, de la valeur la plus basse à la plus élevée, ou descendant, de la valeur la plus élevée à la plus faible). Un critère de tri peut être ajouté et retiré à l'aide des liens *Ajouter* et *Retirer*. L'ordre de classement d'un critère peut être modifié en cliquant sur les liens *ascendant* ou *descendant*.

L'ordre entre les critères peut être ajusté à l'aide du bouton situé à droite de la liste de critères.

Recherche et filtrage des données

Il existe des outils spécifiques dédiés à la recherche d'enregistrements dans une table. On peut y accéder par l'icône , qui affiche les panneaux des filtres.

Chaque panneau de filtre (ou de recherche) propose une case à cocher sur sa barre de titre : cocher cette case applique le filtre ; la décocher le désapplique. Dans le reste de cette section, les différents types de filtres disponibles sont détaillés.

Note

Appliquer une vue personnalisée reinitialisera et retirera tous les filtres appliqués.

Recherche typée

En mode simple, la recherche typée permet d'ajouter des critères contextualisés sur un ou plusieurs champs. Lors de l'ajout d'un critère les opérateurs proposés seront pertinents vis-à-vis du type du champ correspondant.

En activant le mode avancé, il est possible de créer des sous-blocs contenant des critères, afin de créer des opérations logiques plus complexes dans l'élaboration du filtre.

Recherche textuelle

La recherche textuelle est utilisée pour une recherche de texte brut sur un ou plusieurs champs. Cette recherche ne prend pas en compte le type du champ.

- Si le texte entré contient un ou plusieurs mots sans caractère de remplacement (* ou ?), les champs trouvés seront ceux contenant tous ces mots. Les mots entre guillemets ("aa bb" par exemple) sont considérés comme un seul mot.
- Les caractères usuels de remplacement sont proposés : l'étoile * (n'importe quel texte) ou le point d'interrogation ? (n'importe quel caractère). Pour des raisons de performance, leur utilisation est restreinte à un seul de ces caractères par recherche.
- Les caractères de remplacement peuvent être considérés comme de simple caractères en les échappant à l'aide du caractère '\', par exemple, '*'.

Exemples :

- aa bb : le champ contient 'aa' et 'bb'.
- aa "bb cc" : le champ contient 'aa' et 'bb cc'.
- aa* : le champ commence par 'aa'.
- *bb : le champ se termine par 'bb'.
- aa*bb : le champ commence par 'aa' et se termine par 'bb'.
- aa? : le champ commence par 'aa' et a une taille de 3 caractères.
- ?bb : le champ se termine par 'bb' et a une taille de 3 caractères.
- aa?bb : le champ commence par 'aa' et se termine par 'bb' et a une taille de 5 caractères.
- aa*bb : le champ contient 'aa*bb' tel quel.

Sur les tables de grande taille, il est recommandé de ne sélectionner qu'un seul champ de la table ; si le champ n'est pas du type 'chaîne de caractères', il est conseillé d'entrer un texte au bon format, par exemple :

- Pour un booléen : Oui, Non
- Pour une date : 01/01/2000
- Pour un nombre : 100000 ou 100 000
- Pour un champ énuméré : Rouge, Bleu..

Option *Sensible à la casse* : la recherche tient compte de la casse (majuscules et minuscules sont distinctes).

Filtre par messages de validation

Le filtre par messages de validation permet de voir les enregistrements en fonction de leur statut dans la dernière validation effectuée. Vous pouvez voir les enregistrements avec les niveaux 'Erreur', 'Avertissement' et 'Information'.

Note

Cette recherche peut s'appliquer seulement sur les enregistrements de la table qui ont déjà été validés ; pour cela, il faut sélectionner **Actions > Valider** au niveau de la table dans l'espace de travail, ou au niveau du jeu de données dans le panneau de navigation.

Recherches spécifiques sur tables

Pour chaque table, le modèle peut spécifier des filtres additionnels pour la recherche.

Vues personnalisées

Vous pouvez personnaliser la présentation d'une table en appliquant ou en définissant des vues. En utilisant le bouton *Vue*, il est possible de sélectionner une vue personnalisée existante ou d'en définir une nouvelle. Une fois une vue appliquée, elle peut être définie comme la *vue par défaut* pour la table. En cliquant sur *Enlever toutes les vues*, la vue par défaut est rétablie.

Voir aussi : [Vues personnalisées](#)

CHAPITRE 21

Vues personnalisées

Il y a deux modes avancés de visualisation :

- le premier est spécifique aux tables et vous permet de filtrer les informations selon des critères donnés ([vue taublaire simple](#)),
- le deuxième vous permet d'établir des liens et d'organiser des informations issues de différentes tables ([hiérarchies](#)).

Une vue personnalisée peut être créée via le bouton *Vue*.

Publication de vue

Une vue peut être publiée afin de la rendre accessible à tous les autres utilisateurs grâce aux *composants web*, tâches utilisateurs du workflow, ou data services. Il s'agit d'une publication technique, elle ne donnera pas accès à cette vue à tous les utilisateurs via l'interface graphique (voir *Profils autorisés* ci dessous).

Description d'une vue

Ce formulaire permet de spécifier les informations liées à la vue personnalisée communes à tous les modes.

Propriétaire	Nom du propriétaire de la vue personnalisée, qui peut à ce titre la gérer et la modifier.
Documentation	Libellé et description associés à la vue personnalisée dans différentes langues.
Profils autorisés	Détail des profils ayant le droit d'utiliser la vue personnalisée.
Mode de la vue	Mode de la vue personnalisée (voir ci dessus).

Vue tabulaire simple

Les vues tabulaires simples offrent la possibilité de définir des critères pour filtrer les enregistrements et de sélectionner les colonnes à afficher.

Critères de recherche

Ce champs permet de définir des critères, qui vont être utilisés pour filtrer les enregistrements.

Voir aussi : [Documentation sur l'éditeur de critères](#)

Critères de tri

Ce champs permet de définir des critères, qui vont être utilisés pour trier les enregistrements.

Affichage des colonnes

Ce champs permet de sélectionner, à l'aide des cases à cocher, les colonnes de la table à afficher dans la vue personnalisée.

Hiérarchies

Une hiérarchie est une arborescence permettant de souligner les relations existant entre les données. Elle peut être structurée sur plusieurs niveaux, appelés niveaux de dimension. En outre, il est possible de définir des critères sur des niveaux permettant de filtrer les enregistrements.

Dimension d'une hiérarchie

Une dimension est ce qui définit les différentes dépendances dans la hiérarchie (exemple : produits par catégories). En démarrant d'une table cible, sélectionner les liens de dépendances pas à pas jusqu'au dernier.

Chaque lien de dépendance établit devient un nouveau niveau de dimension.

Configuration d'une hiérarchie

Ce formulaire permet de configurer les niveaux de hiérarchie. Pour chaque niveau, vous pouvez configurer les libellés et spécifier les critères de filtrage des enregistrements.

Libellés

Dans l'arbre, les enregistrements sont nommés par des libellés, écrit dans différentes langues. Les champs d'enregistrement peuvent être aisément insérés dans un libellé en utilisant l'assistant (utiliser le bouton à droite du champ pour y accéder).

Filtre

Ce formulaire permet de définir des critères, qui pourront être utilisés pour filtrer les enregistrements.

Voir aussi : [*Documentation sur l'éditeur de critères*](#)

Ordonnancement

Il est possible de spécifier un champ d'ordonnancement qui permettra à l'utilisateur de définir l'ordre des noeuds enfants. Un champ d'ordonnancement doit être un entier et être en mode caché dans le modèle de données.

Des actions de positionnement sur chaque noeud sont alors possibles, à moins que le champ d'ordonnancement ne soit en 'lecture seulement' ou qu'un filtre ne soit défini sur la hiérarchie.

A défaut d'un noeud d'ordonnancement, les noeuds enfants sont triés selon l'ordre alphabétique des libellés des noeuds.

CHAPITRE 22

Héritage


Habituellement les gens créent un jeu de données à partir d'un modèle de données. EBX5 vous permet de créer des jeux de données additionnels, branchés à partir d'un jeu de données racine. Ces jeux de données enfants héritent de leur parents. Plusieurs niveaux d'héritage peuvent être créés.

Cela peut être utilisé pour adapter des données de référence à divers contextes, comme des zones géographiques et/ou des pays.

Note : Le comportement standard est d'interdire l'héritage de jeux de données. Il est nécessaire d'activer explicitement cette fonction quand vous créez votre modèle de données.

Voir aussi : [Configuration du modèle de données](#)

Arbre des jeux de données

Une fois le jeu de données racine créé, un jeu de données fils peut être créé à l'aide du bouton  situé dans l'écran de sélection des jeux de données du panneau de navigation. On demande ensuite à l'utilisateur de fournir un nom unique pour le jeu créé. Il peut aussi, s'il le souhaite, lui donner un libellé local optionnel et une description.


Note :

- Un jeu de données ne peut pas être supprimé s'il a des jeux de données fils. Ces enfants doivent être supprimés préalablement.
- Si un jeu de données fils est dupliqué, le jeu de données nouvellement créé va être inséré dans l'arbre des jeux de données existants, au même niveau de l'arbre que le jeu de données dupliqué.

Héritage de valeurs

Quand un jeu de données fils est créé, il hérite de toutes les valeurs des champs et enregistrements de tables de son parent. Un champ ou un enregistrement peut soit hériter ses valeurs ou les surcharger.


Les valeurs surchargées utilisent le style par défaut tandis que les valeurs héritées sont signalées par un repère dans le coin en haut à gauche de la cellule.

Le bouton  peut être utilisé pour indiquer, si une valeur est héritée ou surchargée.

Héritage d'enregistrement

Une table dans un jeu de données fils va hériter de l'enregistrement de la table située chez son ancêtre. Il est possible d'éditer ou de supprimer ces enregistrements, de nouveaux enregistrements peuvent également être créés et seront hérités par le jeu de données fils. Plusieurs états sont définis pour différencier ces enregistrements.

Racine	Un enregistrement racine est un enregistrement créé dans le jeu de données courant, qui n'existe pas chez ses jeux de données ancêtre. Il sera hérité par les jeux de données fils.
Hérité	Un enregistrement hérité est défini dans un des jeux de données ancêtre du jeu de données courant.
Surchargé	Un enregistrement surchargé est un enregistrement hérité, dont les valeurs sont éditées dans le jeu de données courant. Les valeurs surchargées seront héritées par les jeux de données fils.
Occulté	Un enregistrement occulté est un enregistrement hérité, qui est supprimé du jeu de données courant. Il apparaîtra toujours dans le jeu de données courant comme un enregistrement barré, mais il ne sera pas hérité par les jeux de données fils.

Le bouton  indique, que l'enregistrement est hérité. Il peut également être utilisé pour changer son état en "surchargé" ou "hérité". Veuillez noter que c'est le même bouton, qui permet de changer le statut d'un enregistrement occulté en "hérité".

Les tables suivantes résume, ce qui se produit quand on crée, édite ou supprime un enregistrement dépendant de l'état initial.

Etat \ Opération	Créer	Editer la valeur	Supprimer
Racine	Création normale d'un enregistrement. L'enregistrement nouvellement créé sera hérité par ses jeux de données fils.	Edition normale d'un enregistrement. Les nouvelles valeurs seront héritées par ses jeux de données fils.	Suppression normale d'un enregistrement. L'enregistrement va disparaître du jeu de données hérité.
Hérité	Si un enregistrement est créé à l'aide de la clé primaire d'un enregistrement hérité existant, l'état de l'enregistrement sera changé à "surchargé" et sa valeur modifiée selon celle soumise à sa création.	Un enregistrement hérité doit être déclaré comme surchargé pour éditer cette valeur.	Supprimer un enregistrement hérité le fait passer à l'état "occulté".
Surchargé	Pas applicable. Il est impossible de créer un nouvel enregistrement si la clé primaire est déjà utilisée.	Un enregistrement surchargé peut être retourné à l'état hérité, mais sa valeur spécifique sera perdue. Les valeurs de l'enregistrement surchargé peuvent être changée à "hérité" ou peuvent être spécifiées.	Supprimer un enregistrement surchargé change son état à "occulté".
Occulté	Si un enregistrement est créé en utilisant la clé primaire de l'enregistrement existant occulté, l'état de l'enregistrement sera changé à "surchargé" et sa valeur modifiée d'après celle soumise à la création.	Un enregistrement occulté ne peut plus être édité.	Pas applicable. Un enregistrement occulté est déjà considéré comme supprimé et à ce titre ne peut pas être supprimer une deuxième fois.

CHAPITRE 23

Permissions

Les permissions de jeux de données peuvent être accédées à l'aide du bouton *Actions* dans le panneau de navigation.

Les permissions sont toujours données via un profil.

Profil

Indique le profil affecté par une permission.

Restriction d'accès

Indique si les permissions définies ici restreignent celles définies pour d'autres profils. Pour plus de détails, voir [Restriction d'accès](#).

Actions sur les jeux de données

Cette section spécifie les permissions des actions sur les jeux de données.

Créer un jeu de données fils	Indique si un profil peut créer un jeu de données fils. L'héritage doit aussi être activé dans le modèle de données.
Dupliquer un jeu de données	Indique si un profil peut dupliquer un jeu de données.
Supprimer un jeu de données	Indique si le profil peut supprimer le jeu de données.
Activer/Inactiver un jeu de données	Indique si un profil peut modifier la propriété <i>Activé</i> dans les informations du jeu de données.
Créer une vue	Indique si un profil peut créer des vues personnalisées et des hiérarchies.

Droits sur tables

Cette section spécifie les permissions par défaut pour toutes les tables. Des permissions spécifiques peuvent aussi être définies pour une table en cliquant sur le bouton *Ajouter une occurrence* puis en sélectionnant une des options suivantes :

Créer un nouvel enregistrement	Indique si un profil peut créer des enregistrements dans une table.
Remplace un enregistrement hérité	Indique si un profil peut remplacer des enregistrements hérités dans une table. Cette permission est utile quand on utilise l'héritage de jeu de données.
Occulte un enregistrement hérité	Indique si un profil peut occulter des enregistrements hérités dans une table. Cette permission est utile quand on utilise l'héritage de jeu de données.
Supprimer un enregistrement	Indique si un profil peut supprimer des enregistrements dans une table.

Droits sur valeurs

Cette section spécifie les permissions d'accès par défaut pour tous les éléments (tables, groupes et champs) d'un jeu de données et permet de définir des permissions pour des éléments spécifiques. Les permissions d'accès par défaut sont utilisées, à condition qu'il n'y ait pas de permission spécifique allouée à un élément.

Le selecteur de droits spécifique permet d'attribuer des permissions d'accès spécifiques pour un élément. Les boutons *Non visible*, *Lecture seule* et *Ecriture* règle les permissions d'accès correspondant à l'élément sélectionné.

Il est possible de retirer une permission spécifique d'accès en utilisant le bouton *(défaut)*.

Droits sur les services

Cette section spécifie les permissions d'accès sur les services. Un service n'est pas accessible à un profil s'il est barré.

Modèles de workflow

CHAPITRE 24

Introduction aux modèles de workflow

Le travail collaboratif est un moyen efficace de produire, mettre à jour, fusionner et valider des données dans une entreprise. Cependant il n'est pas toujours facile d'obtenir que des gens de différents endroits et ayant des compétences différentes, travaillent ensemble pour remplir un objectif à une échéance commune.

En cela, vous trouverez l'outil de modélisation de workflow bien utile, car il va vous permettre de définir des processus de gestion des données impliquant vos collaborateurs. Pour ce faire, vous allez avoir besoin de :

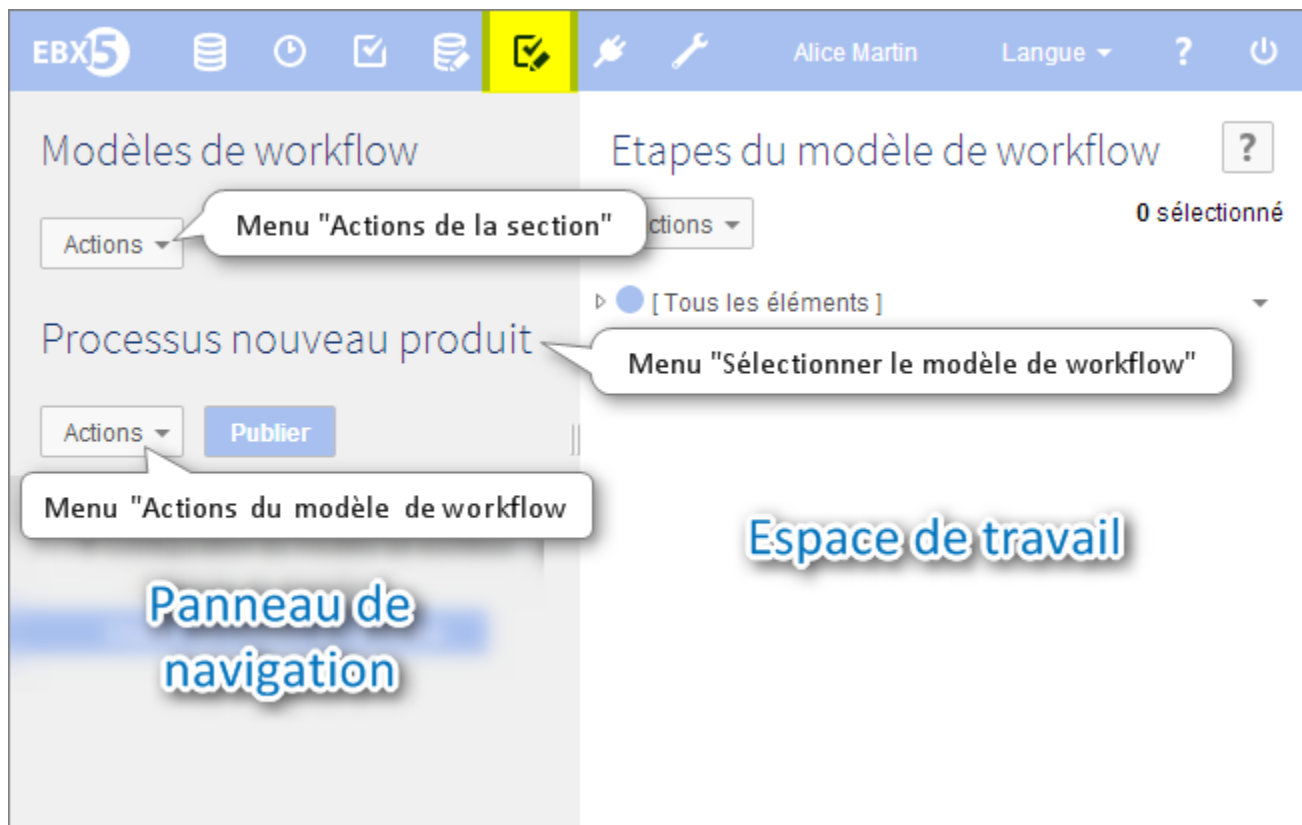
- définir des tâches réalisables par un utilisateur (voir [tâche utilisateur](#)) ou automatiquement par le système (voir [tâche autonome](#)),
- spécifier les responsabilités de chacun (voir [création](#)),
- d'envoyer des courriers électroniques de notification à des collègues, dont la participation est requise (voir [notification](#)),
- de connaître les limites de votre modèle; ce qui peut ou ne peut pas être fait (voir [limitations](#)),
- de publier votre modèle en tant que workflow pour pouvoir vous en servir (voir [publication](#)).

Un modèle de workflow définit les tâches à réaliser et les responsabilités impliquées. Il peut ensuite être publié en tant que publication de workflow. Il s'agit d'une succession de deux types de tâches : *tâches autonomes* et *tâches utilisateurs*, avec la possibilité d'ajouter des fourches conditionnelles entre les tâches.

Pour trouver le sens d'un mot, vous pouvez le rechercher dans notre [glossaire](#).

Vue d'ensemble d'un modèle de workflow

Modalités d'accès dans l'interface



Concepts et outils associés

Tâche autonome	Aucun utilisateur n'est impliqué dans ce type de tâche. Cela peut être, par exemple, une fusion automatique, la création d'une image d'un espace de données, etc.
Tâche utilisateur	Cela implique au moins un utilisateur et éventuellement plusieurs, qui doivent effectuer des bons de travail.
Bon de travail	<p>Tâche unitaire réalisée par l'utilisateur, auquel elle a été allouée, et dont l'exécution permet au workflow de progresser.</p> <p>Un bon de travail est lié à l'exécution d'un composant HttpManager.</p>
Fourche conditionnelle/Condition	Elle décide à partir du résultat des tâches précédentes, quelle route doit être prise par un workflow. Cet embranchement comprends deux dérivations. Par exemple, quelqu'un peut continuer un workflow normalement, tandis que quelqu'un d'autre retourne à une tâche précédente; ou il peut y avoir deux routes différentes en parallèle.
Contexte des données	Il s'agit d'une variable hébergeant temporairement des données entrée/sortie liées à l'exécution d'un workflow. Son but est de faciliter le transfert d'informations clés d'une étape à l'autre (par exemple, le nom d'un espace de données créé à l'étape 1 et étant réutilisé à l'étape 2 dans un autre but).

CHAPITRE 25

Actions principales

Modélisation d'un workflow

Conception

Tout d'abord, vous devez penser à quel type d'évolution vos données vont régulièrement être soumises, et combien de personnes seront impliquées dans ces évolutions.

Ensuite, vous devrez schématiser tout cela et le traduire en un modèle de workflow dans EBX5.

Pour trouver comment procéder dans l'interface, lisez la section [création](#)).

Validation

Une fois le premier jet réalisé dans l'interface, votre modèle va devoir être validé, de manière à vérifier, qu'il n'y a pas d'erreurs dans le code associé. S'il y en a, elles devront être résolues avant que vous puissiez publier et utiliser votre modèle.

Publication

Une fois validé, votre modèle de workflow est prêt pour la publication. En le publiant, vous en générez une image figée, qui devient disponible à l'usage dans la section workflow.

Limitations

Les fonctionnalités suivantes ne sont pas supportées :

- **Tâches parallèles**, deux routes, ou plus, s'exécutent simultanément.
- **Tâches programmées**, tâches exécutées dès lors que leur tour vient, et dont l'exécution ne peut pas être reportée.
- **Tâches événementielles**, permettant au workflow de progresser quand il reçoit un événement, du type appel web service.
- **Limitation temporelle** sur la durée d'une tâche.

Modèles de message génériques

Des courriers électroniques de notification peuvent être envoyés pour informer des utilisateurs spécifiques d'un événement donné pendant l'exécution d'un workflow.

Des modèles de message peuvent être définis et réutilisés dans chaque modèle de workflow en sélectionnant l'entrée "Modèles de message" dans le menu "Actions" global des Modèles de Workflow.

Ces modèles de message sont partagés par tous les modèles de workflow et sont figés et inclus dans chaque publication de workflow. Ainsi, pour prendre en compte les modifications des modèles de message, il sera nécessaire de mettre à jour les publications existantes en re-publiant les modèles de workflow concernés.

Quand vous créez un nouveau message type, deux champs sont requis :

- **Libellé & Description:** spécifie le libellé et la description associée à ce message type en différentes langues.
- **Message:** spécifie l'objet de l'email et son corps de texte en différentes langues.

Le message peut être amélioré à l'aide de variables type contexte de données telles que : `${nom.variable}` . De plus, les variables système ci-dessous sont disponibles :

Syntaxe des variables	Signification
system.time	Heure système.
system.date	Date système.
workflow.lastComment	Dernier commentaire sur la tâche utilisateur précédente.
workflow.lastDecision	Dernières décisions sur la tâche utilisateur précédente.
user.fullName	Nom complet de l'utilisateur notifié.
user.login	Login de l'utilisateur notifié.
workflow.process.label	Libellé du workflow en cours.
workflow.process.description	Description du workflow en cours.
workflow.workItem.label	Libellé du bon de travail en cours.
workflow.workItem.description	Description du bon de travail en cours.
workflow.workItem.offeredTo	Rôle auquel le bon de travail courant a été proposé.
workflow.workItem.allocatedTo	Utilisateur, à qui le bon de travail en cours a été alloué.
workflow.workItem.link	<p>Lien d'accès au bon de travail courant dans la corbeille, au moyen de l'API du composant web.</p> <p>Ce lien ne peut être calculé que si un bon de travail courant est défini et si l'URL est configurée dans Workflow-executions, dans la configuration de mail.</p>
workflow.currentStep.label	Libellé de l'étape courante (tâche autonome, condition ou tâche utilisateur).
workflow.currentStep.description	Description de l'étape courante (tâche autonome, condition ou tâche utilisateur).

Voici un exemple de message : " *Aujourd'hui à \${system.time}, un nouveau bon de travail vous a été proposé.*"

Cela pourra donner le courrier électronique suivant : "*Aujourd'hui à 15:19, un nouveau bon de travail vous a été proposé.*"

Evolution d'un modèle de workflow

Edition

Des améliorations peuvent être apportées au modèle existant. Toutefois il est conseillé, quand vous le publiez, de lui trouver un nom le différenciant de ses prédécesseurs.

Historique

L'historique des images d'un modèle de workflow peut être géré en utilisant la fonctionnalité d'historique accessible en utilisant le bouton *Actions*.

L'historique affiche toutes les images contenant le modèle de workflow et indique si le modèle a été publié. Pour chaque image, il est possible d'exporter ou d'afficher le modèle de workflow correspondant en utilisant le bouton *Actions*.

Suppression

Un modèle peut être supprimé. Cependant toute version publiée auparavant reste accessible dans la section workflow. De plus, si vous recréez un modèle de workflow avec un nom identique, un message vous demandera de confirmer si vous souhaitez remplacer le précédent modèle.

CHAPITRE 26

Modélisation du workflow

Création

Un modèle de workflow peut être créé à partir de la section *Modélisation / Modèle de workflow*. L'assistant de création requiert seulement un nom. Ce nom doit être unique, de manière à identifier chaque modèle de workflow spécifiquement.

Dès lors, le modèle de workflow est créé et les étapes du modèle de workflow sont initialisées par la présence d'une transition initiale. L'objet du modèle de workflow est de décrire l'enchaînement des étapes au delà de cette transition initiale.

Héritage

Il est possible d'utiliser les mécanismes d'héritage avec les limitations suivantes :

- impossibilité d'ajouter des étapes en début ou fin ;
- impossibilité d'insérer des étapes ;
- impossibilité de modifier des liens entre étapes.

Étapes

Un modèle de workflow définit des étapes correspondant à des opérations et des conditions. Les sections suivantes introduisent les différents types d'étape.

Conditions

Deux types de conditions sont disponibles :

- *condition de la bibliothèque* : elle doit être déclarée dans module.xml, et il faut lui définir un libellé, une description et des paramètres pour pouvoir l'utiliser. Dans la définition de la condition, quand l'utilisateur sélectionne une condition de la bibliothèque, les paramètres associés sont affichés dynamiquement.
- *condition spécifique* : il suffit de définir son nom de classe dans la définition de la condition. L'affichage n'est pas dynamique. La classe associée doit appartenir au module de définition du workflow.

Condition de la bibliothèque

Les conditions de la bibliothèque ont la particularité d'étendre les classes **ConditionBean** [ConditionBean]^{API} comme le montre cet **exemple** [package-summary]^{API}.

Il est possible de paramétrer dynamiquement un jeu de variables du bean, si le bean suit la spécification Java Bean. Les variables doivent être déclarées comme paramètres du bean dans module.xml. Le contexte des données n'est pas accessible en Java Bean.

En outre, un libellé et une description peuvent être spécifiés en différentes langues.

EBX5 met à disposition des conditions prédéfinies, grâce auxquelles vous pouvez vérifier :

- si des données sont valides (niveau espace de données, jeu de données ou table) ;
- si un espace de données a été modifié ;
- quelle était la dernière tâche utilisateur acceptée ;
- si deux valeurs sont égales ;
- si une valeur est nulle ou vide ;

Certaines conditions sont marquées comme "obsolètes" car elles ne sont pas compatibles avec I18N. Il est recommandé d'utiliser les nouvelles conditions compatibles avec I18N.

Conditions spécifiques

Les conditions spécifiques ont la particularité d'étendre les classes **Condition** [Condition]^{API} comme le montre l'**exemple** [package-summary]^{API} suivant.

Il n'est pas possible de paramétrer des variables du bean dynamiquement pour des conditions spécifiques. Toutefois, le contexte des données est accessible dans le Java Bean.

Tâche utilisateur

La modélisation d'une tâche utilisateur fait l'objet d'un [chapitre](#) particulier.

Tâche autonome

Il existe deux types de tâches autonomes :

- *tâche autonome de la bibliothèque* : elle doit être déclarée dans module.xml et il faut lui définir un libellé, une description et des paramètres pour pouvoir l'utiliser. Lors de la définition d'une tâche autonome, quand un utilisateur sélectionne une tâche autonome de la bibliothèque, ses paramètres associés s'affichent dynamiquement.
- *tâche autonome spécifique* : il suffit de définir le nom de la classe dans la définition de la tâche autonome. L'affichage n'est pas automatique. La classe associée doit appartenir au module de définition du workflow.

Tâche autonome de la bibliothèque

Les tâches autonomes de la bibliothèque ont la particularité d'étendre les classes **ScriptTaskBean** [ScriptTaskBean]^{API}.

La méthode **executeScript()** [ScriptTaskBean]^{API} est appelée, quand le processus atteint une étape clé, comme dans cet **exemple** [package-summary]^{API}. Cette méthode ne doit lancer aucun Thread ; en effet, le moteur de workflow passera à l'étape suivante quand le script se terminera. L'exécution d'une partie du script en mode asynchrone (via le Thread) ne garantit donc pas sa terminaison avant le passage à l'étape suivante.

Il est possible de paramétrer dynamiquement les variables du bean, si le bean suit la spécification Java Bean. Les variables doivent être déclarées comme paramètres du bean dans le module.xml. Le contexte des données n'est pas accessible en Java Bean.

Par ailleurs, un libellé et une description peuvent être spécifiés en différentes langues.

EBX5 inclut des tâches autonomes prédéfinies, comme :

- créer un espace de données,
- fermer un espace de données,
- créer une image,
- fusionner un espace de données,
- importer une archive,
- envoyer un courrier électronique.

Certaines tâches autonomes sont marquées comme "obsolètes" car elles ne sont pas compatibles avec I18N. Il est recommandé d'utiliser les nouvelles tâches compatibles avec I18N.

Tâches autonomes spécifiques

Les tâches autonomes spécifiques ont la particularité d'étendre les classes **tâche autonome** [ScriptTask]^{API}.

La méthode **executeScript()** [ScriptTask]^{API} est appelée, quand le processus atteint une étape clé, telle que cet **exemple** [package-summary]^{API}. Cette méthode ne doit lancer aucun Thread, car le processus continue, quand la méthode s'arrête. Ainsi, toute opération doit être synchrone dans cette méthode.

Il n'est pas possible de paramétrer dynamiquement les variables du bean pour des tâches autonomes spécifiques. Toutefois, le contexte des données est accessible en Java Bean.

Contexte des données

Un contexte de données est lié à chaque workflow. Ce contexte de données peut être utilisé pour définir des variables, qui pourront être utilisées en entrée et/ou en sortie dans différentes étapes du workflow.

Quelques paramètres ajustables

Informations

Les informations d'un modèle de données peuvent être éditées à l'aide du bouton *Actions* dans le panneau de navigation. Les informations éditables sont les suivantes :

Champs éditables	Contenu requis
Propriétaire	Personne pouvant, en tant que propriétaire, éditer les informations du modèle de workflow et y définir des règles de permissions.
Documentation locale	Libellé et description associés au modèle de workflow en différentes langues.
Activation	Un modèle de workflow doit être activé pour pouvoir être publié.

Paramétrage spécifique au modèle

La configuration d'un modèle de workflow est accessible depuis le panneau de navigation. Les propriétés principales sont les suivantes :

Options de configuration	Contenu requis
Module	Module contenant les déclarations de scripts et les ressources Java spécifiques.
Notification de démarrage	Liste des profils qui doivent recevoir une notification (choisie dans la liste des modèles de messages) quand un workflow démarre.
Notification de fin	Liste des profils qui doivent recevoir une notification (choisie dans la liste des modèles de messages), quand un workflow se termine.
Priorité	<p>Par défaut, chaque workflow associé à ce modèle sera lancé avec cette priorité. Cette valeur est facultative. Si aucune valeur n'est définie ici, et une priorité par défaut est définie pour le référentiel, la priorité par défaut pour le référentiel sera appliquée à tous les workflows et bons de travail sans priorité. Voir Priorité de bons de travail pour plus d'informations.</p> <p>Note : Seuls les utilisateurs définis en tant qu'administrateurs de workflows seront autorisés à modifier la priorité des workflows de données associés manuellement.</p>
Permissions	Définit les permissions pour différentes tâches du workflow (voir section ci-dessous).
Permissions programmatiques	Définit le composant gérant les permissions. S'il est défini, il remplace l'ensemble des permissions ci-dessus.

Permission sur une publication de workflow

Pour trouver comment définir des permissions sur une publication de workflow, voir ci-dessous :

Permissions	Définition des profils autorisés
Administration de workflow	Permet de réaliser des tâches administratives sur une publication de workflow basée sur un modèle de workflow. L'autorisation de tâches spécifiques peut être définie à l'aide du lien <i>Voir la configuration avancée</i> .
Gestion des allocations	Permet d'allouer, désallouer ou réallouer des bons de travail à des utilisateurs. Une autorisation pour une opération spécifique peut être définie à l'aide du lien <i>Voir la configuration avancée</i> .
Lancement de workflow	Permet de lancer un workflow. Si aucun profil n'est spécifié, cela signifie qu'il est possible de créer un nouveau workflow exclusivement de manière programmatique (par exemple, à l'aide de triggers).
Monitoring de workflow	Permet de voir un workflow en cours, même s'il n'est pas directement concerné par les bons de travail courants du workflow. Un tel utilisateur peut également voir l'historique d'un workflow terminé.

Note : Un utilisateur, qui n'a aucun privilège particulier attribué, pourra voir un bon de travail uniquement, si celui-ci lui est proposé ou personnellement alloué.

Voir aussi : [Administration d'un workflow](#)

CHAPITRE 27

Tâche utilisateur

Libellé et description

Le libellé et la description de la tâche utilisateur peuvent être spécifiés en différentes langues, afin de renseigner l'utilisateur sur la nature et le contenu de celle-ci

Définition

Profils


Les profils designent les rôles et les utilisateurs auxquels la tâche utilisateur est destinée. Un bon de travail est créé pour chaque profil spécifié.

Si un profil se rapporte à un utilisateur au lieu d'un rôle, le bon de travail lui est directement alloué.

Service à appeler

EBX5 propose nativement les services suivants :

- Accès à un contenu particulier (data space, data set, table, record, etc.).
- Créer un nouvel enregistrement.
- Valider un espace de données, une image ou un jeu de données.
- Fusionner un espace de données.
- Comparer les contenus.
- Exporter les données d'une table vers des fichiers XML.
- Exporter les données d'une table vers des fichiers CSV.
- Importer des données d'un fichier XML vers une table.
- Importer des données d'un fichier CSV vers une table.

Chaque service requiert des paramètres spécifiques ; Un assistant permet de sélectionner des espaces de données, des images ou des valeurs du contexte des données : .

Voir aussi : [Services prédéfinis de EBX5](#)

Terminaison

Critère de fin de tâche

Une tâche utilisateur peut être assignée à plusieurs *participants* et générer plusieurs bons de travail durant l'exécution du workflow. Lors de la définition d'une tâche utilisateur dans le modèle de workflow, il est possible de sélectionner un des critères prédéfinis qui déterminent quand une tâche utilisateur est terminée, en se basant sur le statut des bons de travail associés. Lorsque la condition de sortie de la tâche utilisateur sera remplie, le workflow de données avancera jusqu'à l'étape suivante définie dans le modèle.

Par exemple, pour le cas d'une tâche utilisateur qui demande la validation d'un enregistrement d'un produit, il est possible de désigner trois participants. Le critère de fin de tâche permet de préciser si l'enregistrement du produit doit être validé par les trois participants, ou uniquement par le premier utilisateur à répondre.

Le critère de fin de tâche par défaut est "Quand tous les bons de travail ont été acceptés".

Remarque : Si une extension de service surcharge la méthode `UserTask.handleWorkItemCompletion` pour gérer la fin de la tâche utilisateur, c'est à la charge du développeur de l'extension de vérifier dans le code de cette méthode que le critère de fin de tâche définie dans l'interface a été satisfait. Voir **`UserTask.handleWorkItemCompletion()`** [`UserTask`]^{API} dans la Javadoc pour plus d'informations.

Tolérance au rejet

Par défaut, si un utilisateur rejette un bon de travail durant l'exécution d'un workflow, la tâche utilisateur est positionnée en erreur et l'avancement du workflow est stoppé.

Pour changer ce comportement par défaut, il est possible de définir un nombre de bons de travail rejetés à tolérer. Tant que la limite de rejets tolérés n'est pas dépassée, aucune erreur ne se produit et c'est le critère de fin de tâche qui détermine quand la tâche utilisateur est terminée.

Les critères de fin de tâche suivant tolèrent automatiquement tous les rejets :

- 'Quand tous les bons de travaux ont été acceptés ou rejetés'
- 'Quand tous les bons de travail ont été acceptés, ou dès qu'un bon de travail a été rejeté'

Libellés personnalisés

Durant l'exécution des tâches utilisateur, l'utilisateur peut accepter ou rejeter son bon de travail en cliquant sur le bouton correspondant. Dans la modélisation de workflow, il est possible pour certaine tâche utilisateur de définir un libellé et un message de confirmation personnalisés pour ces

boutons. Cette fonctionnalité est particulièrement utile pour ajouter une signification particulière à l'action d'accepter ou rejeter un bon de travail.

Demande de confirmation activée

Par défaut, quand un utilisateur enregistre sa décision en cliquant sur le bouton accepter ou rejeter, une demande de confirmation est affichée.

Il est possible de désactiver cette demande de confirmation de la décision en positionnant ce champ à faux.

Extension

Il est possible d'étendre programmatiquement le comportement de la tâche afin que cette dernière s'insère plus finement dans le contexte du workflow, par exemple si un comportement spécifique est nécessaire, soit pour la création du bon de travail ou pour achever la tâche de l'utilisateur.

La règle spécifiée est une classe JavaBean qui doit étendre la classe **UserTask** [UserTask]^{API}.

Attention : si une règle est spécifiée et la méthode `handleWorkItemCompletion` surchargée, la stratégie de fin n'est plus automatiquement contrôlée. C'est au développeur de la vérifier dans cette méthode.

Notification

Une notification par courrier électronique peut être envoyée aux utilisateurs quand des événements spécifiques se produisent. Pour chaque événement, vous pouvez spécifier un message type à utiliser.

En outre, il est possible de définir un profil, auquel une copie de chaque courrier envoyé sera transmise.

Voir aussi : [Modèles de message](#)

Relance

Des courriers électroniques de relance pour les bons de travail proposés ou alloués et inachevés peuvent être envoyés aux utilisateurs concernés de manière périodique.

Le contenu des courriers de relance dépend de l'état courant du bon de travail. En effet, si le bon de travail est proposé, la notification utilisera le modèle de courrier électronique "Bons de travail proposés" ; si le bon de travail est alloué, la notification utilisera le modèle "Bons de travail alloués".

Echéance

Une tâche utilisateur peut avoir une échéance. Quand cette date est passée et si les bons de travail associés n'ont pas été complétés, un courrier électronique spécifique est envoyé aux utilisateurs concernés. Ce courrier électronique va alors être réenvoyé tous les jours jusqu'à l'achèvement de la tâche.

Il y a deux types d'échéances :

- *Echéance absolue* : une date du calendrier.
- *Echéance relative* : durée (en heures, jours ou mois). La durée est évaluée à partir d'une date de référence : début d'une tâche utilisateur ou début d'un workflow.

CHAPITRE 28

Publication d'un modèle de workflow

Les modèles de workflow deviennent exécutables uniquement après publication.

Cela est réalisé en créant une image, où la publication de workflow sera stockée. Ce mécanisme assure la stabilité d'une publication de workflow pendant l'exécution de workflow associés.

Une publication de modèle de workflow peut être accédé à l'aide du bouton *Publier* dans le panneau de navigation.

Etape de publication

Sélection d'un modèle de workflow

Il est possible de publier plusieurs modèles de workflow en une fois. Quand le service de publication est appelé, les modèles de workflow existant s'affichent. L'utilisateur doit sélectionner le ou les modèles de workflow, qu'il souhaite publier.

Note : un modèle de workflow doit être activé pour pouvoir être publié.

Informations sur les images et les publications de workflow

Un libellé et une description peuvent être spécifiés pour l'image à créer. Le libellé par défaut est la date et l'heure de publication et la description par défaut indique, qui a publié le modèle de workflow.

Pour chaque modèle de workflow sélectionné, le nom de publication doit être renseigné et unique. quand un modèle de workflow a déjà été publié, il est possible de le mettre à jour. Les noms des publication de workflow disponibles associées au même modèle de workflow s'affiche. Il est possible d'en sélectionner une pour réutiliser une publication de workflow existante. Auquel cas, l'ancienne publication de workflow devient inaccessible.

Workflows de données

CHAPITRE 29

Introduction aux workflows de données

Présentation

Un workflow de données est un processus exécuté sous la forme d'une succession d'étapes, définie par une publication de modèle de workflow. Le workflow de données permet aux utilisateurs, ainsi qu'aux procédures automatisées, d'effectuer des actions de façon collaborative sur les données. Une fois le modèle de workflow spécifié et publié, la publication résultante peut être utilisée pour lancer un workflow de données afin d'exécuter les étapes définies.

En fonction des permissions définies par le modèle de workflow, un utilisateur peut effectuer une ou plusieurs des actions suivantes sur les workflows de données associés :

- en tant qu'utilisateur avec les permissions par défaut, effectuer les actions attendues par les bons de travail qui lui sont destinés,
- en tant qu'utilisateur avec les permissions pour lancer les workflows, créer les nouveaux workflows de données depuis une publication du modèle de workflow,
- en tant qu'utilisateur avec les permissions de monitoring de workflow, suivre l'avancement des workflows de données en cours, et consulter l'historique des workflows de données terminés.
- en tant que gestionnaire d'allocation des bons de travail, modifier les allocations des bons de travail des autres utilisateurs manuellement.
- en tant qu'administrateur de workflow, effectuer différentes actions d'administration, comme par exemple, redémarrer une étape, terminer un workflow en cours ou rendre une publication indisponible pour le lancement de workflows.

Voir aussi :

- [*Bons de travail*](#)
- [*Lancement et monitoring de workflows de données*](#)
- [*Administration de workflows de données*](#)
- [*Permission sur une publication de workflow*](#)

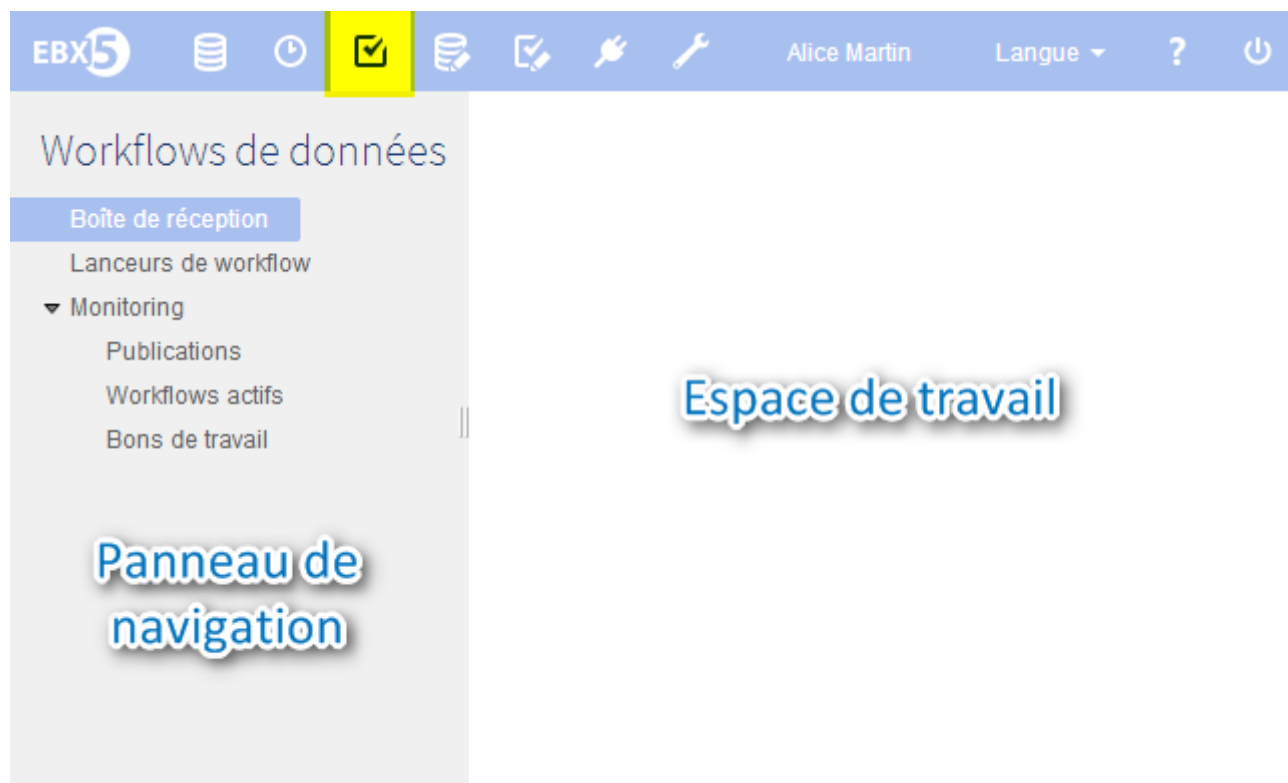
Concepts apparentés : [*Modèles de workflow*](#)

CHAPITRE 30

Utilisation de l'interface utilisateur de la section Workflow de données

Navigation dans l'interface utilisateur

La fonctionnalité workflows de données se trouve dans la section **Workflows de données** dans l'interface utilisateur d'EBX5.

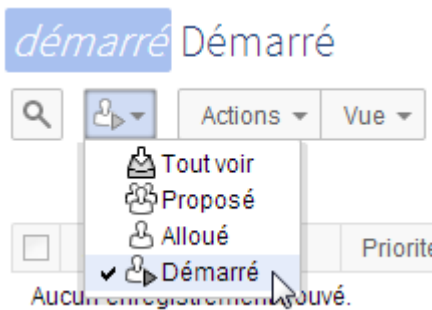


Le panneau de navigation est composé des sections suivantes :


Boîte de réception des bons de travail	Tous les bons de travail qui vous sont alloués ou proposés, pour lesquels vous devez effectuer les actions spécifiées.
Lanceurs de workflow	Liste des publications de workflow à partir desquelles vous pouvez lancer des workflows de données, en fonction de vos permissions.
Monitoring	Vues pour le monitoring des workflows de données en cours pour lesquels vous avez les permissions nécessaires de visualisation.
Publications	Publications pour lesquelles vous avez les permissions nécessaires de visualisation. Si vous disposez de permissions d'administration supplémentaires, vous pouvez également désactiver la possibilité de lancer les workflows de données depuis une publication à partir de cette vue.
Workflows actifs	Workflows de données en cours d'exécution pour lesquels vous avez les permissions nécessaires de visualisation. Si vous disposez de permissions d'administration supplémentaires, vous pouvez également effectuer des actions d'administration à partir de cette vue, comme par exemple redémarrer une étape, ou terminer un workflow en cours.
Bons de travail	Bons de travail pour lesquels vous avez les permissions nécessaires de visualisation. Si vous disposez de permissions d'administration supplémentaires, vous pouvez également effectuer des actions d'administration à partir de cette vue, comme par exemple allouer les bons de travaux aux utilisateurs ou rôles.
Workflows terminés	Workflows de données qui ont fini de s'exécuter, pour lesquels vous avez les permissions nécessaires de visualisation. Si vous disposez de permissions d'administration supplémentaires, vous pouvez également nettoyer les workflows terminés à partir de cette vue.

Filtrage d'items dans les vues

Dans certaines vues, comme "Boîte de réception des bons de travail", vous pouvez filtrer les lignes affichées dans les tables en fonction de leur état. Dans ces vues, un menu est disponible à cet effet pour sélectionner l'état correspondant au filtre attendu.



Vue graphique d'un workflow de données

En tant qu'utilisateur avec un bon de travail à effectuer, ou en tant que superviseur ou administrateur de workflow de données, vous pouvez suivre l'avancement ou l'historique d'exécution d'un workflow de données en cliquant sur le bouton "Afficher"  dans la colonne "Workflow de données" des tables de l'interface de workflows de données. Ce bouton ouvre une pop-up qui affiche une vue interactive graphique de l'exécution du workflow de données. Dans cette vue, vous pouvez suivre l'avancement global de l'exécution, et sélectionner une étape individuelle afin de consulter le détail de ses informations.

CHAPITRE 31

Bons de travail

Présentation des bons de travail

Un bon de travail est une tâche utilisateur unitaire qui doit être effectuée par un utilisateur humain. Par défaut, quand un modèle de workflow définit une tâche utilisateur, les workflows de données qui sont lancés depuis les publications du modèle génèrent un bon de travail individuel pour chacun des participants listés dans la tâche utilisateur.

Etats d'un bon de travail

Quand un workflow de données émet un bon de travail pendant son exécution pour une tâche utilisateur définie dans le modèle de workflow, le bon de travail peut prendre plusieurs états : proposé, alloué, démarré, et terminé.

Par défaut, pour chaque utilisateur défini comme participant de la tâche utilisateur, le workflow de données crée un bon de travail dans l'état *alloué*. L'utilisateur défini peut directement commencer à travailler sur le bon de travail alloué en effectuant l'action 'Prendre et démarrer', et le bon de travail passe alors à l'état *démarré*.

Par défaut, pour chaque rôle défini comme participant de la tâche utilisateur, le workflow de données crée un bon de travail dans l'état *proposé*. Tout membre du rôle peut prendre le bon de travail en utilisant l'action "Prendre et démarrer", le passant ainsi à l'état *démarré*.

Avant qu'un utilisateur ait pris le bon de travail proposé, un gestionnaire du workflow de données peut intervenir pour affecter manuellement le bon de travail à un utilisateur spécifique, plaçant ainsi le bon de travail dans l'état *alloué*. Puis, lorsque l'utilisateur commence à travailler sur le bon de travail via l'action "Démarrer le bon de travail", le bon de travail passe à l'état *démarré*.

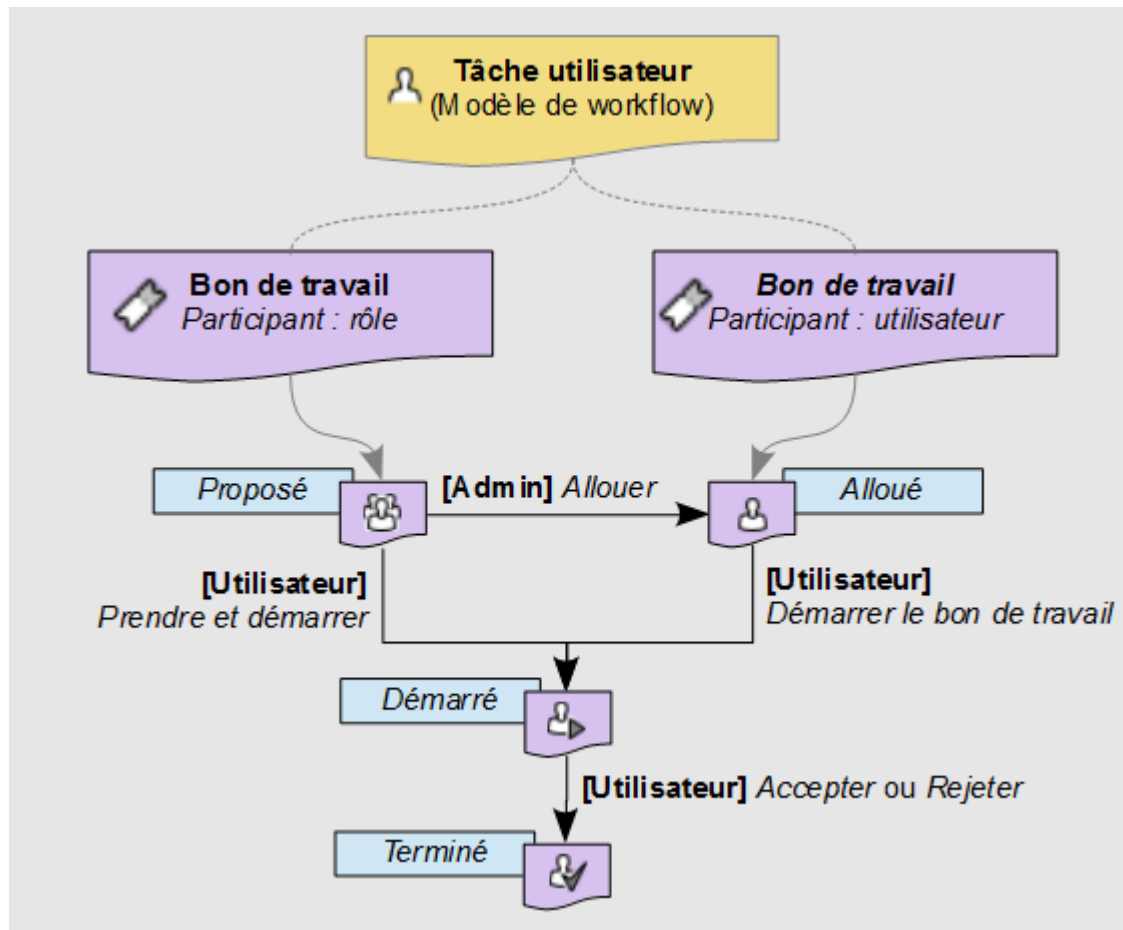
Note

Le comportement par défaut décrit ci-dessus peut être surchargé par une extension programmatique définie dans la tâche utilisateur. Dans ce cas, les bons de travail peuvent être générés programmatiquement, sans se baser obligatoirement sur la liste des participants de la tâche utilisateur.

Une fois que l'utilisateur qui a démarré le bon de travail a réalisé l'action demandée, l'action terminale "Accepter" ou "Rejeter" place le bon de travail dans l'état *terminé*. Lorsqu'un utilisateur finit un bon de

travail, le workflow de données passe automatiquement à l'étape suivante définie dans le modèle de workflow.


Diagramme des états de bon de travail



Travail sur un bon de travail en tant que participant

Tous les bons de travail qui vous sont destinés en tant qu'utilisateur (qui vous sont soit proposé, soit alloué), sont affichés dans votre boîte de réception des bons de travail. Quand vous commencez à travailler sur un bon de travail, vous pouvez ajouter des commentaires associés qui seront visibles par les autres participants du workflow de données, et les administrateurs et superviseurs du workflow. Tant que vous êtes toujours en train de travailler sur le bon de travail, vous pouvez éditer ce commentaire.

Quand vous avez réalisé toutes les actions demandées par le bon de travail, vous devez signaler la fin du travail en cliquant soit sur le bouton **Accepter**, soit sur le bouton **Rejeter**. Les libellés de ces deux boutons peuvent varier en fonction du contexte du bon de travail.

Pour suivre l'avancement du workflow de données associé au bon de travail qui vous est destiné dans votre boîte de réception, cliquez sur le bouton "Afficher"  dans la colonne 'Workflow de données' de la table. Une pop-up affichera une vue graphique interactive du workflow de données jusqu'au moment actuel ainsi que les étapes à venir. Vous pouvez visualiser les détails d'une étape en sélectionnant l'étape.

Note

Si vous interrompez la session actuelle pendant un bon de travail, par exemple en fermant le navigateur ou en déconnectant, l'état actuel du bon de travail est préservé. Quand vous revenez sur le bon de travail, il continues à partir du même point.

Priorité de bons de travail

Les bons de travail peuvent porter une priorité, qui peut être utile pour trier et filtrer les bons de travail à compléter. La priorité d'un bon de travail est définie au niveau de son workflow de données, et n'est pas spécifique au bon de travail lui-même. Par conséquent, si le workflow de données est considéré comme urgent, tous les bons de travail ouverts associés sont aussi considérés comme urgent. Par défaut, il y a six niveaux de priorité, de "Très peu prioritaire" à "Urgent". Cependant, la représentation visuelle et le nommage des priorités dépendent de la configuration de votre référentiel EBX5.

Voir aussi : [tâche utilisateur \(glossaire\)](#)

Concepts apparentés : [Tâche utilisateur](#)

CHAPITRE 32

Lancement et monitoring de workflows de données

Lancement d'un workflow de données

Quand un modèle de workflow vous autorise à lancer des workflows de données depuis ses publications, vous pouvez créer de nouveaux workflows de données en utilisant la vue "Lanceurs de workflow" dans le panneau de navigation. Pour créer un nouveau workflow de données depuis une publication d'un modèle de workflow, cliquez sur le bouton **Lancer** de la ligne de la publication cible dans la table.

Vous pouvez alors définir des libellés et descriptions localisés pour le nouveau workflow de données que vous lancez.

Activités de monitoring

Quand un modèle de workflow vous donne les permissions de monitoring de workflow, vous avez la possibilité de suivre l'avancement des workflows de données qui sont en cours d'exécution. Vous pouvez accéder aux vues de monitoring dans la section "Monitoring" du panneau de navigation. Si vous disposez également de permissions d'administration de workflow, vous pouvez effectuer les actions autorisées associées depuis ces vues.

Lorsque un workflow de données que vous êtes autorisé à suivre a fini son exécution, il est affiché dans "Workflow de données terminés", où vous pouvez consulter son historique.

Gestion de l'allocation des bons de travail

Quand un modèle de workflow vous donne les permissions de gestion de l'allocation des bons de travail, vous pouvez intervenir pour allouer manuellement des bons de travail durant l'exécution des workflows de données associés au modèle de workflow. Dans ce cas, vous pouvez effectuer une ou plusieurs des actions ci-dessous sur les bons de travail.

Note

Ces actions ne sont pas disponibles pour les bons de travail qui ne définissent pas un rôle pour proposer le bon de travail (uniquement un utilisateur spécifique).

Sélectionnez "Bons de travail" dans la section "Monitoring" du panneau de navigation. Les actions que vous pouvez effectuer sont affichées dans le menu **Actions** de l'entrée du bon de travail, selon son état actuel, dans la table.

Allouer	Allouer un bon de travail à un utilisateur spécifique. Cette action est disponible pour les bons de travail dans l'état <i>proposé</i> .
Désallouer	Remettre un bon de travail qui est actuellement dans l'état <i>alloué</i> dans l'état <i>proposé</i> .
Réalouer	Modifier l'utilisateur à qui le bon de travail est alloué. Cette action est disponible pour les bons de travail dans l'état <i>alloué</i> .

Voir aussi :

- [Bons de travail](#)
- [Permission sur une publication de workflow](#)

Concepts apparentés : [Modèles de workflow](#)

CHAPITRE 33

Administration de workflows de données

Si vous disposez de permissions pour administrer des workflows de données, toutes les publications, workflows actifs, et bons de travail associés seront affichés dans les différentes vues de la section "Monitoring" dans le panneau de navigation. Dans ces vues de monitoring, vous pouvez accéder directement aux menus **Actions** sur les lignes des tables pour effectuer les actions d'administration.

Note

Quand un modèle de données vous donne des droits d'administration, vous aurez automatiquement les permissions de monitoring sur tous les objets associés à l'exécution de workflow, comme les publications, les workflows actifs, et les bons de travail.

Présentation de l'exécution de workflow de données

Quand un workflow de données est lancé, un *jeton* qui représente l'étape en cours d'exécution est créé et positionné au début du workflow. A chaque fois qu'une étape est terminée, ce jeton se déplace sur la prochaine étape définie par le modèle de workflow associé à la publication du workflow de données.

Pendant l'exécution d'un workflow de données, le jeton est positionné sur un des types d'étape suivants:

- une tâche automatique, qui est lancée automatiquement et n'a pas besoin d'interaction utilisateur. La tâche automatique est terminée quand les actions définies finissent leur exécution.
- une tâche utilisateur, qui génère un ou plusieurs bons de travail effectués manuellement par les utilisateurs. Chaque bon de travail est terminé par une action "Accepter" ou "Rejeter", réalisée explicitement par l'utilisateur. La fin de la tâche utilisateur chapeau est déterminée en fonction du critère de fin de tâche défini pour la tâche utilisateur dans le modèle de workflow.
- une condition, qui est évaluée automatiquement afin de déterminer l'étape suivante de l'exécution du workflow de données.

Le jeton peut être dans les états suivants :

- **A exécuter** : Le jeton est en train de passer à la prochaine étape, en se basant sur le modèle de workflow.
- **En cours d'exécution** : Le jeton est positionné sur une tâche automatique ou une condition en train de s'exécuter.
- **Utilisateur** : Le jeton est positionné sur une tâche utilisateur et attend une action utilisateur.
- **Terminé** : Le jeton a atteint la fin du workflow de données.
- **Erreur** : Une erreur est survenue.

Voir aussi : [Data workflow administration](#)

Actions d'administration de workflow de données

Actions sur les publications

Désactivation d'une publication de workflow

Afin d'éviter que de nouveaux workflows de données soient lancés depuis une publication de workflow, vous pouvez désactiver la publication. Sélectionnez la vue "Publications" dans la panneau de navigation, puis sélectionnez **Actions > Désactiver** sur la ligne de la publication cible.

Une fois désactivée, la publication n'apparaîtra plus dans la vue "Lanceurs de workflow" des utilisateurs. Toutefois, les workflows de données déjà lancés vont continuer à s'exécuter.

Note

Suite à la désactivation d'une publication, il n'est pas possible de la réactiver à partir de la section "Workflows de données". Seul un utilisateur avec le rôle built-in "Administrateur" peut réactiver une publication inactive dans la section "Administration". Cependant, il n'est pas conseillé de modifier les tables techniques manuellement, car il est important de préserver l'intégrité des données techniques des workflows.

Dépublication d'une publication de workflow

Si une publication de workflow n'est plus utilisée, vous pouvez la supprimer de toutes les vues de la section "Workflows de données" en la dépubliant. Pour faire cela,

1. Désactivez la publication de workflow afin d'éviter que des utilisateurs continuent de lancer des nouveaux workflows de données sur cette publication, en suivant le processus décrit dans la section [Désactivation d'une publication de workflow](#).
2. Dépublier la publication de workflow en sélectionnant **Actions > Dépublier** de la ligne de la publication cible.

Note

Quand vous décidez de dépublier une publication de workflow, une confirmation vous sera demandée pour terminer et purger tous les workflows de données en cours qui ont été lancés depuis cette publication de workflow, ainsi que tous bons de travail associés. Toute perte de données résultante d'une fin prématurée est alors définitive.

Actions sur workflows de données

Ré-exécution d'une étape

Dans le cas d'une erreur inattendue pendant l'exécution d'une étape, par exemple, à cause d'un problème de permissions ou de ressources non disponibles, vous pouvez "rejouer" une étape en tant qu'administrateur de workflow. En rejouant une étape, l'environnement d'exécution associé est nettoyé, notamment les bons de travail liés, et le jeton est repositionné au début de l'étape courante.

Pour rejouer l'étape courant dans un workflow de données, sélectionnez **Actions > Rejouer l'étape** dans la ligne du workflow cible dans la table 'Workflows actifs'.

Forcer un workflow de données actif à se finir

Pour terminer un workflow de données en cours d'exécution, sélectionnez **Actions > Terminer et purger** dans la ligne du workflow cible dans la table 'Workflows actifs'. L'action stoppe l'exécution du workflow de données et supprime le workflow de données et tous les bons de travail associés.

Purge d'un workflow de données terminé

Quand un workflow de données a terminé son exécution, son historique est visible pour ses superviseurs et administrateurs dans la vue 'Workflows terminés'. Pour purger le workflow terminé et son historique, vous pouvez effectuer un nettoyage en sélectionnant **Actions > Purger** dans la ligne du workflow cible de la table 'Workflows terminés'.

Modification de la priorité d'un workflow de données

Suite au lancement d'un workflow de données, un administrateur du workflow peut modifier son niveau de priorité. En modifiant la priorité du workflow de données, la priorité de tous les bons de travail existants et à venir de ce workflow sera modifiée. Pour modifier la priorité d'un workflow de données, sélectionnez **Actions > Modifier la priorité** dans la ligne du workflow cible dans la table 'Workflows actifs'.

Voir aussi : [Permission sur une publication de workflow](#)

Services de données

CHAPITRE 34

Introduction aux services de données

Les services de données offrent la possibilité d'accéder et d'interagir avec EBX5 pour :

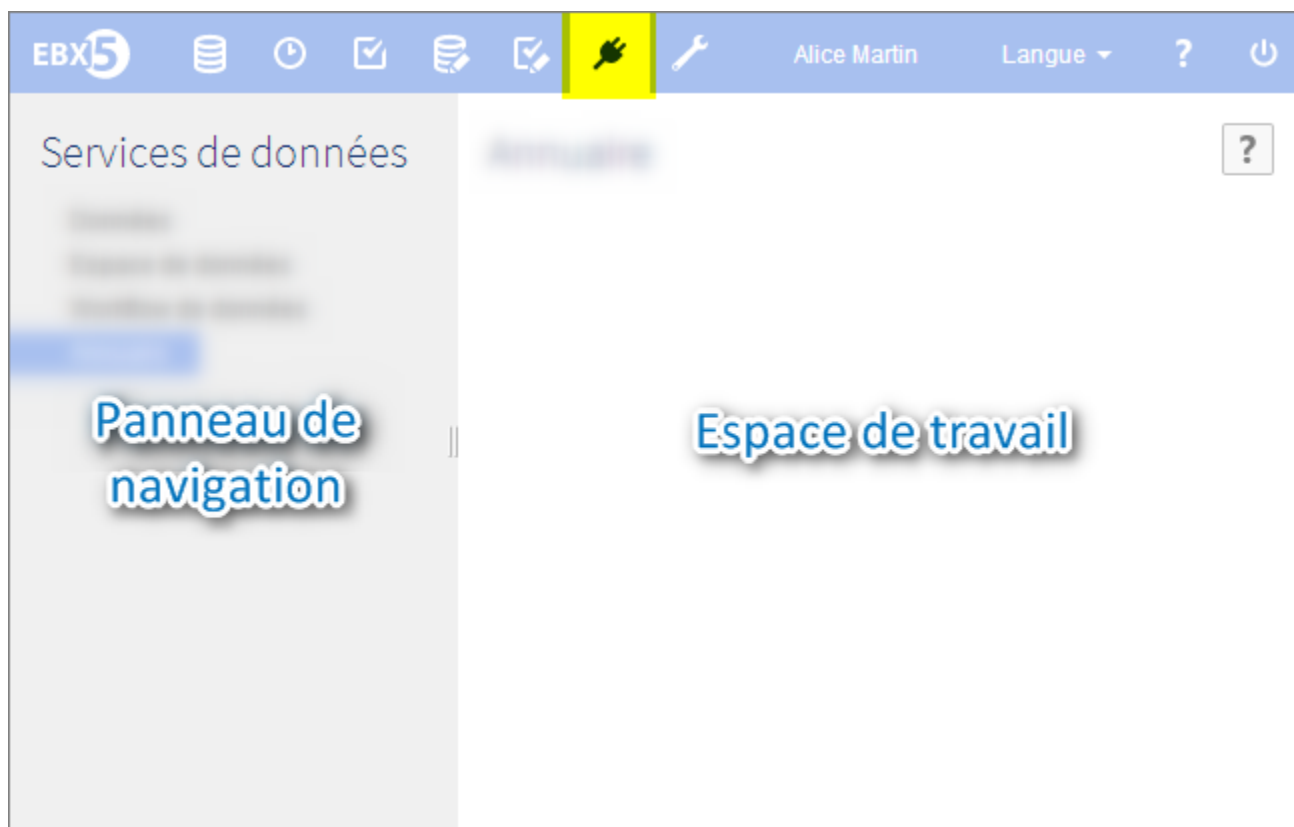
- accéder aux données stockées dans les tables (voir [données](#)),
- interagir avec les espaces de données (voir [espace de données](#)),
- contrôler les workflows (voir [workflow](#)),
- définir un lignage vers une logiciel tiers (voir [lignage](#)).

Un service de données est un service web standard, qui donne accès à toutes les fonctionnalités et données (créer un espace de données, mettre à jour les enregistrements d'un jeu de données, etc.).

Si vous vous demandez toujours à quoi ces termes techniques se réfèrent, n'hésitez pas à aller à la page [glossaire](#).

Vue d'ensemble des services de données

Modalités d'accès dans l'interface



Concepts et outils associés

Services de données standard	Services de données permettant d'accéder à un jeu de données, un espace de données ou un workflow en particulier
Lignage	Services de données prenant en considération les permissions utilisateur et évitant de possibles erreurs, en cas d'accès à des données non visibles de l'utilisateur actuellement identifié.

Pour en savoir plus sur la norme WSDL (Web Services Description Language), veuillez consulter le site du [World Wide Web Consortium \(W3C\)](http://www.w3.org/).

Voir aussi :

- [Modèles de données](#)
- [Espaces de données](#)
- [Jeux de données](#)

- [*Modèles de workflow*](#)
- [*Workflows de données*](#)

CHAPITRE 35

Actions principales

Générer un WSDL pour accéder aux données

La génération de WSDL pour l'accès de données est possible par sélection de *Données* dans la section *Services de données*.

Les étapes de la génération d'un WSDL sont les suivantes :

1. Sélectionnez un espace de données
2. Entrez l'identifiant d'un jeu de données (ou nom unique) et cliquez sur suivant
3. Sélectionnez les opérations autorisées sur chaque table
4. Téléchargez le WSDL généré

Opérations disponibles sur la table d'un jeu de données

- *Sélectionner un (ou plusieurs) enregistrement(s)*
- *Insérer un (ou plusieurs) enregistrement(s)*
- *Mettre à jour un (ou plusieurs) enregistrement(s)*
- *Supprimer un (ou plusieurs) enregistrement(s)*
- *Compter des enregistrements*
- *Récupérer les changements entre espaces de données ou images* : vous donne un résumé des différences entre une table et son équivalent, s'il existe, dans une image ou un autre espace de données
- *Réaliser plusieurs opérations sur les tables d'un jeu de données.*

Générer un WSDL pour accéder à un espace de données

La génération de WSDL pour la manipulation d'un espace de données est accessible par sélection de *Espace de données* dans la section *Services de données*. Le WSDL généré n'est pas spécifique

à un espace de données et aucune information n'est requise. Il peut être téléchargé grâce au bouton *Télécharger un WSDL* situé dans l'espace de travail.

Opérations disponibles

- *Créer un espace de données.*
- *Fermer un espace de données.*
- *Créer l'image d'un espace de données.*
- *Fermer l'image d'un espace de données.*
- *Fusionner un espace de données.*
- *Valider un espace de données ou une image d'espace de données.*
- *Valider un jeu de données.*

Générer un WSDL pour contrôler un workflow

La génération d'un WSDL pour le contrôle d'un workflow est accessible par sélection de *Workflow* dans la section *Services de données*. Le WSDL généré n'est pas spécifique à une publication de workflow donnée et aucune information n'est requise. Il peut être téléchargé grâce au bouton *Télécharger le WSDL* situé dans l'espace de travail.

Opérations disponibles

- *Démarrer un workflow.*
- *Finir un workflow.*

Générer un WSDL pour un lignage

La génération d'un WSDL pour un lignage est accessible par sélection de *Lignage* dans la section *Services de données*, sous réserves que des profils aient été autorisés par un profil administrateur dans la section *Administration > Lignage*.

Les WSDL générés pour accéder aux tables sont les mêmes que ceux utilisés pour la [génération d'un WSDL d'accès aux données](#).

Les étapes de la génération de ce WSDL sont les suivantes :

1. Sélectionnez un rôle ou un utilisateur, dont les permissions seront appliquées. Veuillez noter qu'un rôle ou un utilisateur doit être autorisé à être utilisé pour le lignage par un administrateur.
2. Sélectionnez un espace de données.
3. Entrez l'identifiant du jeu de données (ou nom unique) et cliquez sur suivant.
4. Sélectionnez les tables et les opérations autorisées.
5. Téléchargez le WSDL généré.

Manuel de référence (en anglais)

Intégration

CHAPITRE 36

Overview of integration and extension

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

Using EBX5 as a web component

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

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

Voir aussi : [Using EBX5 as a web component](#)

User interface customization

User interface services (UI services)

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

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

Voir aussi : [UI service reference page](#)

Custom layout

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

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

Voir aussi : **UIForm** [UIForm]^{API}

UI beans

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

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

- Masking characters in password fields
- Incorporating JavaScript UI components

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

Voir aussi :

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

Ajax components

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

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

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

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

Voir aussi : ***UIAjaxComponent*** [*UIAjaxComponent*]^{API}

Data services

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

Voir aussi : [Data Services reference page](#)

XML and CSV import/export services

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

Voir aussi :

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

Programmatic services

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

Some examples of programmatic services include:

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

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

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

CHAPITRE 37

Using EBX5 as a web component

Overview

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

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

Voir aussi : [Supported web browsers](#)

Integrating EBX5 web components into applications

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

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

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

2. A Java application, for example:

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

Note

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

Repository element and scope selection

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

The repository elements that can be selected are as follows:

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

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

Voir aussi : [Scope](#)

Request specifications

Base URL

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

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

Note

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

User authentication and session information parameters

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

Entity and service selection parameters

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

Layout parameters

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

Example calls to an EBX5 web component

Minimal URI:

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

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

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

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

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

Selects the roles table in the default directory:

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

Selects the record 'admin' in the default directory:

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

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

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

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

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

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

CHAPITRE 38

User interface services (UI services)

Overview

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

The following types of UI services exist:

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

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

Accessing UI services

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

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

UI services on data sets

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

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

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

Common use cases

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

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

Declaration and configuration

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

Note

The data model must be packaged in a module.

Example

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

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


Properties

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

Confirmation messages

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

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

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

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

Global UI services on all data spaces and snapshots

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

Common use cases

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

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

Declaration and configuration

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

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

Voir aussi : [Packaging EBX5 modules](#)

Example

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

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

```

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

```

Properties

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

Confirmation Message

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

```

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

```

```
</service>
```

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

Integration of UI services with workflows

Overview

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

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

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

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

Common properties for defining input and output parameters

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

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

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

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

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

Services on a data spaces or a snapshots for data workflows

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

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

Example

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

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

Service on data sets for data workflows

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

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

Example

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

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

Properties

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

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

Service extensions

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

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

Extending a built-in UI service

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

```
</serviceExtension>
```

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

Extending data space and snapshot UI services

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

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

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

Extending data set UI services

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

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

```

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

```

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

CHAPITRE 39

Built-in UI services

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

- when defining workflow model tasks
- as extended UI services when used with service extensions
- when using EBX5 as a web component

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

Voir aussi :

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

Access a data space

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

Service name parameter: `service=@selectDataSpace`

Input parameters

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

Access data (default service)

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

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

Input parameters

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

Compare contents

The compare service is automatically considered complete.

Service name parameter: `service=@compare`

Input parameters

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

Create a new record

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

Service name parameter: `service=@creation`

Input parameters

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

Output parameters

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

Duplicate a record

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

Service name parameter: `service=@duplicate`

Input parameters

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

Output parameters

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

Export data from a table in CSV files

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

Service name parameter: `service=@exportToCSV`

Input parameters

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

Export data from a table in XML files

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

Service name parameter: `service=@exportToXML`

Input parameters

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

Import data into a table from an CSV file

The importFromCSV service is considered complete when import is performed.

Service name parameter: `service=@importFromCSV`

Input parameters

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

Import data into a table from an XML file

The importFromXML service is considered complete when import is performed.

Service name parameter: `service=@importFromXML`

Input parameters

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

Merge a data space

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

Service name parameter: `service=@merge`

Input parameters

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

Output parameters

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

Validate a data space, a snapshot or a data set

The validation service is automatically considered complete.

Service name parameter: `service=@validation`

Input parameters

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

Output parameters

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

CHAPITRE 40

Data services

Overview of data services

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

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

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

Other generic operations for:

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

Enabling and configuring data services

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

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

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

SOAP interactions

Input and output message encoding

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

Tracking information

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

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

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

Exceptions handling

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

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

Accessing data services using JMS

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

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

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

Data services security

Authentication

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

Note

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

- Authentication based on HTTP Request and a call to the method **Directory.authenticateUserFromHttpRequest** [Directory]^{API}. This requires an override of the implementation of this method. For example, it can extract a password-digest or a ticket from an HTTP request.
- Standard HTTP 'Base64 Basic Authentication' encoded using HTTP-Header Authorization, as described in [RFC 2324](#).

If the user agent wishes to send the userid "Aladdin" and password "open sesame", it will use the following header field: Authorization: Basic QWxhZGRpbjpwYVcGVuIHNLc2FtZQ==

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

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

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

Overriding the default security header

It is possible to override the WSS header in order to define another security authentication mechanism. Such an override is taken into account for both HTTP and JMS. To define and override, use the 'SOAP

Header Security declaration' configuration settings under Administration > Lineage, which includes the following fields:

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

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

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

The corresponding XML Schema header declaration would be as follows:

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


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

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

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

Lookup mechanism

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

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

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

Monitoring

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

See [Logging](#) for more information.

Data service operations generated from a data model

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

Common data model operation request parameters

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

Element	Description	Required
branch	The identifier of the data space to which the data set belongs.	One of either this parameter or the 'version' parameter must be defined
version	The identifier of the snapshot to which the data set belongs.	One of either this parameter or the 'branch' parameter must be defined
instance	The unique name of the data set which contains the table to query.	Yes
predicate	XPath predicate defines the records on which the request is applied. If empty, all records will be retrieved, except for the 'delete' operation where this field is mandatory.	Only required for the 'delete' operation
data (used by the insert and update operations)	Contains the records to be modified, represented within the structure of their data model. The whole operation is equivalent to an XML import. The details of the operations performed on data content are specified in the section Import .	Yes
viewPublication	<p>This parameter can be combined with the predicate parameter as a logical AND operation.</p> <p>The behavior of this parameter is described in the section EBX5 as a web component.</p> <p>It cannot be used if the 'viewId' parameter is used, and cannot be used on hierarchical views.</p>	No
viewId	<p><i>Deprecated since version 5.2.3.</i> This parameter has been replaced by the parameter 'viewPublication'. While it remains available for backward compatibility, it will eventually be removed in a future version.</p> <p>This parameter cannot be used if the 'viewPublication' parameter is used.</p>	No
disableRedirectionToLastBroadcast	<p>This property is available for all data service operations.</p> <p>If <code>true</code>, access to a delivery data space on a D3 master node is not redirected to the last broadcast snapshot. Otherwise, access to such a data space is always redirected to the last snapshot broadcast.</p> <p>If this parameter is not present, the default is <code>false</code> (redirection on a D3 master enabled), unless the configuration property ebx.dataservices.disableRedirectionToLastBroadcast.default has been set.</p> <p>If the specified data space is not a delivery data space on a D3 master node, this parameter is ignored.</p>	No

Select operation

Select request

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

with:

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

Select response

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

```

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

```

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

Delete operation

Delete request

```

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

```

with:

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

Delete response

```

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

```

with:

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

Count operation

Count request

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

with:

Element	Description
branch	See the description under Common parameters .
version	See the description under Common parameters .
instance	See the description under Common parameters .
predicate	See the description under Common parameters .

Count response

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

with:

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

Update operation

Update request

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

```

    <d>String</d>
    ...
  </TableName>
</XX>
</data>
</m:update_{TableName}>

```

with:

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

Voir aussi : [Optimistic locking](#)

Update response

```

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

```

with:

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

Insert operation

Insert request

```

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

```

```

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

```

with:

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

Insert response

```

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

```

with:

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

Get changes operations

Get changes requests

Changes between two data sets:

```

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

```



```

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

```

Changes between two tables:

```

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

```

with:

Element	Description	Required
branch	See the description under Common parameters .	
version	See the description under Common parameters .	
instance	See the description under Common parameters .	
compareWithBranch	The identifier of the data space with which to compare. You must use only one parameter between this one and compareWithVersion .	No
compareWithVersion	The identifier of the snapshot with which to compare. You must use only one parameter between this one and compareWithBranch .	No
resolvedMode	Defines whether or not the difference is calculated in resolved mode. Default is 'true'. See Resolved mode <code>[DifferenceHelper]</code> ^{API} for more information.	No

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

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

Voir aussi : *DifferenceHelper* `[DifferenceHelper]`^{API}

Get changes responses

Changes between two data sets:

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

Changes between two tables:

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

Get credentials operation

Get credentials request

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

```

<branch>String</branch>
<version>String</version>
<instance>String</instance>
<predicate>String</predicate>
<viewPublication>String</viewPublication>
</m:getCredentials_{TableName}>

```

with:

Element	Description	Required
branch	See the description under Common parameters .	
version	See the description under Common parameters .	
instance	See the description under Common parameters .	
viewPublication	See the description under Common parameters .	

Get credentials response

```

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

```

With the following possible values:

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

Multiple chained operations

Multiple operations request

It is possible to run multiple operations across tables in the data set, while ensuring a consistent response. The operations are executed sequentially, according to the order defined on the client side. All operations are executed in a single transaction with a 'SERIALIZABLE' isolation level. When an error occurs during one operation in the sequence, all updates are rolled back and the client receives a `StandardException` error message with details.

```

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

```

```

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

```

with:

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

Note:

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

Limitation:

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

Voir aussi :

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

Multiple operations response

See each response operation for details.

```
<ns1:multiResponse xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <response id="id1">
    <ns1:{operation}_{TableName}Response>
      ...
    </ns1:{operation}_{TableName}Response>
  </response>
  <response id="id2">
    <ns1:{operation}_{TableName}Response>
      ...
    </ns1:{operation}_{TableName}Response>
  </response>
</ns1:multiResponse>
```

with:

Element	Description
response	<p>This element contains the response of one operation. It is be repeated multiple times for additional operations. Each response is identified by an 'id' attribute set in the request or automatically generated.</p> <p>The content of the element corresponds to the response of a single operation, such as count, select, getChanges, getCredentials, insert, delete or update.</p>

Optimistic locking

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

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

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

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

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

```

<data>
  <XX>
    <TableName ebxd:lastTime="2010-06-28T10:10:31.046">
      <a>String</a>
      <b>String</b>
      <c>String</c>
      <d>String</d>
      ...
    </TableName>
  </XX>
</data>
</m:update_{TableName}>

```

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

```

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

```

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

Data service operations on data sets and data spaces

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

Element	Description	Required
branch	Identifier of the target data space on which the operation is applied. When not specified, the 'Reference' data space is used except for the merge data space operation where it is required.	One of either this parameter or the 'version' parameter must be defined. Required for the data space merge and replication refresh operations.
version	Identifier of the target snapshot on which the operation is applied.	One of either this parameter or the 'branch' parameter must be defined
versionName	Identifier of the snapshot to create. If empty, it will be defined on the server side.	No
childBranchName	Identifier of the data space child to create. If empty, it will be defined on the server side.	No
instance	The unique name of the data set on which the operation is applied.	Required for the replication refresh operation.
ensureActivation	Defines if validation must also check whether this instance is activated.	Yes
details	<p>Defines if validation returns details.</p> <p>The optional attribute <code>severityThreshold</code> defines the lowest severity level of message to return. The possible values are, in descending order of severity, 'fatal', 'error', 'warning', or 'info'. For example, setting the value to 'error' will return error and fatal validation messages. If this attribute is not defined, all levels of messages are returned by default.</p> <p>The optional attribute <code>locale</code> (default 'en-US') defines the language in which the validation messages are to be returned.</p>	No. If not specified, no details are returned.
owner	<p>Defines the owner.</p> <p>Must respect the inner format as returned by Profile.format() [Profile]^{API}.</p>	No
branchToCopyPermissionFrom	Defines the identifier of the data space from which to copy the permissions.	No
documentation	Documentation for the data space or the snapshot to create. Multiple documentation elements may be used.	No
locale (nested under the documentation element)	Locale of the data space or snapshot documentation.	Only required when the <code>documentation</code> element is used

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

Validate a data space

Validate data space request

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

Validate data space response

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

Validate a data set

Validate data set request

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

Validate data set response

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



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

Create a data space

Create data space request

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

Create data space response

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

with:

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

Create a snapshot

Create snapshot request

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

```
<description>String</description>
</documentation>
</m:createVersion>
```

Create snapshot response

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

with:

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

Merge a data space

Merge data space request

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

with:

Element	Description	Required
deleteDataOnMerge	<p>This parameter is available for the merge data space operation. Sets whether the specified data space and its associated snapshots will be deleted upon merge.</p> <p>When this parameter is not specified in the request, the default value is 'false'. It is possible to redefine the default value by specifying the property <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> in ebx.properties.</p> <p>See Deleting data and history for more information.</p>	No
deleteHistoryOnMerge	<p>This parameter is available for the merge data space operation. Sets whether the history associated with the specified data space will be deleted upon merge. Default value is 'false'.</p> <p>When this parameter is not specified in the request, the default value is 'false'. It is possible to redefine the default value by specifying the property <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> in ebx.properties.</p> <p>See Deleting data and history for more information.</p>	No

Note

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

Merge data space response

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

with:

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

Close a data space or snapshot**Close data space or snapshot request****Close data space request:**

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

Close snapshot request:

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

with:

Element	Description	Required
deleteDataOnClose	<p>This parameter is available for the close data space and close snapshot operations. Sets whether the specified snapshot, or data space and its associated snapshots, will be deleted upon closure.</p> <p>When this parameter is not specified in the request, the default value is 'false'. It is possible to redefine this default value by specifying the property <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> in ebx.properties.</p> <p>See Deleting data and history for more information.</p>	No
deleteHistoryOnClose	<p>This parameter is available for the close data space operation. Sets whether the history associated with the specified data space will be deleted upon closure. Default value is 'false'.</p> <p>When this parameter is not specified in the request, the default value is 'false'. It is possible to redefine the default value by specifying the property <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> in ebx.properties.</p> <p>See Deleting data and history for more information.</p>	No

Close data space or snapshot response

Close data space response:

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

Close snapshot request:

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

Replication refresh

Replication refresh request

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

with:

Element	Description	Required
branch	See the description under Common parameters .	Yes
instance	See the description under Common parameters .	Yes
unitName	Name of the replication unit. Voir aussi : Replication refresh information	Yes

Replication refresh response

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

with:

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

Data service operations on data workflows

Parameters for operations on data workflows are as follows:

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

Start a workflow

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

```

</parameter>
</parameters>
</m:workflowProcessInstanceStart>

```

End a workflow

```

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

```

Known limitations

Naming convention

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

Date, time & dateTime format

Data services only support the following date and time formats:

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

CHAPITRE 41

XML import and export

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

Imports and exports can also be done programmatically.

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

Imports

Attention

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

Import mode

When importing an XML file, you must specify one of the following import modes, which will dictate how the import procedure handles the source records.

Insert mode	Only record creations are allowed. If a record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled.
Update mode	Only modifications of existing records are allowed. If no record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled.
Update or insert mode	If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created.
Replace (synchronization) mode	If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created. If a record exists in the target table but is not present in the source XML file, that record is deleted from the table.

Insert and update operations

The mode '*by delta*' allows ignoring data model elements that are missing from the source XML document. This mode can be enabled through data services or the Java API. The following table summarizes the behavior of insert and update operations when elements are not present in the source document.

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

State in source XML document	Behavior
Element does not exist in the source document	<p>If 'by delta' mode is disabled (default):</p> <p>Target field value is set to one of the following:</p> <ul style="list-style-type: none"> • If the element defines a default value, the target field value is set to that default value. • If the element is of a type other than a string or list, the target field value is set to <code>null</code>. • If the element is an aggregated list, the target field value is set to an empty list. • If the element is a string that distinguishes <code>null</code> from an empty string, the target field value is set to <code>null</code>. If it is a string that does not distinguish between the two, an empty string. <p>Note: The user performing the import must have the permissions necessary to create or change the target field value. Otherwise, the value will remain unchanged.</p> <p>If 'by delta' mode has been enabled through data services or the Java API:</p> <ul style="list-style-type: none"> • For the <code>update</code> operation, the field value remains unchanged. • For the <code>insert</code> operation, the behavior is the same as when <code>byDelta</code> mode is disabled.
Element exists but is empty (for example, <code><fieldA/></code>)	<ul style="list-style-type: none"> • For nodes of type <code>xs:string</code> (or one of its sub-types), the target field's value is set to <code>null</code> if it distinguishes <code>null</code> from an empty string. Otherwise, the value is set to empty string. • For non-<code>xs:string</code> type nodes, an exception is thrown in conformance with XML Schema. <p>See also EBX5 whitespace management for data types.</p>
Element is present and <code>null</code> (for example, <code><fieldA xsi:nil="true"/></code>)	<p>The target field is always set to <code>null</code> except for lists, for which it is not supported.</p> <p>In order to use the <code>xsi:nil="true"</code> attribute, you must import the namespace <code>xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code>.</p>

Optimistic locking

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

Exports

Note

Exported XML documents are always encoded in UTF-8.

When exporting to XML, if the table has filters applied, only the records that correspond to the filter are included in the exported file.

The XML export options are as follows:

Download file name	Specifies the name of the XML file to be exported. This field is pre-populated with the name of the table from which the records are being exported.
User-friendly mode	<p>Specifies whether exported values will be represented in a user-friendly way, or in the standard XML raw format. For example, in user-friendly mode, dates and numbers are formatted according to the user's locale, and foreign keys and enumerated values display their associated labels.</p> <p>Note: If this option is selected, the exported file will not be able to be re-imported.</p>
Include technical data	<p>Specifies whether internal technical data will be included in the export.</p> <p>Note: If this option is selected, the exported file will not be able to be re-imported.</p>
Include computed values	Specifies whether computed values will be exported.
Is indented	Specifies whether the file should be indented to improve readability.

Handling of field values

Date, time & dateTime format

The following date and time formats are supported:

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

CHAPITRE 42

CSV import and export

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

Imports and exports can also be done programmatically.

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

Imports

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

Insert mode	Only record creations are allowed. If a record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled.
Update mode	Only modifications of existing records are allowed. If no record exists in the target table with the same primary key as the source record, an error is returned and the whole import operation is cancelled.
Update or insert mode	If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created.
Replace (synchronization) mode	If a record with the same primary key as the source record already exists in the target table, that record is updated. Otherwise, a new record is created. If a record exists in the target table but is not present in the source XML file, that record is deleted from the table.

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

Exports

When exporting to CSV, if the table has filters applied, only the records that correspond to the filter are included in the exported file.

The CSV export options are as follows:

Download file name	Specifies the name of the CSV file to be exported. This field is pre-populated with the name of the table from which the records are being exported.
File encoding	Specifies the character encoding to use for the exported file. The default is UTF-8.
User-friendly mode	<p>Specifies whether exported values will be represented in a user-friendly way, or in a raw format. For example, in user-friendly mode, dates and numbers are formatted according to the user's locale, and foreign keys and enumerated values display their associated labels.</p> <p>Note: If this option is selected, the exported file will not be able to be re-imported.</p>
Include technical data	<p>Specifies whether internal technical data will be included in the export.</p> <p>Note: If this option is selected, the exported file will not be able to be re-imported.</p>
Include computed values	Specifies whether computed values will be exported.
Column header	<p>Specifies the whether or not to include column headers in the CSV file.</p> <ul style="list-style-type: none"> • No header • Label: For each column in the spreadsheet, the CSV displays its label. Each label is localized according to the locale preference of the current session. If no user-friendly label is defined for a node, the technical name of the node is used. • XPath: For each column in the spreadsheet, the CSV displays the path to the node in the table.
Separator	Specifies the field separator to use in the exported file. The default is to use commas.

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

Handling of field values

Aggregated lists

The CSV import and export services support multi-valued fields, namely aggregated lists. This is only supported for simple typed lists, such as lists of `string`, `date`, or `int`, and for table references. If a table reference is linked to a composite primary key, each item in the list is a formatted string, for example, "true|99". Aggregated lists of groups are not exported.

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

Hidden fields

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

'Null' value for strings

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

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

Date, time & dateTime format

The following date and time formats are supported:

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

Known limitations

Aggregated lists of groups

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

Terminal groups

In a CSV file, it is impossible to differentiate a created terminal group that contains only empty fields from a non-created one.

As a consequence, some differences may appear during comparison after performing an export followed by an import. To ensure the symmetry of import and export, use XML import and export instead. See [XML import and export](#).

Column label headers

If two columns share the same label header, an export of the table can be performed successfully, but exported data cannot later be re-imported.

CHAPITRE 43

Supported XPath syntax

Overview

The XPath notation used in EBX5 must conform to the *abbreviated syntax* of the [XML Path Language \(XPath\) Version 1.0](#) standard, with certain restrictions. This document details the abbreviated syntax that is supported.

Example expressions

The general XPath expression is:

```
path[predicate]
```

Absolute path

```
/library/books/
```

Relative paths

```
./Author
```

```
../Title
```

Root and descendant paths

```
//books
```

Table paths with predicates

```
../../books/[author_id = 0101 and (publisher = 'harmattan')]
```

```
/library/books/[not(publisher = 'dumesnil')]
```

Complex predicates

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

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

Syntax specifications for XPath expressions

Overview

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

Predicate specification

Expression	Format	Notes/Example
<predicate>	Example: A and (B or not(C)) A,B,C: <atomic expression>	Composition of: logical operators braces, not() and atomic expressions.
<atomic expression>	<path><comparator><criterion> or method(<path>,<criterion>)	royalty = 24.5 starts-with(title, 'Johnat')booleanValue = true
<path>	<relative path>	Relative to the table that contains it: ../authorstitle
<comparator>	<boolean comparator>, <numeric comparator> or <string comparator>	
<boolean comparator>	= or !=	
<numeric comparator>	= , != , < , > , <= , or >=	
<string comparator>	=	
<method>	<date method>, <string method>, osd:is-null method or osd:is-not-null method	
<date, time & dateTime method>	date-less-than, date-equal or date-greater-than	
<string method>	matches, starts-with, ends-with, contains, osd:is-empty, osd:is-not-empty, osd:is-equal-case-insensitive, osd:starts-with-case-insensitive, osd:ends-with-case-insensitive, or osd:contains-case-insensitive	
<criterion>	<boolean criterion>, <numeric criterion>, <string criterion>, <date criterion>, <time criterion>, or <dateTime criterion>	
<boolean criterion>	true , false	
<numeric criterion>	An integer or a decimal	-4.6
<string criterion>	Quoted character string	'azerty'
<date criterion>	Quoted and formatted as 'yyyy-MM-dd'	'2007-12-31'
<time criterion>	Quoted and formatted as 'HH:mm:ss' or 'HH:mm:ss.SSS'	'11:55:00'
<dateTime criterion>	Quoted and formatted as 'yyyy-MM-ddTHH:mm:ss' or 'yyyy-MM-ddTHH:mm:ss.SSS'	'2007-12-31T11:55:00'

XPath 1.0 formula

It is possible to use an XPath 1.0 formula in the criterion value part of an atomic predicate expression (right-hand side).

For example, instead of `[./a=3]`, you may use the expression `[./a=(floor(./d)+ 2.0)]`.

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

Contextual values

For predicates that are relative to a selected node, the criterion value right-hand side) can be replaced with a contextual path using the syntax `${<relative-path>}` where *<relative-path>* is the location of the element relative to the selected node.

Note

When calling a method, the criterion is the second parameter, and the first parameter cannot be a relative value.

Aggregated lists

For predicates on aggregated lists, the predicate returns `true` regardless of the comparator if one of the list elements verifies the predicate.

Note

Special attention must be paid to the comparator `!=`. For example, for an aggregated list, `./list != 'a'` is not the same as `not(./list = 'a')`. Where the list contains the elements `(e1,e2,...)`, the first predicate is equivalent to `e1 != 'a' or e2 != 'a' ...`, while the second is equivalent to `e1 != 'a' and e2 != 'a'`

'Null' values

Null values must be explicitly treated in a predicate using the operators `osd:is-null` and `osd:is-not-null`.

For example, `/root/products[./price<100]` or `/root/products[./price!=100]` will not return any products whose prices are not set (null). For the latter case to return unset values as well, the predicate must instead be: `/root/products[./price!=100 or osd:is-null(./price)]`.

Extraction of foreign keys

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

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

Example

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

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

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

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

Divers

CHAPITRE 44

Permissions

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

Main principles

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

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

Users, roles and profiles

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

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

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

Particular definitions:

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

Permission rules

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

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

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

Resolution of permissions

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

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

Note. From Java environment, the class **SessionPermissions** [SessionPermissions]^{API} provides access to the resolved permissions.

Particular privileges on data set

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

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

Notes.

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

Particular privileges on data space

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

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

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

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

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

Merge and permissions

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

Definition of permissions with EBX5

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

Permissions on data space

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

1. Data space access: defines access rights (read-write, read-only or hidden, see below for definition).
2. Restriction: indicates if "profile (role or user) - permission (right or action)" associations, for the current data space, should have priority.
3. Create a child data space: indicates if it is possible to create a child data space from the current data space.
4. Create a child snapshot: indicates if it is possible to create a snapshot of the current data space.
5. Initiate merge: indicates if a profile has the right to merge a data space with its parent data space.
6. Export archive: indicates if a profile has the right to export the current data space as an archive.
7. Import archive: indicates if a profile has the right to import an archive into the current data space.
8. Close a data space: indicates if a profile has the right to close the current data space.
9. Close a snapshot: indicates if a profile has the right to close a snapshot of the current data space.
10. Rights on services: indicates if a profile has the right to execute some home services. By default, all data space services are enabled. The administrator or the "super owner" of the target data space or a given user who is allowed to change permissions on the target data space can then modify these data space services permissions for a given profile.
11. Permissions of child data space when created: defined the same permissions as above but on current data space children.

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

Mode	Authorization
	<ul style="list-style-type: none"> The user cannot perform any action on the data space.

Permissions on a new data space

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

Note

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

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

Permissions on data set

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

Actions on data set

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

Default actions on tables

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

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

Permissions on tables

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

Default access right on nodes values

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

Permissions on terminal nodes

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

Permissions on services

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

Definition of permissions with programmatic rules

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

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

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

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

Resolution of permissions

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

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

Restriction policy notion

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

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

Example

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

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

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

Solving access rights

Solving access rights defined with EBX5

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

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

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

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

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

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

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

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

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

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

Data space/snapshot access right resolution

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

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

Data set access right resolution

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

Node access right resolution

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

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

Specific case of table and table child node

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

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

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

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

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

Solving access rights defined with programmatic rules

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

Rule resolution on data set

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

Rule resolution on record

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

Access rule resolution on node

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

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

Display policy in tableRef with access rules on rows

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

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

Solving actions and services

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

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

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

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

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

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

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

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

Users	Actions list applied
	Duplicate a record

CHAPITRE 45

Labeling and localization

Overview

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

Localization of the user interaction

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

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

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

Textual information

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

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

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

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

Values formats

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

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

Formatting policies

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

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

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

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

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

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

The elements `date`, `dateTime` and `time` are mandatory.

Syntax for locales

There are two ways to write a locale:

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

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

Extending EBX5 internationalization

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

CHAPITRE 46

Extending EBX5 internationalization

Overview

EBX5 offers the following localization capabilities:

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

EBX5 native localization

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

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

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

Extending EBX5 user interface localization

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

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

Adding a new locale

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

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

Specific localization deployment

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

Known limitations

Default localization

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

Non extendable materials

Localization of the following cannot be extended:

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

CHAPITRE 47

Inheritance and value resolution

Overview

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

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

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

Data set inheritance

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

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

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

Inherited fields

As opposed to data set inheritance (which exploits a global built-in relationship between data sets), inherited fields are able to exploit finer-grained dependencies that are specific to the data structure. It allows you to factorize and specialize data at the business entities-level.

For example, if the model specifies that a Product is associated with a FamilyOfProducts, it is possible that some attributes of Product inherit their value from attributes defined in its associated family.

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

Computed values (functions)

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

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

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

Data set inheritance

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

Data set inheritance declaration

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

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

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

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

If not specified, inheritance mechanism is disabled.

Lookup mechanism for values

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

1. if the value is locally defined, returns it

It can explicitly be `null`

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

`null` is returned if no default value is defined

Note: default values cannot be defined on:

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

Lookup mechanism for records

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

- A *root record* is locally defined in the table and has no parent. This means that no record with the same primary key exists in the parent table, or that this parent is an occulting record.
- An *overwriting record* is locally defined in the table and has a parent record. This means that a record with the same primary key exists in the parent table, and that this parent is not an occulting record. The overwriting record inherits its values from its parent, except for the values that it explicitly redefines.
- An *inherited record* has a parent record but is not locally defined in the current table. All values are inherited.

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

- An *occulting record* specifies that if a parent with same primary key is defined, this parent will not be seen in table descendants.

Inherited fields

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

Field inheritance declaration

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

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

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

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

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

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

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

Lookup mechanism for value

The lookup mechanism for inherited fields values is the following:

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

It can explicitly be `null`

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

Optimize & Refactor

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

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

Procedure details

Data sets are processed from the bottom up, which means that if the service is run on the data set at level N , with $N+1$ being the level of its children and $N+2$ being the level of its children's children, the service will first process the data sets at level $N+2$ to determine if they can be optimized with respect to the data sets at level $N+1$. Next, it would proceed with an optimization of level $N+1$ against level N .

Note

- These optimization and refactoring functions do not handle default values that are declared in the data model.
- The highest level considered during the optimization procedure is always the data set on which the service is run. This means that optimization and refactoring is not performed between the target data set and its own parent.
- Table optimization is done for records with the same primary key.
- Inherited fields are not optimized.
- *Optimization and refactoring functions do not modify the resolved view of a data set if it is activated.*

Service availability

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

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

Note

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

CHAPITRE 48

Criteria Editor

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

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

Voir aussi : [Supported XPath syntax](#)

Conditional Blocks

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

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

Atomic Criteria

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

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

Expression

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

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

CHAPITRE 49

Performance guidelines

Basic performance check-list

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

Expensive programmatic extensions

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

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

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

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

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

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

Insufficient memory

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

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

Relational mode

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

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

Directory integration

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

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

Directory.isUserInRole [Directory]^{API} .

Aggregated lists

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

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

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

Reorganization of database tables

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

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

Memory Management

Loading strategy

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

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

The following sections give details about the existing modes.

On-demand loading and unloading

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

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

Forced loading

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

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

Forced loading and prevalidation

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

Monitoring

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

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

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

Tuning the memory

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

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

Validation

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

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

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

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

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

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

Massive updates

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

Transaction boundaries

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

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

For reducing transactions' size, it is possible to:

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

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

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

Triggers

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

Access to tables

Functionalities

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

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

Semantic mode: Architecture and design

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

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

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

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

Semantic mode: Performance considerations

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

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

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

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

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

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

Semantic mode: Other operations on tables

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

Semantic mode: Conclusion about tables

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

Relational mode: Performance considerations

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

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

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

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

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

Persistence

CHAPITRE 50

Overview of persistence

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

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

Note

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

Persistence of managed master data

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

Semantic mode

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

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

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

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

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

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

Voir aussi :

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

Relational mode

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

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

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

Voir aussi : [*Relational mode*](#)

Comparison of semantic mode and relational mode

This table summarizes the differences between the two persistence modes:

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

Historization of tables

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

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

Voir aussi : [History](#)

Replication and SQL access to master data

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

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

Voir aussi : [Replication](#)

Mapped mode

Overview of mapped mode

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

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

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

Voir aussi :

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

Data model restrictions due to mapped mode

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

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

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

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

Voir aussi : [Data model evolutions](#)

CHAPITRE 51

Relational mode

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

Enabling relational mode for a data model

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

Example of a relational mode declaration:

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

with the elements:

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

Validation

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

Structural constraints

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

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

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

Foreign key blocking mode

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

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

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

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

Constraints on the whole table

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

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

Voir aussi : ***ConstraintOnTable interface*** `[ConstraintOnTable]API`

SQL access to data in relational mode

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

Voir aussi : [SQL access to history](#)

Finding the table in the database

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

SQL reads

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

SQL writes

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

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

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

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

Limitations of the relational mode

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

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

Data model restrictions for tables in relational mode

Some restrictions apply to data models in relational mode:

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

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

Voir aussi : [Data model evolutions](#)

Other limitations

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

CHAPITRE 52

History

Overview

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

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

Voir aussi :

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

Configuring history

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

Configuring history in the repository

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

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

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

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

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

Configuring history in the data model

Activating table history

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

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

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

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

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

Disabling history on a specific field or group

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

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

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

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

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

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

Note

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

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

Integrity

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

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

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

History views and permissions

Table history view

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

The next section explains how permissions are resolved.

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

Permissions for table history

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

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

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

Transaction history views

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

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

For more information, see [vue historique des transactions](#).

SQL access to history

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

Voir aussi : [SQL access to data in relational mode](#)

Access restrictions

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

Relational schema overview

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

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

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

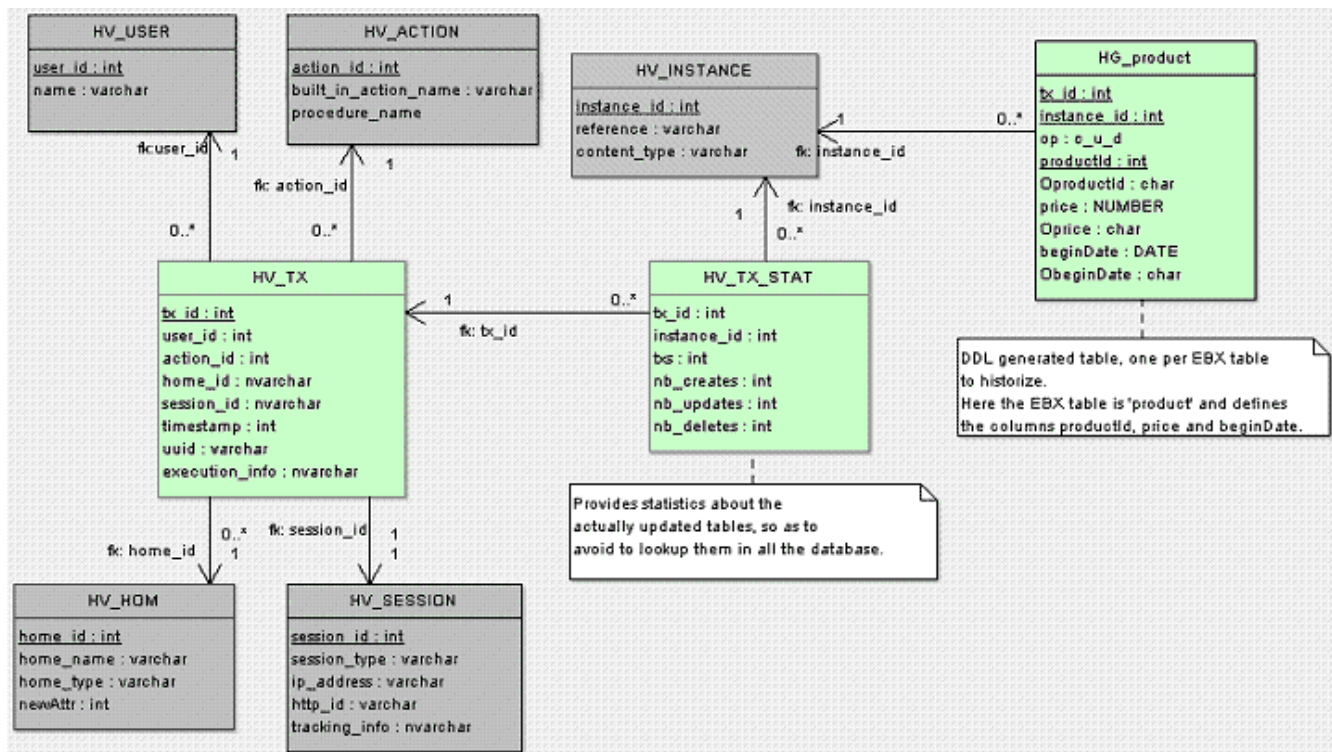
Example of a generated history table

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

Product

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

The diagram below shows the resulting relational schema:



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

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

Operation field values

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

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

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

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

Impacts and limitations of the historized mode

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

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

Validation

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

Data model restrictions for historized tables

Some restrictions apply to data models containing historized tables:

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

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

Voir aussi : [Data model evolutions](#)

Other limitations

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

CHAPITRE 53

Replication

Overview

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

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

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

Voir aussi :

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

Configuring replication

Enabling replication

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

The nested elements are as follows:

Element	Description	Required
name	Nom de l'unité de réplication. Ce nom identifie l'unité de réplication dans le modèle de données. Ce nom doit être unique dans le modèle de données.	Yes
dataSpace	Indique l'espace de données concerné par la réplication. Cet espace de données ne peut ni être une version ni être relationnel.	Yes
dataSet	Indique le jeu de données concerné par la réplication.	Yes
refresh	Specifies the data synchronization policy. The possible policies are: <ul style="list-style-type: none"> onCommit: The replica table content in the database is always up to date with respect to its source table. Every transaction that updates the EBX5 source table triggers the corresponding insert, update, and delete statements on the replica table. onDemand: The replication of specified tables is only done when an explicit refresh operation is performed. See Requesting an 'onDemand' replication refresh. 	Yes
table/path	Indique le chemin de la table dans le modèle de données qui doit être répliquée dans la base de données.	Yes
table/nameInDatabase	Indique le nom de la table dans la base de données qui contiendra les données répliquées. Ce nom doit être unique par rapport à toutes les unités de réplication.	Yes

For example:

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

Notes:

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

Disabling replication on a specific field or group

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

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

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

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

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

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

Note

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

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

Accessing a replica table using SQL

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

Voir aussi : [SQL access to history](#)

Finding the replica table in the database

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

Access restrictions

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

Voir aussi : [Rules for the access to the database and user privileges](#)

SQL reads

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

Requesting an 'onDemand' replication refresh

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

There are several ways to request a replication refresh:

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

Impact and limitations of replication

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

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

Validation

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

Data model restrictions for replicated tables

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

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

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

Voir aussi : [Data model evolutions](#)

Database configuration

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

For instance, with Oracle:

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

Other limitations

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

CHAPITRE 54

Data model evolutions

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

Note

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

Voir aussi : [Mapped mode](#)

Types of permitted evolutions

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

Model-level evolutions

The following modifications can be made to existing data models:

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

Table-level evolutions

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

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

Field-level evolutions

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

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

Index-level evolutions

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

Limitations/restrictions

Note

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

Limitations related to primary key evolutions

When a primary key definition is modified:

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

Limitations related to foreign key evolutions

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

Limitations related to field-level evolutions

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

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

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

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

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

Guide d'administration (en anglais)

Installation & configuration

CHAPITRE 55

Supported environments

Browsing environment

Supported web browsers

EBX5 web tool supports the following browsers:

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

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

Note

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

Screen resolution

The minimum screen resolution for EBX5 is 1024x768.

Refreshing pages

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

Browser configuration

The following features must be activated in your browser configuration:

- JavaScript,
- Ajax,
- pop-ups.

Attention

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

Supported application servers

EBX5 supports the following configurations:

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

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

Supported databases

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

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

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

Attention

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

Voir aussi :

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

CHAPITRE 56

Java EE deployment

Software components

EBX5 uses the following components:

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

Voir aussi : [Supported environments](#)

Third-party libraries

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

Database drivers

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

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

Voir aussi :

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

SMTP and emails

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

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

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

Voir aussi :

- [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/>

Voir aussi : [EBX5 main configuration file](#)

Java Message Service (JMS)

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

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

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

Voir aussi : [EBX5 main configuration file](#)

Web applications

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

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

EBX5 built-in web applications

EBX5 includes the following built-in web applications.

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

Custom web applications

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

Voir aussi :

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

Deployment details

Introduction

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

Attention

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

Voir aussi :

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

User interface and web access

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

Configuring the deployment descriptor for 'FrontServlet'

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

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

Configuring the application server for 'FrontServlet'

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

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

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

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

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

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

Data source of the EBX5 repository

Note

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

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

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

Voir aussi :

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

Mail sessions

Note

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

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

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

JMS connection factory

Note

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

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

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

Note

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

JMS for data services

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

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

Note

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

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

JMS for distributed data delivery (D3)

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

Note

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

Common declarations for master and slave nodes

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

Additional declarations on the master node

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

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

Note

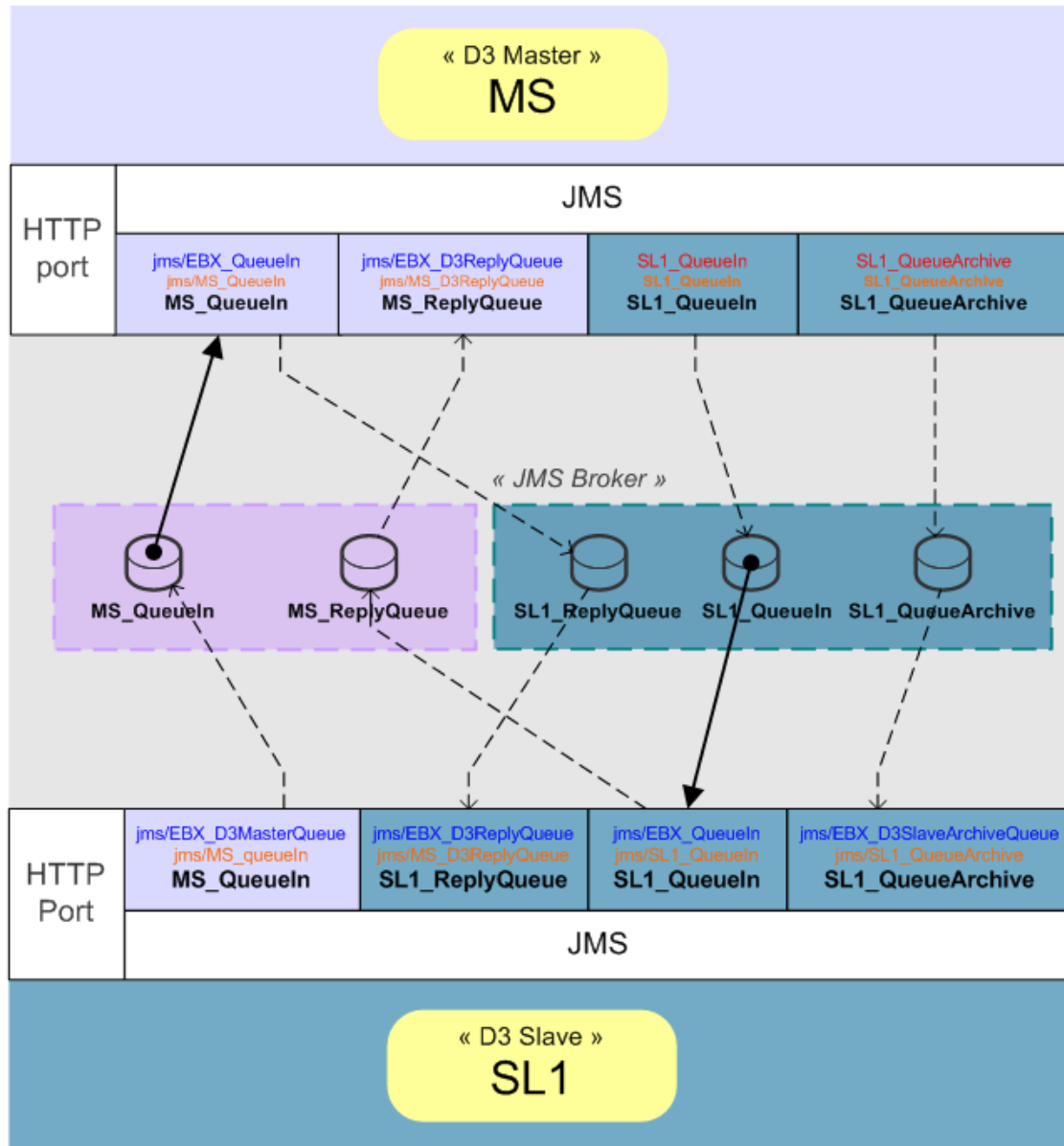
Due to restrictions on the master node, the resource name of the slave communication queue must be the same as the physical name of the queue.

Note

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

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

Message Queueing (D3 / JMS)



Java EE deployment examples

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

- [\[voir documentation HTML\]](#) Installation on Tomcat 5
- [\[voir documentation HTML\]](#) Installation on WebSphere 6
- [\[voir documentation HTML\]](#) Installation on WebLogic 8.1

Attention

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

J2EE.8.2 Optional Package Support

(...)

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

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

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

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

Limitations

Clustering

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

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

CHAPITRE 57

EBX5 main configuration file

Overview of the EBX5 main configuration file

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

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

Voir aussi :

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

Location of the file

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

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

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

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

Note

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

Custom properties and variable substitution

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

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

Setting EBX5 license key

Voir aussi : [Installing a repository using the Configuration Assistant](#)

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

Setting the EBX5 root directory

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

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

Configuring the EBX5 repository

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

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

Voir aussi :

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

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

```

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

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

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

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

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

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

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

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

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

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

```

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

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

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

Configuring the user and roles directory

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

Voir aussi :

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

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

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

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

Setting temporary files directories

Temporary files are stored as follows:

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

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

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

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

Activating the XML audit trail

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

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

Configuring the EBX5 logs

The most important logging categories are:

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

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

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

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

Specific appender 'ebxFile'

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

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

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

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

Specific SMTP appender

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

Voir aussi : [Activating and configuring SMTP and e-mails](#)

Activating and configuring SMTP and e-mails

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

Voir aussi : [Mail sessions](#)

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

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

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

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

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

Configuring data services

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

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

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

Activating and configuring JMS

Voir aussi : [JMS for data services](#)

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

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

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

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

Configuring distributed data delivery (D3)

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

Voir aussi :

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

Configuring web access from end user browsers

URLs computing

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

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

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

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

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

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

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

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

Proxy mode

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

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

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

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

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

The main configuration file is configured as follows:

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

#####

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

#####

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

Reverse proxy mode

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

The main configuration file is configured as follows:

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

URL parameters encoding

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

The main configuration file is configured as follows:

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

Tuning the EBX5 repository

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

The main configuration file is configured as follows:

```
#####
## Technical parameters for memory and performance tuning
#####
# Import commit threshold allows to specify the commit threshold
# exclusively for the archive import launched directly from Manager.
#
# For more details about the commit threshold,
# see the JavaDoc ProcedureContext.setCommitThreshold().
# Default value is 0.
#
ebx.manager.import.commit.threshold=100
# Validation messages threshold allows to specify the maximum number of
```

```
# messages to consider when performing a validation.
# This threshold is considered for each severity in each validation report.
# When the threshold is reached:
# - for severities 'error' or 'fatal', the validation is stopped.
# - for severities 'info' or 'warning', the validation continues without
# registering messages beyond the threshold. However the number of messages
# is still counted and messages of other severities can still be added.
#
# When set to 0 or a negative value the threshold is not considered.
# Default value is 0.
#
ebx.validation.messages.threshold=100
```

Miscellaneous

Activating data workflows

This parameter specifies whether the data workflow is activated. This parameter is not taken into account on the fly. The server must be restarted whenever the value changes.

```
#####
## Workflow activation.
## Default is false.
#####
ebx.workflow.activation = true
```

Deployment site identification

This parameter allows you to specify the address for technical log emails.

```
#####
## Unique Site Name
## --> used by monitoring mails and by repository
#####
ebx.site.name= name@domain.com
```

Dynamically reloading the main configuration

Some parameters can be dynamically reloaded, without restarting EBX5. The parameter 'thisfile.checks.intervalInSeconds' indicates the time interval between each check of the main configuration file.

```
#####
### Checks if this file has been updated
### If value <= 0, no more checks will be done
#####
thisfile.checks.intervalInSeconds=1
```

In development mode, this parameter can be set as low as one second. On production systems, where changes are expected to be less frequent, the value can be greater, or set to '0' to disable hot reloading entirely.

This property is not always supported when the module is deployed as a WAR, as it depends on the application server.

Applications that use the EBX5 navigation engine

Application template reloading:

```
#####
## Reload templates when it is updated
## (default value for all EBX5 modules).
## (value can be overridden by each EBX5 module.
#####
templates.checksIfUpdated=true
```

Debug mode in the EBX5 modules web pages (use only when developing):

```
#####
## End-User Debug Mode
## (default for all EBX5 modules ).
## Debug information appears on end-user web page.
#####
frontEnd.debugMode=false
```

Module registration

When the application server is started, all web applications declared as EBX5 modules have to register. Concurrently, depending on the loading strategy of data spaces, the deployment of the web application `ebx` may require the compilation of data models and their modules. Since these modules may not yet be registered (it depends on the order of web application deployment at server startup), a wait loop is implemented. This gives the module time to be registered.

Voir aussi : [Packaging EBX5 modules](#)

```
#####
## When the application server is started, all web applications declared as
## EBX5 modules have to register. This property
## specifies the estimated time in seconds taken by the application server
## to deploy all its web applications at startup. Beyond this time,
## if a required module has not yet registered, it is considered to be absent
## and an error is reported to 'kernel' log.
## Default is 30 seconds.
#####
#ebx.module.timeInSecondsForModuleRegistration=30
```

Running mode

This property defines how EBX5 runs. Three modes are available: *development*, *integration* and *production*.

When running in *development* mode, the [development tools](#) are activated in EBX5 and more technical information is displayed.

Note

The administrator always has access to this information regardless of the mode used.

The technical information is as follows:

Documentation pane	In the case of a computed value, the Java class name is displayed. The button to get the path to a node is also displayed.
Data model assistant	Data model configuration and additional options, such as Services, Business Objects and Rules, Java Bindings, and some advanced properties.
Validation report	The <i>Validation</i> tab is displayed.
Log	The logs include additional technical information intended for the developer. For example, a warning is written to logs if a drop-down list is defined on a node which is not an enumeration in a UI Bean.

```
#####
## Server Mode
## Value must be one of: development, integration, production
## Default is production.
#####
backend.mode=integration
```

Note

There is no difference between the modes *integration* and *production*.

Resource filtering

This property allows the filtering of certain files and directories in the resource directory contents (resource type node, with an associated facet that indicates the directory that contains usable resources).

```
#####
## list (separated by comma) of regexps excluding resource
## the regexp must be of type "m:[pattern]:[options]".
## the list can be void
#####
ebx.resource.exclude=m:CVS/*:
```

CHAPITRE 58

Installing a repository using the Configuration Assistant

The EBX5 Configuration Assistant guides you through the initial configuration of the EBX5 repository. If EBX5 does not have a repository installed upon start up, the configuration assistant launches automatically.

Before starting the configuration of the repository, ensure that EBX5 is correctly deployed on the application server. For further information about EBX5 deployment, see [Java EE deployment overview](#).

Note

The main configuration file `ebx.properties` must also be correctly configured. See [EBX5 configuration properties](#) for more information.

License key

When you launch EBX5, the license key page displays automatically if no valid license key is available, that is, if there is no license key entered in the main configuration file `ebx.properties`, or if the current license key has expired.

If you do not have a license key, you may obtain a trial license key at <http://www.orchestranetworks.com/support>.

Configuration steps

The EBX5 configuration assistant guides you through the following steps:

1. Validating the license agreement.
2. Configuring the repository
3. Defining users in the default user and roles directory (if a custom directory is not defined)
4. Validating the information entered
5. Installing the EBX5 repository

Validating the license agreement

The page of the configuration assistant displays the product license agreement. In order to proceed with the configuration, you must read and accept the license agreement.

Configuring the repository

This page displays some of the properties defined in the EBX5 main configuration file `ebx.properties`. You also define several basic properties of the repository in this step.

Id of the repository (<code>repositoryId</code>)	Must uniquely identify the repository (in the scope of the enterprise). The identifier is 48 bits (6 bytes) long and is usually represented as 12 hexadecimal digits. This information is used for generating the Universally Unique Identifiers (UUIDs) of entities created in the repository, and also of transactions logged in the history. This identifier acts as the "UUID node", as specified by RFC 4122.
Repository label	Defines a user-friendly label that indicates the purpose and context of the repository.

Defining users in the default directory

If a custom user and roles directory is not defined in the main configuration file `ebx.properties`, the EBX5 configuration assistant allows you to define default users for the default user and roles directory. For more information about the default user and roles directory, see the [User and roles directory](#).

An administrator user must be defined. You may optionally create a second user.

Validating the information entered

Before proceeding with the installation of the repository, you can review the configuration of the repository and the information you have entered on the 'Configuration Summary' page. If you need to modify information, you can return to the previous pages using the configuration assistant's **< Back** buttons.

Once you have verified the configuration, click the button **Install the repository** > to proceed with the installation.

Installing the EBX5 repository

The repository installation is performed using the information that you provided. When the installation is complete, you are redirected to the repository login page.

CHAPITRE 59

Deploying and registering EBX5 add-ons

Note

Refer to the documentation of each add-on for additional installation and configuration information in conjunction with that which is found here.

Deploying an add-on module

Note

Each EBX5 add-on build is intended to run with a specific version of EBX5. Ensure that you have the correct build of the add-on for the version of EBX5 on which it will run.

The web application deployment descriptor for the add-on module must specify that class definitions and resources from the web application are to be loaded in preference to classes from the parent and server classloaders.

For example, on WebSphere Application Server, this can be done by setting `<context-priority-classloader>true</context-priority-classloader>` in the `web-app` element of the deployment descriptor.

On WebLogic, include `<prefer-web-inf-classes>True</prefer-web-inf-classes>` in `weblogic.xml`.

See the documentation on class loading of your application server for more information.

Registering an add-on module

To register a new EBX5 add-on in the repository:

1. Navigate to the 'Administration' area.
2. Click the down-arrow in the navigation pane and select the 'Add-ons' entry.
3. On the **Add-ons > Registered Add-ons** page, click the **+** button to create a new entry.
4. Select the add-on you are registering, and enter its license key.

Note

If the EBX5 repository is under a trial license, no license key is required for the add-on. The add-on will be subject to the same trial period as the EBX5 repository itself.

5. Click **Submit**.

Technical administration

CHAPITRE 60

Repository administration

Technical architecture

Overview

The main principles of the EBX5 technical architecture are as follows:

- A Java process (JVM) that runs EBX5 is limited to a single EBX5 repository. This repository is physically persisted in a [supported relational database instance](#) accessed through a [configured data source](#).
- A repository cannot be shared by multiple JVMs at any given time. However, a failover architecture may be used. These aspects are detailed in the subsequent sections [Single JVM per repository](#) and [Failover with hot-standby](#). Furthermore, to achieve horizontal scalability, an alternative is to deploy a [distributed data delivery \(D3\)](#) environment.
- A single relational database instance can support multiple EBX5 repositories (used by distinct JVMs). This is done by specifying distinct table prefixes using the property `ebx.persistence.table.prefix`.

Voir aussi :

- [Configuring the EBX5 repository](#)
- [Supported databases](#)
- [Data source of the EBX5 repository](#)

Rules for the access to the database and user privileges

Attention

In order to guarantee the integrity of persisted master data, **it is strictly forbidden to perform direct SQL writes to the database**, except in the specific use cases described in the section [SQL access to data in relational mode](#).

It is recommended for the database user specified by the [configured data source](#) to have 'create/alter' privileges on tables, indexes and sequences. This allows [automatic repository installation and upgrades](#).

If this is not the case, it is still possible to perform [manual repository installation and upgrades](#), however, [Relational mode](#) and [History](#) will be inconvenient to maintain since modification of concerned models will require manual intervention.

Voir aussi :

- [SQL access to history](#)
- [Accessing a replica table using SQL](#)
- [SQL access to data in relational mode](#)
- [Data source of the EBX5 repository](#)

Single JVM per repository

A repository cannot be shared by multiple JVMs. If such a situation were to occur, it would lead to unpredictable behavior and perhaps even to corruption of data in the repository.

EBX5 performs checks to enforce this restriction. Before the repository becomes available, the repository must first acquire initial exclusive ownership on the relational database. After starting the repository, the JVM periodically checks that it still holds ownership of the repository.

These checks are executed by repeatedly tagging a technical table in the relational database. A shutdown command for the application server ensures that the tag on this technical table is removed. If the server shuts down abruptly, the tag may be left in the table. If this occurs, the server has to wait a few seconds upon restart, to ensure that the table is not being updated by another live process.

Attention

To avoid a wait period at the next start up, it is recommended to always cleanly shut down the application server.

Failover with hot-standby

The exclusion mechanism described above is compatible with failover architectures, where only one server is active at any given time (active/passive cluster). To ensure this is the case, the main server must declare the property `ebx.repository.ownership.type=failovermain`. The main server thus claims the repository database, as in the case of a single server.

A backup server can start, but it will not have access to the repository. It must declare the property `ebx.repository.ownership.type=failoverstandby` to act as the backup server. Once started, the backup server is registered in the connection log. Its status can be retrieved using the Java API or by an HTTP request, as described in the section [Repository status information and logs](#) below.

In order to activate the backup server and transfer exclusive ownership of the repository to it, a specific request must be issued by an HTTP request or using the Java API:

- Using HTTP, the request must include the parameter `activationKeyFromStandbyMode`, and the value of this parameter must be equal to the value declared for the entry `ebx.repository.ownership.activationkey` in the [EBX5 main configuration](#).
- Using the Java API, call the method **wakeFromStandby** [`RepositoryStatus`]^{API}.

The backup server then requests a clean shutdown for the main server, allowing any running transactions to finish. Only after the main server has yielded ownership may the backup use the repository.

Repository status information and logs

A log of all attempted Java process connections to the repository is available in the Administration area under 'History and logs' > 'Repository connection log'.

The status of the repository may be retrieved using the methods in the **RepositoryStatus** [RepositoryStatus]^{API} API.

It is also possible to get repository status information using a HTTP request that includes the parameter `repositoryInformationRequest` with one of following values:

state	<p>The state of the repository regarding the registration of ownership.</p> <ul style="list-style-type: none"> • D: Java process has been stopped. • O: Java process has taken exclusive ownership of the database. • S: Java process has been started in failover standby mode, but is not yet allowed to interact with the repository. • N: Java process has failed to take ownership of the database because another process was holding it.
heart_beat_count	<p>The number of times that the repository has made contact since associating with the database.</p>
info	<p>Detailed information for the end-user about the repository's registration status. The format of this information may be subject to modifications in future without explicit warning.</p>

Repository installation and upgrade

Automatic installation and upgrades

If the specified user for EBX5 data source has the rights to create and alter tables, indexes and sequences in the relational database, then the repository installation or upgrade is done automatically. Otherwise, you must see the procedure in the next section [Manual installation or upgrades](#).

Note

Concerning relational mode and history, declaring a table in relational mode or history mode implies the creation of a dedicated table in the database. Similarly, in these modes, modifying the structure of an existing table implies altering its declaration in the database (note that this action may need to take place at any time, independent of repository creation). The EBX5 applicative user must have 'create/alter' privileges on tables and indexes, since these actions must be carried out from EBX5.

Manual installation or upgrades

If the specified user for EBX5 data source does *not* have the rights to create or alter tables, indexes or sequences in the relational database, then the administrator must execute the SQL scripts located in the `ebx.software/files/ddl/xxx` folder, where `xxx` is the name of the target RDBMS.

If manually upgrading an existing repository, the administrator must execute the required subset of these SQL scripts; the file name of each script indicates the EBX5 version to which the patch applies.

Note:

- If a specific table prefix is specified by the property `ebx.persistence.table.prefix`, the default `EBX_` prefix must be renamed accordingly in the provided scripts.

Voir aussi : [Configuring the EBX5 repository](#)

Repository backup

Global backup of EBX5 repository must be delegated to the underlying RDBMS. The database administrator must use the standard backup procedures of the underlying database.

Archives directory

Archives are stored in a sub-directory named `archives` into `ebx.repository.directory` (see [configuration](#)). This directory is automatically created during the first export.

When manually creating this directory, be careful of the access rights: the EBX5 process must have read-and-write access to it. Furthermore, cleaning the directory is not done by EBX5 and is the responsibility of the administrator.

Note

The file transferred between two EBX5 environments must be done externally from the product. Tools such as FTP or simple file copies by network sharing can be used.

Repository attributes

A repository has the following attributes:

- *repositoryId*. It uniquely identifies a repository (at least in the scope of the enterprise): it is 48 bits (6 bytes), and it is usually represented as 12 hexadecimal digits. This information is used for generating UUID (Universally Unique Identifier) of entities created in the repository and also of transactions logged in the history tables or the XML audit trail (it plays the role of the 'UUID node' part, as specified by *RFC 4122*).
- *repository label*. It provides a label for the user so that he knows the purpose and context of the repository (for example, «production environment»).
- *store format*. It identifies the underlying persistence system, including the current version of its structure.

Monitoring and cleanup of relational database

Some entities are accumulated during the execution of EBX5. It is the *administrator's responsibility* to monitor and to cleanup these entities.

General considerations when using data spaces

[Data spaces](#) are an invaluable tool for managing complex data life cycles. While this feature brings great flexibility, it also implies a certain overhead cost, which should be taken into consideration for optimizing usage patterns.

As an extreme example, consider the case where every transaction triggers the following actions: a data space is created, and the transaction modifies some data; the data space is merged, closed, then deleted. In this case, no future references to the data space are needed, so using it to make isolated data modifications is unnecessary. In this case, using **Procedure** [Procedure]^{API} already provides sufficient isolation to avoid conflicts from concurrent operations.

For a developer-friendly analogy, this is like using a source-code management tool (CVS, SVN, etc.): when you need to perform a simple modification impacting only a few files, you probably do it directly on the main branch. In fact, it would be neither practical nor sustainable, with regard to file tagging, if every file modification involved branching the whole project, modifying the files, then merging the dedicated branch.

On larger repositories, creating data spaces can have a significant impact on the size of the repository persisted and the amount of maintenance required. As such, avoid creating data spaces unnecessarily.

Monitoring and reorganization

The persistence data source of the repository must be monitored through RDBMS monitoring.

The EBX5 tables specified in the [default semantic mode](#) have their content persisted in a set of generic database tables. They are the following:

- The table `{ebx.persistence.table.prefix}HOM` in which each record represents a data space or a snapshot (its name is `EBX_HOM`, if the property `ebx.persistence.table.prefix` is unset).
- The table `{ebx.persistence.table.prefix}BRV` where the data of EBX5 tables in semantic mode are segmented into blocks of at most 100 EBX5 records (its name is `EBX_BRV`, if the property `ebx.persistence.table.prefix` is unset).
- The table `{ebx.persistence.table.prefix}HTB`, which defines which blocks belong to a given EBX5 table in a given data space or snapshot (its name is `EBX_HTB`, if the property `ebx.persistence.table.prefix` is unset).

Note

For repositories having large volumes of data in semantic mode and on which frequent or heavy updates occur, it may be necessary to schedule a regular reorganization of the above database tables `...HTB` and `...BRV`, and their indexes. This is especially true for large repositories where many data spaces are created and deleted.

Cleaning up data spaces, snapshots, and history

A full clean-up of data spaces, snapshots, and history from the repository involves several stages:

1. Closing unused data spaces and snapshots to keep the cache to a minimal size.
2. Deleting data spaces, snapshots, and history.
3. Purging the remaining entities associated with the deleted data spaces, snapshots, and history from the repository.

Closing unused data spaces and snapshots

In order to keep the cache and the repository to a reasonable size, it is recommended to close any data spaces and snapshots that are no longer required. This can be done in the following ways:

- Through the user interface, in the Data Spaces area.
- From the 'Data spaces / Snapshots' table under 'Data spaces' in the Administration area, using the **Actions** menu in the workspace. The action can be used on a filtered view of the table.
- Through the Java API, using the method `closeHome [Repository]API`.
- Using the data service close data space and close snapshot operations. See [Closing a data space or snapshot](#) for more information.

Once the data spaces and snapshots have been closed, the data and associated history can be cleaned from the repository.

Note

Closed data spaces and snapshots can be reopened in the Administration area, under 'Data spaces'.

Deleting data spaces, snapshots, and history

Data spaces, their associated history, and snapshots can be permanently deleted from the repository. After you have deleted a data space or snapshot, some entities will remain until a repository-wide purge of all obsolete data is performed. Thus, both stages, the deletion and the repository-wide purge, must be performed in order to completely remove the data and history. This process has been separated into two steps for performance considerations. As the total clean-up of the repository can be time-intensive, this allows the purge execution to be initiated during off-peak periods on the server.

Note

The process of deleting the history of a data space takes into account all history transactions recorded up until the point that the deletion is submitted. Any subsequent historized operations will not be included when you run the purge operation. If you want to delete these new transactions, you must delete the history of that data space again.

The deletion of data spaces, snapshots, and history can be performed in a number of different ways:

- Using the dedicated 'Create deletion requests' interface, accessible from the Administration **Actions** menu in the Administration area. This interface presents all items available for deletion using hierarchical views, categorized into the following entries in the navigation panel:

Closed data spaces	Displays all data spaces and snapshots that are currently closed, and thus can be deleted. You can choose to delete the history associated with data spaces at the same time.
Open data spaces	Displays all data spaces that are currently open, for which you can delete the associated history.
Deleted data spaces	Displays all data spaces that have already been deleted, but have associated history remaining in the repository. You may delete this remaining history.

- From the 'Data spaces / Snapshots' table under 'Data spaces' in the Administration area, using the **Actions** menu in the workspace. The action can be used on a filtered view of the table.
- Using the Java API, specifically, the methods **deleteHome** [Repository]^{API} and **markHomeForHistoryPurge** [RepositoryPurge]^{API}.
- At the end of data service close data space operation, using the parameters `deleteDataOnClose` and `deleteHistoryOnClose`, or at the end of the merge data space operation, using the parameters `deleteDataOnMerge` and `deleteHistoryOnMerge`.

Purging remaining entities after data space, snapshot, and history deletion

Once items have been deleted, a purge can be executed to clean up the remaining data from *all* deletions performed up until that point. A purge can be initiated in the following ways:

- Through the user interface, by selecting Administration **Actions** > **Execute purge** in the navigation pane.
- Using the Java API, specifically the method **purgeAll** [RepositoryPurge]^{API}.
- Using the task scheduler. See [Task scheduler](#) for more information.

The purge process is logged into the directory `${ebx.repository.directory}/db.purge/`.

Cleaning up other repository entities

It is the *administrator's responsibility* to monitor and to regularly cleanup the following entities.

Cleaning up database resources of mapped tables

EBX5 gives the ability to purge mapped tables wherever it detects a mapped table in the database that is no longer used. This means that mapped mode must be deactivated before the mapped table database resources can be purged.

To deactivate mapped mode for a table, follow the procedure below.

For history mode:

- Deactivate historization of the table in the data model, or
- Remove the table from the data model

For relational mode:

- Remove the table from the data model, or
- Deactivate the relational mode on the data model. As data models in semantic mode cannot be used for relational data spaces, it is thus necessary to create a data set on a semantic data space using this modified data model. EBX5 will then detect that relational mode was previously used, and therefore propose the relational table database resources for purge.

Once mapped mode has been deactivated, you can perform a clean-up procedure similar to the process described in [Deleting data spaces, snapshots, and history](#). To select the tables to clean up, open the 'Create deletion requests' interface, accessible from the Administration **Actions** menu in the Administration area. Select 'Database tables' in the navigation pane.

A purge can then be executed to clean up the remaining data from *all* deletions, that is, deleted data spaces, snapshots, history, and database resources, performed up until that point. A purge can be initiated by selecting Administration **Actions** > **Execute purge** in the navigation pane.

Task scheduler execution reports

Task scheduler execution reports are persisted in the 'executions report' table, in the *Task scheduler* section of the Administration area. This table is constantly added to as scheduled tasks are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

User interactions

User interactions are used by the EBX5 component as a reliable means for an application to initiate and get the result of a service execution. They are persisted in the *ebx-interactions* administration feature. It is recommended to regularly monitor the user interactions table, as well as to clean it, if needed.

Workflow history

The workflow events are persisted in the workflow history table, in the *Workflow* section of the Administration area. This table is constantly added to as data workflows are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

Monitoring and cleanup of file system

In order to guarantee the correct operation of the software, the disk usage and disk availability of the following directories must be supervised:

- XML audit trail: `${ebx.repository.directory}/History/`
- archives: `${ebx.repository.directory}/archives/`
- logs: `ebx.logs.directory`
- temporary directory: `ebx.temp.directory`

Attention

- The cleaning of the above directories is not ensured by EBX5: *this is the administrator's responsibility*.
- For XML audit trail, if large transactions are executed with full updates detail activated (the default setting), *the disk space needed can be quite large*. For more information, see [XML audit Trail](#) section.

See also [configuration / tuning](#) section.

CHAPITRE 61

Front end administration

Several administrative tasks can be performed from the *Administration* section of EBX5.

Administration tools

EBX5 offers tools for its administration.

System information

This tool lists information about EBX5 configuration, repository information and operating system information.

Modules and data models

This tool lists all the registered modules as well as the existing data models.

User sessions

This tool lists the user sessions and allows the termination of active sessions.

Data spaces administration

Some data space administrative tasks can be performed from the *Administration* section of EBX5 by selecting the *Data spaces* administration feature.

Data spaces / Snapshots

This table lists all the existing data spaces and snapshots in the repository, whether open or closed. You can view and modify the information of data spaces included in this table.

Voir aussi : [Data space information](#)

It is also possible to close open data spaces, reopen closed data spaces, as well as delete and purge open or closed data spaces, associated history, and snapshots.

Voir aussi : [*Cleaning up data spaces, snapshots, and history*](#)

Data space permissions

This table lists all the existing permission rules defined on every data space in the repository. Each permission rule can be accessed and its information modified.

Voir aussi : [*Permissions*](#)

Repository history

The table *Deleted Data space / Snapshot* lists all the data spaces that have been purged from the repository.

Directory administration

Voir aussi : [*Users and roles directory*](#)

Policy

These properties configure the policies of the user and roles directory, for example, whether or not users can edit their own profiles.

Users table

This table lists all the user defined in the internal directory. New users can be added from there.

Roles table

This table lists all the user defined in the internal directory. New roles can be created from there.

Global permissions

Global permission rules can be created in EBX5.

These rules allow the restriction of access to sections of the software by hiding them from the interface. They can be accessed, created and/or modified by selecting the *Global permissions* Data Set.

Profile	Indicates the profile affected by the rule.
Restriction policy	Indicates if the permissions defined here restricts the ones defined for other profiles. See the Restriction policy notion for more information.
Modeling DMA	Defines permissions for accessing data modeling section.
Modeling Workflow	Defines permissions for accessing workflow modeling section.
Data Space	Defines permissions for accessing data space section..
Data Services	Defines permissions for accessing data services section.
Administration	Defines permissions for accessing administration section.

User interface administration

Some options are available to configure the web interface from the *Administration* section then selecting the *User interface* data set in the navigation pane.

Attention

Pay special attention to "URL Policy" configuration! If the web interface configuration results in a non usable application, use the "rescue mode" by setting `frontEnd.rescueMode.enable=true` in [EBX5 main configuration file](#), and using the following URL in your browser, with a built-in administrator user: `http://.../ebx/?onwbpID=iebx-manager-rescue`.

Session configuration

User session default locale	Default session localization.
Session time-out (in seconds)	Maximum time of inactivity of the end-user before he will be considered as inactive and its session will be terminated. A negative time indicates the session should never timeout.

Interface configuration

Entry policy

Describes the URL to access the application.

Login URL	If the user is not authenticated, the session is forwarded to this url.
------------------	-------------------------------------------------------------------------

It defines, for this configuration, EBX5 login page, replacing the default one.

If defined,

- it even replaces the authentication url that may have been defined by the mean of a specific user **Directory** [Directory]^{API},
- it is used to build the permalinks in the user interface,
- if the url is full (starting with http:// or https://), it replaces the url of the workflow email configuration.

URL policy

Describes URL and proxy policy. Both dynamic (servlet) and static (resources) URLs can be configured.

HTTP Servlet Policy	Header Content of Servlet HTTP request: <ul style="list-style-type: none"> • if a field is not set, the default value is chosen (in environment configuration), • if a default value is not set, the value carried by the initial request is chosen.
HTTPS Servlet Policy	Header Content of Servlet HTTPS request: <ul style="list-style-type: none"> • if a field is not set, the default value is chosen (in environment configuration), • if a default value is not set, the value carried by the initial request is chosen.
HTTP External resource Policy	Header Content of External resource URL in HTTP: <ul style="list-style-type: none"> • if a field is not set, the default value is chosen (in environment configuration), • if a default value is not set, the value carried by the initial request is chosen.
HTTPS External resource Policy	Header Content of External resource URL in HTTPS: <ul style="list-style-type: none"> • if a field is not set, the default value is chosen (in environment configuration), • if a default value is not set, the value carried by the initial request is chosen.

Exit policy

Describes how to exit the application.

Normal redirection	Specifies the URL redirection address used when exiting the session normally.
Error redirection	Specifies the URL redirection address used when exiting the session with error.

Graphical interface configuration

Application locking

EBX5 availability status:

Disponibilité	Pour maintenance, l'application peut être fermée au public (mais toujours accessible aux administrateurs). Les paramètres définis sont appliqués immédiatement.
Message d'indisponibilité	Message affiché aux utilisateurs quand l'accès est restreint aux administrateurs.


Security policy

EBX5 access security policy. These parameters only apply on new HTTP sessions.

Restriction d'accès IP	Restreindre l'accès aux adresses IP déclarées (voir ci-dessous).
Description de restriction IP	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.*.*.
Unicité de session	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:

Nombre maximal de colonnes d'une table	En fonction des performances du réseau et du navigateur, ajustez le nombre maximal de colonnes d'une table à afficher (dans le contenu d'un jeu de données). Cette propriété n'est pas prise en compte lorsqu'une vue est appliquée sur une table.
Largeur automatique maximale des colonnes de tables	Cette valeur définit une largeur automatique maximale pour chaque colonne. Ceci permet d'éviter que les colonnes avec un contenu très long (tel qu'une URL) ne prennent trop de largeur. La largeur des colonnes reste modifiable avec la souris au delà de cette valeur.
Nombre maximal d'éléments développés d'une hiérarchie	Définit, pour les hiérarchies, la limite du nombre d'éléments qui peuvent être développés par l'action "Développer tout". Une valeur inférieur ou égale à 0 désactive ce paramètre.
Filtre de table sélectionné par défaut	Définit le filtre de table sélectionné par défaut dans la liste des filtres affichés avec la vue tabulaire. En cas de modification, les utilisateurs doivent se déconnecter et se reconnecter afin d'utiliser la nouvelle valeur.
Afficher la boîte de message automatiquement	Defines the message severity threshold for displaying the messages pop-up.
Mode de compatibilité IE	<p>Defines whether or not to compensate for Internet Explorer 8+ displaying EBX5 in compatibility mode.</p> <p>In order to prevent Internet Explorer browsers from using compatibility mode when displaying the repository user interface, the meta-tag <i>http-equiv="X-UA-Compatible" content="IE=EmulateIE8"</i> is added to the header of pages. However, in some local environments, this setting may conflict with existing policies, in which case this header must be omitted by setting the parameter to 'No'. The default value is 'Yes'.</p> <p>See Specifying Document Compatibility Modes  for more information.</p>
Formulaires : largeur des libellés	La largeur des libellés des champs dans les formulaires.
Formulaires : largeur des champs	La largeur des champs de saisie dans les formulaires.
Formulaires : largeur de l'éditeur HTML	La largeur de l'éditeur HTML dans les formulaires.

Formulaires : hauteur de l'éditeur HTML	La hauteur de l'éditeur HTML dans les formulaires.
Formulaires : hauteur des champs texte	La hauteur des champs de saisie de type text dans les formulaires.
Sélection en liste : taille de page	Nombre de lignes affichées dans une page du composant de sélection dans une liste (utilisé pour sélectionner les clé étrangères, les énumérations, etc.).
Formulaire d'enregistrement : mode de présentation des noeuds	En fonction des performances du réseau et du navigateur, ajustez la manière d'afficher chaque noeud non terminal du formulaire d'enregistrement. En ce qui concerne le poids de la page téléchargée, le mode lien est léger, les modes développé et réduit sont plus lourds. En cas de modification de cette propriété, les utilisateurs doivent se déconnecter et se reconnecter afin que la nouvelle valeur soit prise en compte.

Default option values

Defines default values for options in the user interface.

Import / Export

CSV : jeu de caractères:	Définit le jeu de caractères par défaut utilisé pour les imports et les exports CSV.
Mode d'import :	Spécifie le mode d'import.
Valeurs XML manquantes à nul	Si 'Oui', si un nœud est manquant ou vide dans le fichier importé, la valeur est considérée comme 'nulle' lors de la mise à jour des enregistrements existants. Si 'Non', la valeur n'est pas modifiée.

Colors and themes

Customizes EBX5 colors and themes.

URL de l'icône du site (favicon)	Le format recommandé est ICO car il est compatible avec Internet Explorer.
URL du logo (SVG)	Laissez le champ vide pour utiliser l'image PNG. L'image SVG est utilisée sur les navigateurs compatibles. Le système utilisera le logo PNG si le navigateur n'est pas compatible. Si l'image PNG n'est pas renseignée, l'image GIF/JPG sera utilisée. Le logo doit avoir une hauteur maximale de 40px. Si la hauteur est supérieure à 40px, elle sera rognée à 40px de hauteur. La largeur du logo détermine la position des boutons du bandeau.
URL du logo (PNG)	L'image PNG est utilisée sur les navigateurs compatibles, sinon le système utilisera l'image GIF/JPG. Laissez ce champ et le champ du logo SVG vide pour utiliser l'image GIF/JPG. Le logo doit avoir une hauteur maximale de 40px. Si la hauteur est supérieure à 40px, elle sera rognée à 40px de hauteur. La largeur du logo détermine la position des boutons du bandeau.
URL du logo (GIF/JPG)	L'image GIF/JPG est utilisée quand les images PNG et SVG ne sont pas renseignées, ou sur Internet Explorer 6. Les formats recommandés sont GIF et JPG. Le logo doit avoir une hauteur maximale de 40px. Si la hauteur est supérieure à 40px, elle sera rognée à 40px de hauteur. La largeur du logo détermine la position des boutons du bandeau.
Principale	Couleur principale de l'interface utilisateur, utilisée pour les sélections et les surbrillances.
Secondaire	Couleur secondaire de l'interface utilisateur, utilisée pour certains textes.
Bandeau	Couleur de fond du bandeau de l'interface utilisateur. Par défaut, définie à la même valeur que la couleur Principale.
Surbrillance du bandeau	Couleur de la surbrillance du bandeau. Par défaut, définie à la même valeur que la couleur Secondaire.
Pastille du workflow	Couleurs de fond et de texte/bordure de la pastille du workflow (compteur de workflows utilisateur).
Page de login (fond)	Couleur de fond de la page de login. Par défaut, définie à une version plus claire de la couleur Principale.

Boutons	Couleurs des boutons et luminosité du texte et de l'icone des boutons.
Surbrillance des boutons	Couleur des boutons en surbrillance et sélectionnés par défaut. Par défaut, définie à la même valeur que la couleur Principale.
Onglets	Couleur des onglets des formulaires.
Surbrillance des onglets	Couleur des onglets sélectionnées. Par défaut, définie à la même valeur que la couleur Principale.
Panneau de navigation	Couleur de fond du panneau de navigation.
Barre du bas du formulaire	Couleur de fond de la barre du bas du formulaire.
Vue historique de table : données techniques	Couleur de fond des cellules de données techniques dans la vue historique de table.
Vue historique de table : création	Couleur de fond des cellules ayant l'état création dans la vue historique de table.
Vue historique de table : suppression	Couleur de fond des cellules ayant l'état suppression dans la vue historique de table.
Vue historique de table : mise à jour	Couleur de fond des cellules ayant l'état mise à jour dans la vue historique de table.

Lineage administration

Three tables are accessible:

- **Authorized profiles:** a profile must be added to this table to be used for data lineage WSDL generation.
- **History:** lists the data lineage WSDL generated and their configuration.
- **JMS location:** lists the JMS URL locations.

Other technical tables

Several technical tables can be accessed through the *Administration* section of EBX5. These tables are for internal use and/or historizes uses and their content should not be edited, except for removing obsolete or erroneous data.

Auto-increments	This table lists the defined auto-increments field.
Interaction	This table lists the defined interaction.
User preferences	This table lists the preferences of each user.
Custom views	This table lists the custom views defined by users.
User filters	This table lists the filters defined by users.
Default views	This table lists the default view preferences set by users.
Workflows	This table lists the all data workflows and the information associated with each.
Tokens	This table lists all data workflow tokens.
Work items	This table lists all work items and the information associated with each.
History table	This table lists all data workflow history.

CHAPITRE 62

Users and roles directory

Overview

EBX5 uses a directory for user authentication and user role definition.

A default directory is provided and integrated into EBX5 repository. It is also possible to integrate with any specific enterprise directory.

Voir aussi : [Configuring the user and roles directory](#)

Concepts

In EBX5, a user can participate in several roles, and a role can be shared by several users. Moreover, a role can be included in another role. The generic term "profile" describes either a user or a role.

In addition to the directory-defined roles, EBX5 provides the following *built-in roles*:

Role	Definition
Profile.ADMINISTRATOR	Built-in Administrator role. Administrator role allows performing general administration tasks.
Profile.READ_ONLY	Built-in read-only role. A user associated with role read-only has no rights for doing any modifications on EBX5 repository. He can only visualize the repository.
Profile.OWNER	Dynamic built-in owner role. This role is checked dynamically for the current instance. It is activated only if the user belongs to the profile defined as owner of this current instance.
Profile.EVERYONE	All users have this role.

Information related to profiles is mainly defined in the directory. However, an association between a user and one of the last two roles (*OWNER*, *EVERYONE*) must not be managed by the directory, since EBX5 is assigned to perform this task automatically. User permissions are managed independently of the directory (see chapter [Permissions](#)).

Voir aussi :

- [profile](#)
- [role](#)
- [user](#)
- [administrator](#)
- [user and roles directory](#)

Default directory

Directory content

The default directory is represented by the data set named *Directory*, accessible through the 'Administration' area.

This data set contains two tables: one for users and one for roles. By default, only the administrator is allowed to modify the directory. However, users can modify all information related to their own accounts, except for the associated roles.

Note

It is not possible to delete or duplicate the default directory.

Password recovery procedure

In the default directory, passwords are encrypted (by default with a SHA256-like algorithm), and stored in this state. Consequently, it is impossible to retrieve lost passwords. A new password must be generated and sent to the user.

There are two options for this procedure:

1. A notification email is sent to the administrator, the administrator manually changes the password and sends it to the user.
2. A procedure automatically generates a new password and sends it to the user.

By default, the first option is used. For activating the second option, the file [ebx.properties](#) must specify the property:

```
ebx.password.remind.auto=true.
```

Note

For security reasons, the password recovery procedure is not available for administrator profiles. If required, use administrator recovery procedure instead.

Administrator recovery procedure

If an administrator has lost a user name or password, a special procedure must be followed. A specific directory class redefines an administrator user with the login "admin" and the password "admin". To activate this procedure, the file [ebx.properties](#) must specify the following:

```
ebx.directory.factory=\n\ncom.orchestranetworks.service.directory.DirectoryDefaultRecoverFactory
```

Once EBX5 has been restarted, then stopped, this line must be set back to its default value.

Custom directory

As an alternative to the default directory, it is possible to integrate a specific enterprise directory. For example, an LDAP instance, a relational database or a specific directory model instantiated into EBX5.

For more information, see **DirectoryFactory** [DirectoryFactory]^{API} in the Java API.

CHAPITRE 63

Audit Trail

Overview

XML audit trail is a feature that allows logging updates into XML files (an alternative history feature is available, see [History](#) documentation for more details).

Any persistent update performed on the EBX5 repository is logged to an audit trail XML file. This is also the case for any procedure execution (even if it does not perform an update, a procedure is always considered as a transaction). The following information is logged:

- the transaction type (data set creation, record modification, record deletion, specific procedure, etc.);
- the data space or snapshot on which the transaction is executed;
- the transaction source: if the action is initiated by EBX5, this source is described by the user identity, HTTP session identifier and client IP address; if it is initiated programmatically, only the user identity is shown;
- also concerning the session, the optional value "trackingInfo";
- the transaction date and time (in milliseconds);
- the transaction UUID (conforming to Leach-Salz variant, version 1);
- if the transaction has failed, information about the error;
- the detail of updates done (if there are any updates and if history detail is activated, see next section).

Updates detail and disk management

The audit trail is able to describe all updates made into the EBX5 repository, at the finest level. Hence, the files can be quite large and the audit trail directory must be supervised with caution. The following facts should be taken into account:

1. If an archive import is executed in non-interactive mode (without a change set), the audit trail does not detail the updates: it only specifies the archive that has been imported. In this case, if it is important to keep a fine trace of the import-replace, the used archive must be preserved.
2. If an archive import is executed in interactive mode (with change set), or if a data space is merged to its parent, the resulting log size will be nearly equal to the unzipped size of the archive multiplied by three. Furthermore, for consistency issues, each transaction is logged into a temporary file (in the audit trail directory) before being moved into the main file. So EBX5 requires *at least six times the unzipped size of the largest archive that can be imported*.
3. In the context of a specific procedure that performs many updates that need not be audited, it is still possible for the developer to disable the detailed history with the method **ProcedureContext.setHistoryActivation(boolean)** [ProcedureContext]^{API}.

Voir aussi : [EBX5 monitoring](#)

Files organization

All audit trail files are stored in the directory `${ebx.repository.directory}/History`.

"Closed" audit files

Each file is named as:

`aaaa-mm-dd-partnn.xml`

where `aaaa-mm-dd` is the file date and `nn` is the file index in this day.

Current audit files for writing

When an audit file is being written, the XML structure implies to work in an "open mode". XML elements of the modifications are added to a text file named:

`aaaa-mm-dd-partnnContent.txt`

The standard XML format is still available in an XML file that references the text file. This file is named:

`aaaa-mm-dd-partnnRef.xml`

Those two files are then re-aggregated in a "closed" XML file when the repository is cleanly shut down or EBX5 is restarted.

Example of an audit directory:

```
2004-04-05-part00.xml
2004-04-05-part01.xml
2004-04-06-part00.xml
2004-04-06-part01.xml
2004-04-06-part02.xml
2004-04-06-part03.xml
2004-04-07-part00.xml
2004-04-10-part00.xml
2004-04-11-part00Content.txt
2004-04-11-part00Ref.xml
```


CHAPITRE 64

Data model administration

Administering publications and versions

Technical data about data model publications and versions can be accessed in the *Administration* section by an administrator.

Data Modeling contains the two following tables:

- *Publications*. Stores the publications available in the repository.
- *Versions*. Stores the versions of the data models available in the repository.

These tables are in read-only but it is however possible to delete manually a publication or a version.

Important: If a publication or a version is deleted then the content of associated data sets will become unavailable. So these technical data must be deleted with caution.

It is possible to spread these technical data to other EBX5 repositories exporting an archive from a EBX5 repository and importing it to another one. It may be useful for propagating the evolutions of data models to other repositories.

Migration of previous data models in repository

In versions before 5.2.0, published data models not depending on module were generated in the file system directory `${ebx.repository.directory}/schemas/`, with the name of the data model (*product.xsd* for example if the data model is named *Product*). Since the 5.2.0, this kind of data model is now fully managed inside EBX5 through *Publications*. That is, republishing an existing data model migrates it as a *Publication* and redirects linked data sets to the new embedded data model. The previous XML Schema Document located in `${ebx.repository.directory}/schemas/` is renamed and suffixed with *toDelete* meaning that the document is no more used and can be safely deleted.

CHAPITRE 65

Data workflow administration

To define general parameters for data workflow execution, manage workflow publications, and oversee data workflows in progress, navigate to the Administration area. Click the down arrow in the navigation pane and select the entry 'Workflow'.

Execution of workflows and history

Several tables facilitate managing the data workflows currently in progress. They are accessible by navigating to **root > Execution of workflows and history** in the navigation pane.

Workflows table

The 'Workflows' table contains instances of all data workflows in the repository. A data workflow is a particular execution instance of a workflow model publication. This table provides access to the data context variables for all data workflows. It can be used to access the status of advancement of the data workflow in terms of current variable values, and in case of data workflow suspension, to modify the variable values.

Tokens table

The 'Tokens' table allows the management of data workflows advancement. Each token marks the current step of execution of a data workflow in progress, as well as the state of the data workflow.

Voir aussi : [token](#)

Work items table

The 'Work items' table contains all the work items associated with user tasks that currently exist. If necessary, you can manually allocate a work item to a user from this table in the case of a blockage in a data workflow through this table. It is preferable, however, to use the buttons in the workspace of the 'Data Workflows' area whenever possible to allocate, reallocate, and deallocate work items.

Voir aussi : [work item](#)

History table

The 'History' table contains all actions that have been performed during the execution of workflows. The table is especially useful to view the states of various objects related to workflows at a given moment. This includes actions on work items, variables in the data context, as well as tokens. In case of error, a technical log is available in addition to the contents of the history table.

From the 'Services' menu in the workspace, you can clear the history of completed data workflows or data workflows older than a certain date.

Note

In cases where unexpected inconsistencies have arisen between the workflow execution technical tables, data workflows may encounter errors. It may then be necessary to run the operation 'Clean up inconsistencies in workflow execution tables' from the **Services** menu in the navigation panel under Administration > Workflows.

Interactions

An *interaction* is generated automatically for every work item that is created. It is a local data context of a work item and is accessible from an EBX5 session. When a work item is executed, the user performs the assigned actions based upon its interaction, independent of the workflow engine. User tasks define mappings for their input and output parameters to link interactions with the overall data contexts of data workflows.

Interactions can be useful for monitoring the current parameters of work items. For example, an interaction can be updated manually by a trigger or a UI service.

Workflow publications

The 'Workflow publications' table is a technical table that contains all the workflow model publications in the repository. This table associates published workflow models with their snapshots. It is not recommended to directly modify this table, but rather to use the actions available in the 'Workflow Modeling' area to make changes to publications.

Configuration

The 'Workflow publications' table is a technical table that contains all the workflow model publications in the repository. This table associates published workflow models with their snapshots. It is not recommended to directly modify this table, but rather to use the actions available in the 'Workflow Modeling' area to make changes to publications.

Email configuration

In order for email notifications to be sent during data workflow execution, the following settings must be configured under 'Email configuration':

- 'Activate email notification' must be set to 'Yes'.
- The 'From email' field must be filled in with the email address from to mark as the sender of notification emails.

Priorities configuration

The property 'Default priority' defines how data workflows and their work items across the repository appear if they have not set a priority level. For example, if this property is set to the value 'Normal', any workflow and work item with no priority will appear to have the priority 'Normal'.

The table 'priorities' defines all priority levels available to data workflows in the repository. You may add as many integer priority levels as required, along with their labels, which will appear when users hover over the priority icon in work item tables. You may also select the icons that correspond to each priority level, either from the set provided by EBX5, or by specifying a URL to an icon image file.

Temporal tasks

Under 'Temporal tasks', you can set the polling interval for time-dependent tasks in the workflow, such as deadlines and reminders. If no interval value is set, the steps in progress are checked every hour.

CHAPITRE 66

Task scheduler

Overview

EBX5 offers the ability to schedule programmatic tasks.

Note

In order to avoid conflicts and dead-locks, tasks are scheduled in a single queue.

Configuration from EBX5

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the Administration area:

- **Schedules:** define scheduling using "cron expressions".
- **Tasks:** configure tasks, including parameterizing task instances and user profiles their execution.
- **Scheduled tasks:** current schedule, including task scheduling activation/deactivation.
- **Executions report:** reports of each scheduled task run that appear immediately after the task is triggered. The reports include actions to interrupt, pause, or resume running tasks, when made available by the task definition.

Cron expression

(extract from [Quartz Scheduler](#) documentation)

The task scheduler uses "cron expressions", which are able to create firing schedules such as: "At 8:00am every Monday through Friday" or "At 1:30am every last Friday of the month".

Format

A cron expression is a string comprised of 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

Field Name	Mandatory	Allowed Values	Allowed Special Characters
Seconds	Yes	0-59	, - * /
Minutes	Yes	0-59	, - * /
Hours	Yes	0-23	, - * /
Day of month	Yes	0-31	, - * ? / L W
Month	Yes	1-12 or JAN-DEC	, - * /
Day of week	Yes	1-7 or SUN-SAT	, - * ? / L #
Year	No	0-59	empty, 1970-2099

A cron expression can be as simple as this: "*** * * * ? ***",

or more complex, like this: "**0/5 14,18,3-39,52 * ? JAN,MAR,SEP MON-FRI 2002-2010**".

Note

The legal characters and the names of months and days of the week are not case sensitive. MON is the same as mon.

Special characters

A cron expression is a string comprised of 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

- ***** ("all values") - used to select all values within a field. For example, "*" in the minute field means "every minute".
- **?** ("no specific value") - useful when you need to specify something in one of the two fields in which the character is allowed, but not the other. For example, if I want my trigger to fire on a particular day of the month (say, the 10th), but don't care what day of the week that happens to be, I would put "10" in the day-of-month field, and "?" in the day-of-week field. See the examples below for clarification.
- **-** - used to specify ranges. For example, "10-12" in the hour field means "the hours 10, 11 and 12".
- **,** - used to specify additional values. For example, "MON,WED,FRI" in the day-of-week field means "the days Monday, Wednesday, and Friday".
- **/** - used to specify increments. For example, "0/15" in the seconds field means "the seconds 0, 15, 30, and 45". And "5/15" in the seconds field means "the seconds 5, 20, 35, and 50". You can also specify '/' after the " **character - in this case** " is equivalent to having '0' before the '/'. '1/3' in the day-of-month field means "fire every 3 days starting on the first day of the month".
- **L** ("last") - has different meaning in each of the two fields in which it is allowed. For example, the value "L" in the day-of-month field means "the last day of the month" - day 31 for January, day 28 for February on non-leap years. If used in the day-of-week field by itself, it simply means "7" or "SAT". But if used in the day-of-week field after another value, it means "the last xxx day of the month" - for example "6L" means "the last friday of the month". When using the 'L' option, it is important not to specify lists, or ranges of values, as you'll get confusing results.
- **W** ("weekday") - used to specify the weekday (Monday-Friday) nearest the given day. As an example, if you were to specify "15W" as the value for the day-of-month field, the meaning is: "the nearest weekday to the 15th of the month". So if the 15th is a Saturday, the trigger will fire on Friday the 14th. If the 15th is a Sunday, the trigger will fire on Monday the 16th. If the 15th is a Tuesday, then it will fire on Tuesday the 15th. However if you specify "1W" as the value for day-of-month, and the 1st is a Saturday, the trigger will fire on Monday the 3rd, as it will not 'jump' over the boundary of a month's days. The 'W' character can only be specified when the day-of-month is a single day, not a range or list of days.

Note

The 'L' and 'W' characters can also be combined in the day-of-month field to yield 'LW', which translates to "*"last weekday of the month"*.

- **#** - used to specify "the nth" XXX day of the month. For example, the value of "6#3" in the day-of-week field means "the third Friday of the month" (day 6 = Friday and "#3" = the 3rd one in the month). Other examples: "2#1" = the first Monday of the month and "4#5" = the fifth Wednesday of the month. Note that if you specify "#5" and there is not 5 of the given day-of-week in the month, then no firing will occur that month.

Examples

Expression	Meaning
0 0 12 * * ?	Fire at 12pm (noon) every day.
0 15 10 ? * *	Fire at 10:15am every day.
0 15 10 * * ?	Fire at 10:15am every day.
0 15 10 * * ? *	Fire at 10:15am every day.
0 15 10 * * ? 2005	Fire at 10:15am every day during the year 2005.
0 * 14 * * ?	Fire every minute starting at 2pm and ending at 2:59pm, every day.
0 0/5 14 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day.
0 0/5 14,18 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at 6pm and ending at 6:55pm, every day.
0 0-5 14 * * ?	Fire every minute starting at 2pm and ending at 2:05pm, every day.
0 10,44 14 ? 3 WED	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
0 15 10 ? * MON-FRI	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday.
0 15 10 15 * ?	Fire at 10:15am on the 15th day of every month.
0 15 10 L * ?	Fire at 10:15am on the last day of every month.
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month.
0 15 10 ? * 6L 2002-2005	Fire at 10:15am on every last friday of every month during the years 2002, 2003, 2004 and 2005.
0 15 10 ? * 6#3	Fire at 10:15am on the third Friday of every month.
0 0 12 1/5 * ?	Fire at 12pm (noon) every 5 days every month, starting on the first day of the month.
0 11 11 11 11 ?	Fire every November 11th at 11:11am.

Note

Pay attention to the effects of '?' and '*' in the day-of-week and day-of-month fields!

Support for specifying both a day-of-week and a day-of-month value is not complete (you must currently use the '?' character in one of these fields).

Be careful when setting fire times between the hours of the morning when "daylight savings" changes occur in your locale (for US locales, this would typically be the hour before and after 2:00 AM - because the time shift can cause a skip or a repeat depending on whether the time moves back or jumps forward).

Task definition

EBX5 scheduler comes with a predefined task: "Repository clean-up".

Custom scheduled tasks can be added by the means of **scheduler** [package-summary]^{API} Java API.

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the Administration area.

Task configuration

A user must be associated to a task definition; this user will be used to generate the **session** [Session]^{API} that will run the task.

Note

The user will not be authenticated, and no password is required. As a consequence, a user with no password set in the directory can only be used to run scheduled tasks.

A custom task can be parameterized by means of JavaBean specification (getter and setter).

Supported parameter types are:

- java.lang.boolean
- java.lang.int
- java.lang.Boolean
- java.lang.Integer
- java.math.BigDecimal
- java.lang.String
- java.lang.Date
- java.net.URI
- java.net.URL

Parameter values are set in XML format.

Distributed Data Delivery (D3)

CHAPITRE 67

Introduction to D3

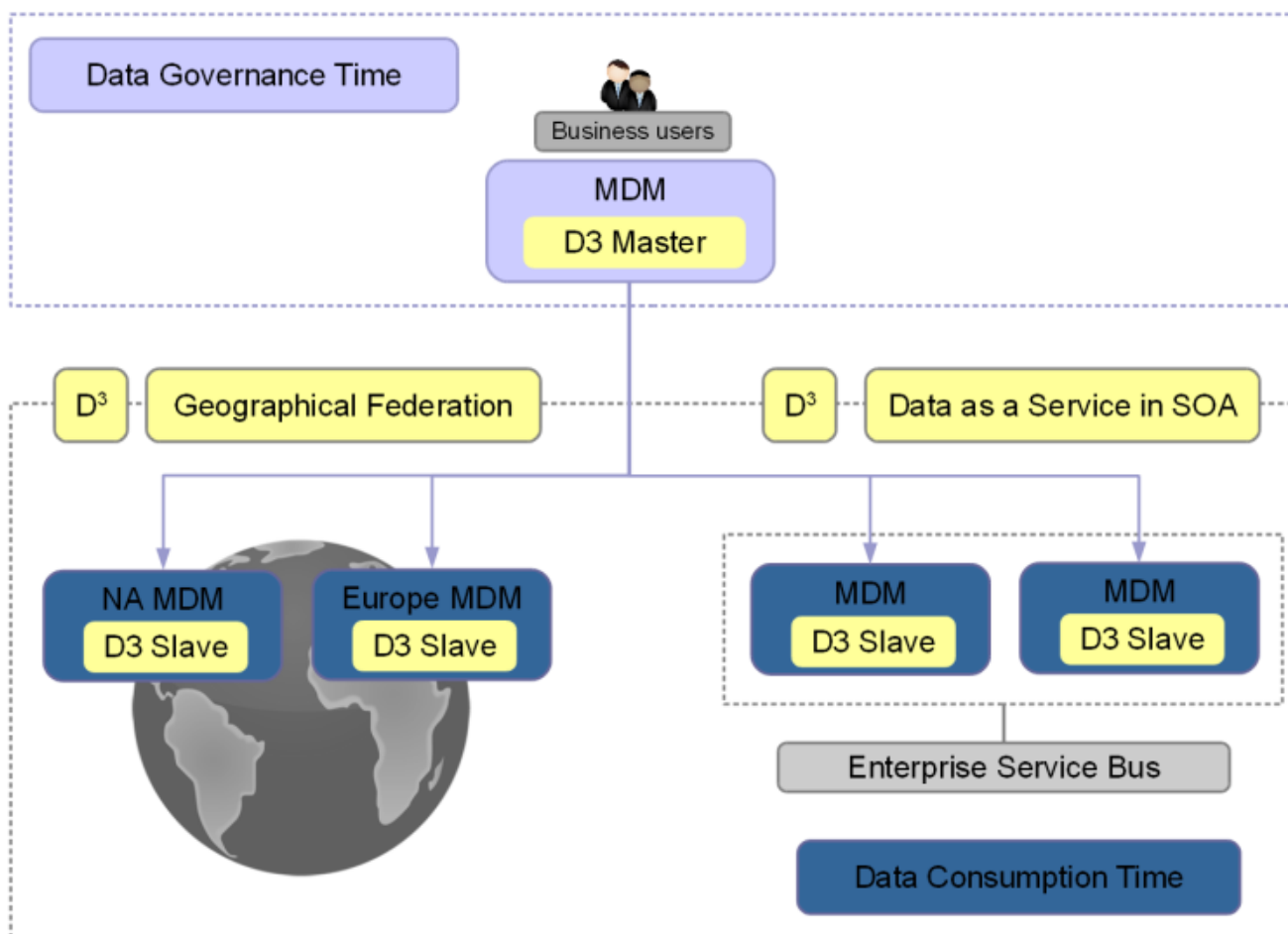
Overview

EBX5 offers the ability to replicate data from one EBX5 instance to several others. By means of the broadcast service, it also provides an additional layer of security and control to the other features of EBX5: it is particularly adapted when data governance requires the highest levels of data consistency, approval and/or capacity of rollback.

Architecture

A typical D3 installation consists of one master and several slaves. In the master, the Data Steward declares which data spaces must be replicated, and which user profile is allowed to broadcast them to the slaves. The Data Steward also defines delivery profiles, which are groups of one or more data spaces.

Each slave must define which delivery profile has to be replicated towards it.



Involvement of third-party systems

The features of D3 also allow access by third-party systems to data managed in EBX5 through Data Services. Essentially, when a system consumes the data of a delivery data space, the data is transparently redirected to the last Snapshot that has been broadcast. This ensures a more controlled and reliable view of the managed data.

Third-party systems can either access the master node or a slave node (hence a physical architecture involving a master node and no slave node is also an option).

Protocols

If JMS is activated, conversation between a master and a slave is based on SOAP over JMS, while archive transfer between a master and a slave is based on JMS binary messages.

If JMS is not activated, conversation between a master and a slave is based on SOAP over HTTP, while archive transfer (binary) between a master and a slave is based on TCP sockets.

Voir aussi : [JMS for distributed data delivery \(D3\)](#)

Glossary

Broadcast

The action broadcast is a way to "publish an official snapshot" of data. Underneath, it transparently and transactionally ensures that it is replicated to the slave nodes and becomes "the official truth" for all repositories (master and slaves).

Delivery data space

A delivery data space is a data space which can be broadcast to authenticated and allowed users, using a dedicated service.

By default, when a data service accesses a delivery data space (on any node), it is redirected to the last snapshot that has been broadcast. More information in the [Data Services](#) section.

Delivery profile

A delivery profile is a logical name that groups one or several delivery data spaces. The slaves subscribe to one or several delivery profiles.

Cluster delivery mode

Synchronization with subscribed slave nodes is performed in a two-phase commit transactional process. This delivery mode is designed to respond to a high volume of queries using load balancing and/or fault tolerance. It ensures the consistency of data in the cluster between slave nodes and their master delivery data spaces. Master and slave nodes use the same committed snapshots.

Federation delivery mode

Synchronization is performed in a single phase, and with each registered slave node independently. This delivery mode is designed to be used with geographically distributed and/or heterogeneous architectures where response time and network availability cannot be guaranteed. At any one time, slaves can be at different committed snapshot versions. The synchronization processes are thus independent of one another and replay individual slave nodes are performed for certain broadcast failures.

Master node

The master node is an instance of EBX5 that can define one or several delivery data spaces, and to which slave nodes can subscribe. Otherwise, the master node can also act as a regular EBX5 server.

Slave node

A slave node is an instance of EBX5 attached to a master node, in order to receive delivery data spaces replication. Except for update restrictions on the delivery/replicated data spaces, the slave node acts as a regular EBX5 server.

Hub node

A hub node is an instance of EBX5 acting as a master node and as a slave node.

Known limitations

General limitations

- Each slave node must have only one master.
- Embedded data models cannot be used in D3 data spaces. Therefore, it is not possible to create a data set based on a publication in a D3 data space.

Broadcast and delivery data space limitations

- Access rights on data spaces are not replicated (whereas access rights on data sets are).
- Data space information is not replicated.
- If a data space and its parent are replicated, this parent-child relationship will be lost in the slaves.
- If a data space is deleted from the master, it remains on the slaves.
- Re-broadcasting a snapshot where inherited data sets have been added or removed is not supported.
- Once a snapshot has been broadcast to a slave, subsequent broadcasts of *any* snapshot with the same name will result in restoring the originally broadcast version of that same name on the slave. That is, if the original snapshot on the master is purged and a new one is created with the same name and subsequently broadcast, then the content of the slave will be restored to that of the previously broadcast snapshot, and not the latest one of the same name.
- The data model on which the broadcast contents are based must be the same between the broadcast and the current version.

Administration limitations

- Technical data spaces cannot be replicated, thus the EBX5 default user directory cannot be synchronized through D3.

CHAPITRE 68

D3 broadcasts and delivery data spaces

Broadcast procedure

Scope and contents of a broadcast

A D3 broadcast occurs at the data space or snapshot level. For data space broadcasts, D3 first creates a snapshot to capture the current state, then broadcasts this newly created snapshot.

A broadcast performs one of the following procedures depending on the situation:

- An update of the differences computed between the new snapshot broadcast and the current 'commit' version on the slave.
- A full synchronization containing all data sets, tables, records, and permissions. This is done on the first broadcast to a given slave node, or if the previous slave commit is not known to the master node.

Performing a broadcast

The broadcast can be done:

- By the end user, using the action **Broadcast** available in the delivery data space or a snapshot of the delivery data space,
- Using a custom code that uses **D3Engine** [`D3Engine`]^{API} in the Java API.

Conditions

To be able to broadcast, the following conditions must be fulfilled:

- The authenticated user profile is granted permission to broadcast.
- The data space or snapshot to be broadcast has no validation errors.
- The D3 master configuration has no validation errors.
- The data space or snapshot must not contain any tables in relational mode.
- There must be an associated delivery profile.
- If broadcasting a data space, the data space must not be locked.
- If broadcasting a snapshot, the snapshot must belong to a data space declared as delivery data space and must not already be the current broadcast snapshot (but a rollback to a previously broadcast snapshot is possible).

Note

Broadcasts are performed asynchronously. Therefore, no information is displayed in the user interface about the success or failure of a broadcast. Nevertheless, it is possible to monitor the slaves and the deliveries. See [D3 supervision](#) for more information.

Delivery data spaces

Data Services

By default, when a data service accesses a delivery data space, it is redirected to the current snapshot, which is the last one broadcast. However, this default behavior can be modified either at the request level or the global configuration level.

Voir aussi : [Common parameter 'disableRedirectionToLastBroadcast'](#)

Access restrictions

On the master node, a delivery data space can neither be merged nor closed. Other operations are available, for example, according to permissions, it can be modified directly, snapshot can be created independently of a broadcast, or a child data space can be created and merged.

On the slave node, no modification can be made on a delivery/replicated data space, by any means (end user, data services, Java program) except broadcast process. Furthermore, any data space-related operation (such as merge, close, etc) is forbidden on the slaves.

Transactional view

The current snapshot may change between two calls if a broadcast has been made meanwhile. If a fully stable view is required for several successive calls, these calls will need to directly specify the same expected Snapshot.

CHAPITRE 69

D3 administration

Configuring D3 nodes

Runtime configuration of master or hub nodes through the user interface

The declaration of delivery data spaces and delivery profiles is done by selecting the '[D3] Master configuration' feature from the Administration area, where you will find the following tables:

Delivery data spaces	Declarations of the data spaces that can be broadcast.
Delivery profiles	Profiles to which slaves nodes can subscribe. The delivery mode must be defined for each delivery profile.
Delivery mapping	The association between delivery data spaces and delivery profiles.

Configuring master, hub and slave nodes

This section details how to configure a node in its EBX5 main configuration file.

Voir aussi : [Overview of the EBX5 main configuration file](#)

Master

In order to act as a *master* node, an instance of EBX5 must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=master` node:

```
#####
## D3 configuration
#####
# Configuration for both master and slave
#####
# Optional property.
```

```
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=master
```

Hub

In order to act as a *hub* node (combination of master and slave configurations), an instance of EBX5 must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=hub` node:

```
#####
## D3 configuration
#####
# Configuration for both master and slave
#####
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=hub

#####
# Configuration dedicated to hub or slave
#####
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

Slave

In order to act as a *slave* node, an instance of EBX5 must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=slave` node:

```
#####
## D3 configuration
#####
# Configuration for both master and slave
#####
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=slave
```

```
#####
# Configuration dedicated to hub or slave
#####
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

Configuring the network protocol of a node

This section details how to configure the network protocol of a node in its EBX5 main configuration file.

Voir aussi : [Overview of the EBX5 main configuration file](#)

HTTP and socket TCP protocols

Sample configuration for ebx.d3.mode=hub or ebx.d3.mode=slave node with HTTP network protocol:

```
#####
# HTTP and TCP socket configuration for D3 slaves
#####
# URL to access the data services connector of the master
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not
  activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: http://[master_host]:[master_port]/ebx-dataservices/
  connector
ebx.d3.master.url=

# URL to access the data services connector of the slave
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not
  activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: http://[slave_host]:[slave_port]/ebx-dataservices/
  connector
ebx.d3.slave.url=

# Minimum port to use to transfer archives on TCP mode.
# Must be a positive integer above zero and below 65535.
# If not set, a random port will be used.
#ebx.d3.slave.socket.range.min=

# Max port to use on TCP mode to transfer archives.
# Must be a positive integer above ebx.d3.slave.socket.range.min and below 65535.
# Mandatory if ebx.d3.slave.socket.range.min is set.
#ebx.d3.slave.socket.range.max=
```

JMS protocol

If JMS is activated, the following properties can be defined in order to enable JMS functionalities for a D3 node.

Sample configuration for all D3 nodes with JMS network protocol:

```
#####
## JMS configuration for D3
#####
# Taken into account only if Data Services JMS is configured properly
#####
# Configuration for master, hub and slave
#####
# Default is false, activate JMS for D3
## If activated, the deployer must ensure that the entry 'jms/EBX_D3ReplyQueue'
## are bound in the operational environment of the application server.
ebx.jms.d3.activate=false

# Change default timeout when use reply queue, default is 10000 milliseconds
#ebx.jms.d3.reply.timeout=10000
```

Voir aussi : [JMS for distributed data delivery \(D3\)](#)

Services on master nodes

Services to manage a master node are available in the Administration area of the slave node under '[D3] Master configuration' and also on the tables 'Delivery data spaces' and 'Registered slaves'. The services are as follows:

Relaunch replays	Immediately relaunch all replays for waiting federation deliveries.
Delete data space	Delete the delivery branch on registered slaves and remove it from the configuration of the D3 master.
Launch slave subscription	Send a request to the selected slaves to subscribe them.
Deactivate slaves	Remove the selected slaves from the broadcast scope and switch their states to 'Unavailable'.
Unregister slaves	Completely remove the selected slaves from the master.

Services on slave nodes

Services to manage a slave node's subscription to the master node are available in the Administration area of the slave node under '[D3] Slave configuration', in the navigation pane. The services are as follows:

Register slave	Re-subscribes the slave to the master if it has been unregistered.
Unregister slave	Disconnects the slave from the master.

Supervision

Master node management console

Several tables compose the management console for the master node, located in the Administration area of the master node, under '[D3] Master configuration'. They are as follows:

Registered slaves	Slaves registered with the master node. From this table, several services are available on each record.
Broadcast history	History of broadcast operations that have taken place.
Slave registration log	History of initialization operations that have taken place.
Detailed history	History of archive deliveries that have taken place. The list of associated delivery archives can be accessed from the tables 'Broadcast history' and 'Initialization history' using selection nodes.

Master node supervision services

Available in the administration area of the master node under '[D3] Master configuration'. The services are as follows:

Display and refresh state of slaves	Lists the slaves and related information, such as the slave's state, associated delivery profiles, and delivered snapshots.
Clear history content	Deletes all records in all history tables, such as 'Broadcast history', 'Slave registration log' and 'Detailed history'.

Slave monitoring through the Java API

A slave monitoring class can be created to implement actions that are triggered when the slave's status switches to either 'Available' or 'Unavailable'. To do so, it must implement the interface

`NodeMonitoring`. This class must be outside of any EBX5 module and accessible from the class-loader of 'ebx.jar' and its full class name must be specified under '[D3] Slave configuration'.

Voir aussi : ***NodeMonitoring*** [*NodeMonitoring*]^{API}

Log supervision

The technical supervision can be done through the log category 'ebx.d3', declared in the EBX5 main configuration file. For example:

```
ebx.log4j.category.log.d3= INFO, Console, ebxFile:d3
```

Voir aussi : [*Configuring the EBX5 logs*](#)

Guide du développeur (en anglais)

CHAPITRE 70

Notes to developers

Vocabulary evolution

EBX5 introduced a new vocabulary for users (see the [Glossaire](#)) but this new vocabulary has not impacted the API. Hence the Java classes, data services WSDL and EBX5 components are still using the old vocabulary.

The typography rules applied below are representative of the ones applied in the documentation, but not of those used in the user interface.

The following table summarizes those evolutions:

Vocabulary in EBX5	Vocabulary prior to version 5
Data set	Adaptation instance
Child data set	Child adaptation instance
Data model	Data model
Data space	Branch
Snapshot	Version
Data space or snapshot	Home
Data Workflow	Workflow instance
Workflow model	Workflow definition
Workflow publication	Workflow
Data services	Data services
Field	Attribute
Inherited field	Inherited attribute
Record	Record/occurrence
Validation rule	Constraint
Simple/advanced control	Simple/advanced constraint

Model design

CHAPITRE 71

Introduction

A data model is a structural definition of the data to be managed in the EBX5 repository. Globally, the goal is to guarantee the highest level of data consistency and to facilitate their management.

In concrete terms, the data model is a document that conforms to the XML Schema standard (W3C recommendation). The main standard features that are supported are the following:

- A rich library of well-defined basic [data types](#) (*simple types*: integer, boolean, decimal, date, time, ...);
- The ability to build more [simple types](#) (*simple types*); and [complex structures](#) (*complex types*);
- The ability to define simple lists of items ([aggregated lists](#));
- [Validation constraints](#) (*facets*): enumerations, uniqueness, minimum/maximum boundaries, ...

EBX5 also uses the extensibility features of XML Schema for other useful information, such as:

- Useful [predefined types](#) (locale, resource, html, ...);
- Definition of [tables](#) and [foreign key constraints](#) ;
- Mapping of data in EBX5 to Java beans;
- [Advanced validation constraints](#) (extended *facets*), like dynamic enumerations;
- Extensive [presentation information](#): label, description, error messages, ...

Note: EBX5 supports a subset of the W3C recommendation, some features actually being useless for Master Data.

Model Editor

The data model can be defined by using an XML Schema editor or the *Data Model Assistant* . The latter has the advantage of being integrated into EBX5 user interface and hiding the XML verbose language.

References

For an introduction to XML Schema, we recommend the [W3School](#) web site.

W3C specification documents: XML Schema [Part 0: Primer](#) , [Part 1: Structures](#) , [Part 2: Datatypes](#) .

Links between data sets and data models

Any root data set is associated with a single data model.

In order to create a root data set, you must have the rights granted. Being logged in EBX5, you have to position to a data space conten (click on the link "View or edit content"). On the left side of the screen, click on the "create..." link. Several creation modes are then available.

Pre-requisite for XML Schemas

In order to be accepted by EBX5, an XML Schema must include a global element declaration which has an attribute `osd:access="--"` .

```
<?xml version="1.0" encoding="UTF-8"?>
<!-->
<!-- {ebx.copyright.text} -->
<!-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:import namespace="urn:ebx-schemas:common_1.0"
    schemaLocation="http://schema.orchestranetworks.com/common_1.0.xsd"/>
  <xs:element name="root" osd:access="--">
    ...
  </xs:element>
</xs:schema>
```

Conventions

By convention, namespaces are always defined as below:

Prefix	Namespace
xs:	http://www.w3.org/2001/XMLSchema
osd:	urn:ebx-schemas:common_1.0
fmt:	urn:ebx-schemas:format_1.0

Schemas with reserved names

Several data model have reserved names in EBX5.

All references to other data models (attribute `schemaLocation` an import, includes or redefines) that end with one of the following strings are reserved:

- `common_1.0.xsd`
- `org_1.0.xsd`
- `coreModel_1.0.xsd`
- `session_1.0.xsd`

Those files correspond to the schemas provided for the module `ebx-root-1.0`, path `/WEB-INF/ebx/schemas`. The attribute `schemaLocation` can reference those files at this location or reference a copy, if the file name is the same. This is useful if you want to avoid the module dependency to `ebx-root-1.0`.

For security reasons, EBX5 uses an internal definition of those schemas (to avoid any modification).

CHAPITRE 72

Packaging EBX5 modules

An EBX5 module allows the packaging of a data model with its resources: included XML Schema Documents, Java classes, etc.

On a Java EE application server, a module in the EBX5 repository is equivalent to a standard Java EE web application. This provides features such as class-loading isolation, WAR or EAR packaging, web resources exposure, hot-redeployment. In addition, if your user application is a web application, it is possible to merge the EBX5 module with your application, in order to simplify the deployment.

Structure

An EBX5 module contains the following files:

1. /WEB-INF/web.xml:

This is the standard Java EE deployment descriptor. It must ensure that the EBX5 module is registered when the application server is launched (see [Registration](#) section).

2. /WEB-INF/ebx/module.xml:

This mandatory document defines the main properties and services of the module.

3. /www/:

This optional directory contains all external resources (accessible by a public URL). This directory is optional, localized and structured by resource type (html, images, jscripts, stylesheet). External resources in this directory can be referenced in data models (type `osd:resource`).

Declaration

A module is declared by means of the document /WEB-INF/ebx/module.xml. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:ebx-schemas:module_2.3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ebx-schemas:module_2.3 http://schema.orchestranetworks.com/
module_2.3.xsd">
  <name>moduleTest</name>
```

```

<locales>
  <locale isDefault="true">it</locale>
  <locale>en-US</locale>
</locales>
</module>

```

The [associated schema](#) provides a documentation on each property. Here is a summary of the main ones:

Element	Description	Required
name	Defines the unique identifier of the module on the server instance. The module name usually corresponds to the web application's name (the name of the directory).	Yes.
publicPath	If a specific path (different from the above module's name) identifies the web application in the public URLs, defines this path. This path is added to the URL of an external resource of the module, in the case of absolute URL computing. If this field is not present, the public path is the module's <code>name</code> defined above.	No.
locales	Defines the locales supported by the data model(s) of the module. This list must contain all the locales that can be found in the schemas within the module, and that we want to expose to the end user (EBX5 will not be able to display labels and messages in a language that is not declared in this list). If the element is not present, the module supports the locales of EBX5.	No.
services	Declares the UI services. For more information, see UI services section .	No.
beans	Declares the reusable Java bean components. For more information, see workflow package [package-summary] ^{API} .	No.
ajaxComponents	Declares the Ajax components. For more information, see the section "Declaration in a module" in UIAjaxComponent [UIAjaxComponent] ^{API} in the Java API.	No.

Registration

In order to be identified by EBX5, a module must be registered at runtime, when the application server is launched. For a web application, every EBX5 module must:

1. contain a Servlet whose standard method `init` invokes `ModulesRegister.registerwebApp(...)`, see Java code example at the end of this section;
2. declare this servlet in the deployment descriptor `/WEB-INF/web.xml`, in the standard way;
3. ensure that this servlet will be launched at server startup by adding the following standard element to the deployment descriptor: `<load-on-startup>1</load-on-startup>`.

Example of a Java EE deployment descriptor (`/WEB-INF/web.xml` file):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <servlet>
    <servlet-name>InitEbxServlet</servlet-name>
    <servlet-class>com.foo.RegisterServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
</web-app>
```

Notes:

- If Java classes are located in the web application (in `/WEB-INF/classes` or `/WEB-INF/lib`) and these classes are used as resources of the schemas in the module, it is mandatory that the specific servlet class also be located in the web application (the reason is that the servlet class is internally used as a hook to the application's class-loader).
- It is recommended that the servlet also implements the method `destroy()`, otherwise the "stop" of the web application will not work.

Once the method `ModulesRegister.unregisterWebApp()` has been executed, the schemas and the adaptations that depend on it become unavailable.

EBX5 supports hot-deployment and hot-redeployment operations that are implemented by Java EE application servers: deployment-start, stop-restart, stop-redeployment-restart, stop of a web application. However, these operations remain sensitive, more particularly concerning class-loading and potential complex dependencies, consequently these operations must be followed carefully.

- All modules' registrations and unregistrations are logged to 'log.kernel' category.
- If an exception occurs while loading a module, the cause of the error is written in the application server log.

Java code example of a servlet that registers/ unregisters the module to/from EBX5:

```
package com.foo;
import javax.servlet.*;
import javax.servlet.http.*;
import com.onwbp.base.repository.*;
/**
 * /
public class RegisterServlet extends HttpServlet
{
    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
        ModulesRegister.registerWebApp(this, config);
    }
    public void destroy()
    {
        ModulesRegister.unregisterWebApp(this, this.getServletConfig());
    }
}
```

Voir aussi : [Module registration](#)

CHAPITRE 73

Types

This page presents the data types supported by EBX5. For the definition of tables, read the section [Tables & Filters](#).

XML Schema built-in simple types

The table below lists all simple types defined in XML Schema and supported by EBX5:

WXS type		Java class
string	primitive datatype	java.lang.String
boolean	primitive datatype	java.lang.Boolean
decimal	primitive datatype	java.math.BigDecimal
dateTime	primitive datatype	java.util.Date
time	primitive datatype	java.util.Date
date	primitive datatype	java.util.Date
anyURI	primitive datatype	java.net.URI
Name	string restriction	java.lang.String
int	decimal restriction	java.lang.Integer

The rightmost column shows the Java class that is instantiated in for each XML Schema type in EBX5. Correspondence rules between XML Schema types and Java types are detailed in the section [Mapping to Java](#).

XML Schema named simple types

Named simple types can be defined when designing a data model for redefining an existing built-in simple type. A *named simple type* can be reused in the data model.

Restrictions:

- In term of content model, only the tag `<restriction>` is allowed in a named simple type (in particular, tags `<list>` and `<union>` are not supported), then only derivation by restriction is supported.
- Facets definition is not cumulative. That is, if an element and its named type both define the same kind of facet then the one defined in the type is overwritten by the local definition. This restriction is however not performed on programmatic facets defined by the tag `osd:constraint`. In this case, if an element and its named type both define a programmatic facet with distinct Java classes then the definition of these facets will be cumulative. Contrary to the XML Schema Specification, EBX5 is not strict regarding to the definition of a facet of the same kind in an element and its named type. That is, the value of a same kind of facet defined in an element is not checked according to the one defined in the named type. However, in the case of static enumerations defined both in an element and its type the local enumeration will be replaced by the intersection between these enumerations.
- It is forbidden to define different kind of enumerations on both an element and its named type. For instance, it is forbidden to specify a static enumeration in an element and a dynamic enumeration in its named type.
- It is forbidden to define simultaneously a pattern facet in both an element and its named type.

XML Schema complex types

Complex types can be defined when designing a data model. A *named complex type* can be reused in the data model.

Restrictions:

- In term of content model, only the tag `<sequence>` is allowed (in particular, the attribute definition is not supported)
- Type extension is not supported in the current version of EBX5

Extended simple types defined by EBX5

EBX5 provides pre-defined simple data types:

WXS type		Java class
osd:text	string restriction	java.lang.String
osd:html	string restriction	java.lang.String
osd:email	string restriction	java.lang.String
osd:password	string restriction	java.lang.String
osd:resource	anyURI restriction	internal class
osd:locale	string restriction	java.util.Locale

Those types are defined by the reserved schema *common-1.0.xsd*. They are defined as follows:

- *osd:text*

This type represents a textual information. Regarding a basic `xs:string`, its default user interface in EBX5 consists of a dedicated editor with several lines for input and display.

```
<xs:simpleType name="text">
  <xs:restriction base="xs:string" />
</xs:simpleType>
```

- *osd:html*

This represents a character string with HTML format. A "wysiwyg" editor is provided in EBX5 to edit it.

```
<xs:simpleType name="html">
  <xs:restriction base="xs:string" />
</xs:simpleType>
```

- *osd:email*

This represents an email address as specified in RFC822.

```
<xs:simpleType name="email">
  <xs:restriction base="xs:string" />
</xs:simpleType>
```

- *osd:password*

This represents an encrypted password. A specific editor is provided in EBX5 to edit it.

```
<xs:element name="password" type="osd:password" />
```

The default editor performs an encryption using the SHA-256 algorithm (this encryption function is available from a Java client by means of the method **DirectoryDefault.encryptString** [DirectoryDefault]^{API}).

It is also possible that the default editor uses a different encryption by specifying a class implementing the interface **Encryption** [Encryption]^{API}:

```
<xs:element name="password" type="osd:password">
  <xs:annotation>
    <xs:appinfo>
      <osd:uiBean class="com.orchestranetworks.ui.UIPassword">
        <encryptionClass>package.EncryptionClassName</encryptionClass>
      </osd:uiBean>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

- *osd:resource*

This represents a resource in a module for EBX5. A resource is a file of type image, HTML, CSS or JavaScript, stored into a module within EBX5. It requires the definition of the facet [FacetOResource](#) .

```
<xs:simpleType name="resource">
  <xs:restriction base="xs:anyURI" />
</xs:simpleType>
```

- *osd:locale*

This represents a geographical, political or cultural specific location. The locale type is translated into Java by the class `java.util.Locale` .

```
<xs:simpleType name="locale">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ar" osd:label="Arabic" />
    <xs:enumeration value="ar_AE" osd:label="Arabic (United Arab Emirates)" />
    <xs:enumeration value="ar_BH" osd:label="Arabic (Bahrain)" />
    <xs:enumeration value="ar_DZ" osd:label="Arabic (Algeria)" />
    <xs:enumeration value="ar_EG" osd:label="Arabic (Egypt)" />
    <xs:enumeration value="ar_IQ" osd:label="Arabic (Iraq)" />
    ...
    <xs:enumeration value="vi_VN" osd:label="Vietnamese (Vietnam)" />
    <xs:enumeration value="zh" osd:label="Chinese" />
    <xs:enumeration value="zh_CN" osd:label="Chinese (China)" />
    <xs:enumeration value="zh_HK" osd:label="Chinese (Hong Kong)" />
    <xs:enumeration value="zh_TW" osd:label="Chinese (Taiwan)" />
  </xs:restriction>
</xs:simpleType>
```

Complex types defined by EBX5

EBX5 provides pre-defined complex data types:

WXS type	Description
UDA	User Defined Attribute: this type allows any user, according to his access rights, to define a value associated with an attribute defined in a dictionary called a UDA Catalog.
UDACatalog	Catalog of User Defined Attributes: this type consists of a table in which attributes can be specified. This catalog will be used by all <code>osd:UDA</code> declared in the same data model.

- *UDA*

A User Defined Attribute (UDA) supports both `minOccurs` and `maxOccurs` parameters and `osd:UDACatalogPath` to specify the path of the matching catalog.

```
<xs:element name="firstUDA" type="osd:UDA" minOccurs="0"
  maxOccurs="unbounded" osd:UDACatalogPath="//insuranceCatalog" />
<xs:element name="secondUDA" type="osd:UDA" minOccurs="1" maxOccurs="1"
  osd:UDACatalogPath="/root/userCatalog" />
<xs:element name="thirdUDA" type="osd:UDA" minOccurs="0" maxOccurs="1"
  osd:UDACatalogPath="//userCatalog" />
```

In the manager, when working with an UDA, the editor will adapt itself to the type of the selected attribute.

- *UDACatalog*

A catalog is a table. The parameters `minOccurs` and `maxOccurs` have to be specified.

Several catalogs can be defined in one data model.

```
<xs:element name="insuranceCatalog" type="osd:UDACatalog" minOccurs="0"
  maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation xml:lang="en-US">Insurance Catalog.</xs:documentation>
    <xs:documentation xml:lang="fr-FR">Catalog assurance.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="userCatalog" type="osd:UDACatalog" minOccurs="0"
  maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation xml:lang="en-US">User catalog.</xs:documentation>
    <xs:documentation xml:lang="fr-FR">Catalogue utilisateur.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Only the following types are available for creating new attributes:

- `xs:string`

- `xs:boolean`
- `xs:decimal`
- `xs:dateTime`
- `xs:time`
- `xs:date`
- `xs:anyURI`
- `xs:Name`
- `xs:int`
- `osd:html`
- `osd:email`
- `osd:password`
- `osd:locale`
- `osd:text`

Restrictions on User Defined Attributes and Catalogs

The following are unsupported on UDA elements:

- Facets
- Functions using the `osd:function` property
- UIBean editors using the `osd:uiBean` property
- `osd:checkNullInput` property
- Historization features
- Inheritance features using the `osd:inheritance` property

As UDA catalogs are considered to be tables, the restrictions that apply to tables also exist for `UDACatalog` elements.

Aggregated lists

In XML Schema, the maximum number of times an element may appear is determined by the value of a `maxOccurs` attribute in its declaration. If this value is strictly greater than 1 or is equal to `unbounded`, then the data can have multiple occurrences. If no `osd:table` declaration is added, this case is called *aggregated lists*. In Java, it is represented as an instance of class `java.util.List`.

Important note: the addition of a `osd:table` declaration to an element with `maxOccurs > 1` is a very important consideration that must be taken into account during the design process. An aggregated list is indeed severely limited regarding the many features that are supported by tables. As a reminder, here are some of them: lookups, filters and searches; sorting, custom view and display in hierarchies; identity constraints (primary keys and uniqueness constraints); detailed permissions for creation, modification, delete and particular permissions at record level; detailed comparison and merge; and last but not least performance and memory optimizations. Hence *aggregated lists should be used only for small volumes of simple data (one or two dozen of occurrences), with no advanced requirements* . For larger volumes of data or more advanced functionalities, it is strongly advised to use `osd:table` declarations.

For more information on table declarations, read the [chapter about tables](#) .

Below is an example of an aggregated list that defines the pricing of a loan product, depending on the amount borrowed.

```
<xs:element name="pricing" minOccurs="0" maxOccurs="unbounded"
  osd:access="RW">
  <xs:annotation>
    <xs:documentation>
      <osd:label>Pricing</osd:label>
      <osd:description>Pricing grid </osd:description>
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="amount" type="xs:int">
        <xs:annotation>
          <xs:documentation>
            <osd:label>Amount borrowed</osd:label>
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="monthly" type="xs:int">
        <xs:annotation>
          <xs:documentation>
            <osd:label>Monthly payment </osd:label>
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="cost" type="xs:int">
        <xs:annotation>
          <xs:documentation>
            <osd:label>Cost</osd:label>
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Aggregated lists have a dedicated editor in EBX5. This editor allows you to add occurrences or to delete occurrences.

Including external data models

Including another data model in your current model allows you to use the reusable types that are defined in that data model. You can therefore use the inclusion of external data models to share data types between multiple XML Schema Documents.

To include another XML Schema Document in your model, thereby including the data types that it defines, specify the element `xs:include` as follows:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:include schemaLocation="./schemaToInclude.xsd"/>
  ...
</xs:schema>
```

The attribute `schemaLocation` is mandatory and must specify either an absolute path or a relative path to the XML Schema Document to include.

The inclusion of XML Schema Documents is not namespace aware, thus all included data types must belong to the same namespace. As a consequence, including XML Schema Documents that define data types of the same name is not supported.

EBX5 includes extensions with specific URNs for including embedded data models and data models packaged in modules.

To include an embedded data model in a model, specify the URN `urn:ebx:publication:aPublicationName` for the attribute `schemaLocation`, where 'aPublicationName' is the name of a publication that was created using the data model assistant by publishing an embedded data model. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:include schemaLocation="urn:ebx:publication:myPublication"/>
  ...
</xs:schema>
```

To include a data model packaged in a module, specify the URN `urn:ebx:module:aModuleName:aPathInModule` for the attribute `schemaLocation`. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:include schemaLocation="urn:ebx:module:ebx-tutorial:/WEB-INF/ebx/schema/ebx-
tutorial.xsd"/>
  ...
</xs:schema>
```

Note

If the packaged data model uses Java resources, the class loader of the module containing the data model will be used at runtime for resolving these resources.

Also, as the above URNs are specific to EBX5, they will not be able to be resolved by external tools. Thus, when using an external editor to edit any XML Schema Documents that contain these URNs, the editor may report errors. To workaround this issue, you can manually substitute the physical locations of the targeted XML Schema documents for the URNs before beginning to edit the XML Schema document using an external editor.

CHAPITRE 74

Tables, filters and selection nodes

EBX5 supports the features of relational databases tables with a high volume of records and primary key identification.

Tables bring many benefits over [aggregated lists](#) . Beyond pure relational aspects, here are some features that tables provide:

- filters and searches;
- sorting, custom views and custom hierarchies;
- identity constraints: primary keys, [foreign keys](#) and [uniqueness constraints](#) ;
- detailed permissions for creation, modification, delete;
- dynamic and contextual permissions at individual record level;
- detailed comparison and merge;
- inheritance capacity at record level (see [data set inheritance](#));
- performance and memory optimizations.

Table definition

An element declaration with *maxOccurs* > 1 is declared as a table by adding the following annotation:

```
<xs:annotation>
  <xs:appinfo>
    <osd:table>
      <primaryKeys>/pathToField1 /pathToField...n</primaryKeys>
    </osd:table>
  </xs:appinfo>
```

</xs:annotation>

Element	Description	Required
primaryKeys	<p>Specifies the primary key fields of the table. Each element in the primary key is denoted by its absolute XPath notation that starts just under the table root element. If there are several elements in the primary key, the list is white-space delimited.</p> <p>Note: A specific whitespaces management has been defined for primary keys of type <code>xs:string</code>.</p>	Yes.
defaultLabel	<p>Defines the end-user display of records. Multiple localized versions can be specified:</p> <ul style="list-style-type: none"> Default expression and localized expressions are specified using the <code>defaultLabel</code> element, for example: <pre> <defaultLabel>Product: \${./ProductID}</defaultLabel> <defaultLabel xml:lang="fr-FR">Produit : \${./ProductID}</defaultLabel> <defaultLabel xml:lang="en-US">Product : \${./ProductID}</defaultLabel> </pre> JavaBean that implements the interface UILabelRenderer [UILabelRenderer]^{API} and/or the interface UILabelRendererForHierarchy [UILabelRendererForHierarchy]^{API}. The JavaBean is specified by means of the attribute <code>osd:class</code>, for example: <pre> <defaultLabel osd:class="com.wombat.MyLabel"></defaultLabel> </pre> <p>Note: The priority of the tags at display is the following:</p> <ol style="list-style-type: none"> 1. <code>defaultLabel</code> tags with a JavaBean (but it is not allowed to define several renderers of the same type); 2. <code>defaultLabel</code> tags with a localized expression; 3. <code>defaultLabel</code> tag with a default expression. 	No.
index	<p>Property used to index some fields; indexing speeds up table access for requests matching these fields. (see performances).</p> <p>Notes:</p> <ul style="list-style-type: none"> It is possible to define several indexes on a table. It is not allowed to define two indexes with the same name. It is not allowed to declare two indexes containing the same exact fields. An indexed field must be terminal. An indexed field cannot be a list or under a list. A field declared as an <i>inherited field</i> cannot be indexed. A field declared as a function cannot be indexed. 	No.

Element	Description	Required
	<p>For performance purposes, the following nodes are automatically indexed:</p> <ul style="list-style-type: none"> • Primary keys nodes (see primary keys). • Nodes defining a foreign key constraint (see foreign key constraint). • Nodes declared as being unique (see uniqueness constraint). • Auto-incremented nodes (see auto-incremented values). 	
mayCreateRoot	expression that specifies when a <i>root</i> record may be created (see Definition modes). The expression must follow the syntax below (see property <code>mayDelete</code>).	No, default is "always".
mayCreateOverwriting	expression that specifies when an <i>overwriting</i> record may be created (see Definition modes). The expression must follow the syntax below (see property <code>mayDelete</code>).	No, default is "always".
mayCreateOcculting	expression that specifies when an <i>occulting</i> record may be created (see Definition modes). The expression must follow the syntax below (see property <code>mayDelete</code>).	No, default is "always".
mayDuplicate	expression that specifies when a record may be <i>duplicated</i> . The expression must follow the syntax below (see property <code>mayDelete</code>).	No, default is "always".
mayDelete	<p>expression that specifies when a record may be <i>deleted</i>. The expression must follow the syntax:</p> <p>The <code>may...</code> expression specifies when the action is possible, however the user access rights may restrict this possibility independently. The expressions have the following syntax:</p> <pre> expression ::= always never <condition>* condition ::= [root:yes root:no] "always": the operation is "always" possible (but user rights may restrict this). "never": the operation is never possible. "root:yes": the operation is possible if the record is in a root instance. "root:no": the operation is not possible if the record is in a root instance. If the record does not verify any condition, then default is taken. </pre>	No, default is "always".

Below is an example of a product catalog as a table:

```

<xs:element name="product" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>
      <osd:label>Product Table </osd:label>
    
```

```

    <osd:description>List of products in Catalog </osd:description>
  </xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:annotation>
  <xs:appinfo>
    <osd:table>
      <primaryKeys>/productRange /productCode</primaryKeys>
      <index name="indexProductCode">/productCode</index>
    </osd:table>
  </xs:appinfo>
</xs:annotation>
<xs:sequence>
  <xs:element name="productRange" type="xs:string"/><!-- key -->
  <xs:element name="productCode" type="xs:string"/><!-- key -->
  <xs:element name="productLabel" type="xs:string"/>
  <xs:element name="productDescription" type="xs:string"/>
  <xs:element name="productWeight" type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>

```

Voir aussi :

- [Foreign keys](#)
- [Actions principales](#)
- [Vue tabulaire simple](#)
- [Hiérarchies](#)
- [History](#)

Specifying filters on a table

By default, tables present all records in EBX5. However, the user has the ability to add filters and display only the records corresponding to given criteria.

In order to do that, you must follow the steps below:

- Define the filter in the data model (see example below),
- Implement the class in charge of the filtering, that extends `UITableFilter` ^{API}
[UITableFilter]

As a result, a filter option will appear in the user screen, giving the opportunity to refine your search.

```

<xs:element name="product" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:appinfo>
      <osd:table>
        ...
      </osd:table>
      <osd:uiFilter class="com.mycompany.product.filters.productCodeFilter">
        <label>Product code filter</label>
        <label xml:lang="en-US">Product code filter</label>
        <label xml:lang="fr-FR">Filtre code produit</label>
      </osd:uiFilter>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```



```

<osd:uiFilter class="com.mycompany.product.filters.productLabelFilter">
  <label>Product label filter</label>
  <label xml:lang="en-US">Product label filter</label>
  <label xml:lang="fr-FR">Filtre nom produit</label>
</osd:uiFilter>
</xs:appinfo>
</xs:annotation>
...
</xs:element>

```

Element	Description	Required
label	Specifies a label for the filter. This label can be localized using the <code>xml:lang</code> attribute.	No.
class	Attribute declaring the Java class which extends <code>UITableFilter</code> . Full qualified Java name must be used.	Yes.

Selection nodes

An element declaration may define a dynamic and contextual XPath selection. In this case, the user interface provides a link so that the user can navigate to the pre-filtered table that corresponds to the selection.

A selection node is useful for creating an *association between two entities* in the user interface and also for validation purposes. For example, the tutorial library data model specifies that a book is written by an author (defined explicitly by a foreign key in the 'Book' complex type definition). The relation in the opposite direction, that an author has written certain books, cannot be easily expressed in XML Schema, unless the 'Author' complex type definition includes the following selection node:

```

<xs:element name="linkToAuthorTitles" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:select>
        <xpath>/root/Titles[au_id =${../au_id}]</xpath>
        <minOccurs>1</minOccurs>
        <minOccursValidationMessage>
          <severity>error</severity>
          <message>A default validation message.</message>
          <message xml:lang="en-US">A validation message in English.</message>
          <message xml:lang="fr-FR">Un message de validation en français.</message>
        </minOccursValidationMessage>
        <maxOccurs>10</maxOccurs>
        <maxOccursValidationMessage>
          <severity>error</severity>
          <message>A default validation message.</message>
          <message xml:lang="en-US">A validation message in English.</message>
          <message xml:lang="fr-FR">Un message de validation en français.</message>
        </maxOccursValidationMessage>
      </osd:select>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

The element `minOccurs` specifies that, for being valid, an author must have written at least one book.

The element `maxOccurs` specifies that, for being valid, an author must have written at most ten books.

Note

The "official" cardinality constraints (`minOccurs="0"` `maxOccurs="0"`) are required because from an external XML point of view (that is, an instance of XML Schema), the corresponding node is absent. In other words, a selection node is a "virtual" element regarding XML and XML Schema.

It is also possible to specify an additional constraint on the relation. In the following example, each relation between an author and a book is valid, if the date of the publication of the book is anterior to the death of the current author:

```
<osd:select>
  <xpath>/root/Titles[au_id =${../au_id}]/</xpath>
  <constraintPredicate>date-less-than(datePub, ${../death})</constraintPredicate>
  <constraintPredicateValidationMessage>
    <severity>error</severity>
    <message>A default validation message.</message>
    <message xml:lang="en-US">A validation message in English.</message>
    <message xml:lang="fr-FR">Un message de validation en français.</message>
  </constraintPredicateValidationMessage>
```

</osd:select>

Element	Description	Required
xpath	<p>Specifies the selection to be performed, relative to the current node.</p> <p>Examples: <code>/root/Titles[au_id = \${../au_id}]</code> or <code>//Titles[au_id = \${../au_id}]</code> or <code>../Titles[au_id = \${../au_id}]</code>.</p> <p>The path up to the predicate (for example <code>../Titles</code>) specifies the target table to be filtered. This part of the path is resolved relative to the current table root.</p> <p>If the selection depends on the local state, the XPath expression predicate must include references to the node on which it depends, with this notation: <code>\${ <relative-path> }</code> where <code>relative-path</code> is a path that locates the element relative to the node that holds the selection link.</p> <p>For XPath syntax, see EBX5 XPath supported syntax.</p>	Yes.
container	Reference of the instance that contains the target table.	No, default is current instance.
minOccurs	Specifies an additional validation constraint: the selection must be at least of the specified size.	No, default is 0.
minOccursValidationMessage	<p>Specifies additional localized messages that will be displayed if the selection does not comply with the <code>minOccurs</code> constraint.</p> <p>A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'.</p> <p>Each localized message variant is defined in a nested <code>message</code> element with its locale in an <code>xml:lang</code> attribute. To specify a default, non-localized message, define a <code>message</code> element with no <code>xml:lang</code> attribute.</p>	No.
maxOccurs	Specifies an additional validation constraint: the selection must be at most of the specified size.	No, by default the maximum is not restricted.
maxOccursValidationMessage	<p>Specifies an additional localized message that will be displayed if the selection does not comply with the <code>maxOccurs</code> constraint.</p> <p>A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'.</p> <p>Each localized message variant is defined in a nested <code>message</code> element with its locale in an <code>xml:lang</code> attribute. To specify a default, non-localized message, define a <code>message</code> element with no <code>xml:lang</code> attribute.</p>	No.
constraintPredicate	Specifies an additional validation constraint: each relation of the selection must satisfy the specified	No.

Element	Description	Required
	XPath predicate. The notation <code>\${<relative-path>}</code> has the same meaning as for the <code>xpath</code> element (see above).	
<code>constraintPredicateValidationMessage</code>	<p>Specifies a localized message to display when the constraint predicate is not satisfied.</p> <p>A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'.</p> <p>Each localized message variant is defined in a nested <code>message</code> element with its locale in an <code>xml:lang</code> attribute. To specify a default, non-localized message, define a <code>message</code> element with no <code>xml:lang</code> attribute.</p>	No.

Important: When creating a data set, you can create a data set that defines a selection node to a container that does not exist in the repository. However, the content of this data set will not be available after its creation. After the creation of the container, a data model refresh is required to make the data set available. When creating a data set that refers to a container that does not yet exist, the following limitations apply:

- Triggers defined at the data set level are not executed,
- Default values on fields outside tables are not initialized,
- During an archive import, it is not possible to create a data set that refers to a container that does not exist.

CHAPITRE 75

Constraints, triggers and functions

Facets allow you to define constraints on the data in your data models. EBX5 supports XML Schema facets and provides extended and programmatic facets for advanced data controls.

XML Schema supported facets

The tables below show the facets that are supported by different data types.

Key:

- **X** - Supported
- **1** - The `whiteSpace` facet can be defined, but is not interpreted by EBX5
- **2** - In XML Schema, boundary facets are not allowed on the type `string`. Nevertheless, EBX5 allows those facets as extensions.
- **3** - The `osd:resource` type only supports the facet `FacetOResource`, which is required. See [Extended Facets](#).

	length	minLength	max Length	pattern	enumeration	white Space
string	X	X	X	X	X	1
boolean				X		1
decimal				X	X	1
dateTime				X	X	1
time				X	X	1
date				X	X	1
anyURI	X	X	X	X	X	1
Name	X	X	X	X	X	1
integer				X	X	1
osd:resource 3						

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
string			2	2	2	2
boolean						
decimal	X	X	X	X	X	X
dateTime			X	X	X	X
time			X	X	X	X
date			X	X	X	X
anyURI						
Name			2	2	2	2

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
integer	X	X	X	X	X	X
osd:resource 3						

Example:

```
<xs:element name="loanRate">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="4.5" />
      <xs:maxExclusive value="17.5" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Uniqueness constraint

It is possible to define a uniqueness constraint, by using the standard XML Schema element [xs:unique](#). This constraint indicates that a value or a set of values has to be unique inside a table.

In the example below, a uniqueness constraint is defined on the *publisher* table, for the target field *name* (this means that two records of the *publisher* table cannot have the same name):

```
<xs:element name="publisher">
  ...
  <xs:complexType>
    <xs:sequence>
      ...
      <xs:element name="name" type="xs:string" />
      ...
    </xs:sequence>
  </xs:complexType>
  <xs:unique name="uniqueName">
    <xs:annotation>
      <xs:appinfo>
        <osd:validationMessage>
          <severity>error</severity>
          <message>Name must be unique in table.</message>
          <message xml:lang="en-US">Name must be unique in table.</message>
          <message xml:lang="fr-FR">Le nom doit être unique dans la table.</message>
        </osd:validationMessage>
      </xs:appinfo>
    </xs:annotation>
    <xs:selector xpath="." />
    <xs:field xpath="name" />
  </xs:unique>
</xs:element>
```

A uniqueness constraint has to be defined inside a table and has the following properties:

Property	Description	Mandatory
name attribute	Identifies the constraint in the data model.	Yes
xs:selector element	Indicates the table where the uniqueness applies, using a restricted XPath expression ('..' is forbidden). It can also indicate an element inside the table (without changing the meaning of the constraint).	Yes
xs:field element	Indicates the field in the context whose values must be unique, using a restricted XPath expression. It is possible to indicate that a set of values must be unique by defining multiple xs:field elements.	Yes

Additional localized validation messages can be defined using the element `osd:validationMessage` under the elements `annotation/appinfo`. If no custom validation messages are defined then a built-in validation message will be used.

Limitations:

1. The target of the `xs:field` element has to be in a table.
2. Uniqueness constraint cannot apply if the field is inside an aggregated list.
3. Uniqueness constraint cannot apply if the field is computed.

Extended facets

EBX5 provides additional constraints that are not specified in XML Schema, but that are useful for managing master data.

In order to guarantee conformance to XML Schema, those Extended Facets are defined under the element `annotation/appinfo/otherFacets`.

Foreign keys

A reference to a [table](#) is defined using the extended facet `osd:tableRef`.

The node holding the `osd:tableRef` declaration must be of type `xs:string`. At instantiation, any value of the node identifies a record in the target table by mean of its **primary key syntax**

[PrimaryKey]^{API}. This extended facet is also interpreted as an enumeration whose values refer to the records in the target table.

Element	Description	Required
tablePath	XPath expression that specifies the target table.	Yes.
container	Reference of the data set that contains the target table.	Only if element 'branch' (i.e data space) is defined. Otherwise, default is current data set.
branch	Reference of the data space (former 'branch') that contains the data set container.	No, default is current data space or snapshot.
display	<p>Custom display for presenting the selected foreign key in the current record and the sorted list of possible keys. Two variants can be specified, either pattern-based expressions, or a JavaBean if the needs are very specific:</p> <ul style="list-style-type: none"> Default and localized expressions. They are specified by means of the <code>display</code> and <code>pattern</code> elements, for example: <pre> <display> <pattern>Product : \${./ProductID}</pattern> <pattern xml:lang="fr-FR">Produit : \${./ProductID}</pattern> <pattern xml:lang="en-US">Product : \${./ProductID}</pattern> </display> </pre> A JavaBean that implements the interface TableRefDisplay [TableRefDisplay]^{API}. It is specified by means of the attribute <code>osd:class</code>; for example: <pre> <display osd:class="com.wombat.MyLabel"></display> </pre> 	No, if the <code>display</code> property is absent, then the table's record rendering is used.
filter	<p>Specifies an additional constraint that filters the records of the target table. Two types of filters are available:</p> <ul style="list-style-type: none"> An XPath filter is an XPath predicate in the target table context. It is specified by means of the <code>predicate</code> element; for example: <pre> <filter><predicate>type = \${../refType}</predicate></filter> </pre> <p>A localized validation message can be specified using the element <code>validationMessage</code>, which will be displayed to the end user at validation time if a record is not accepted by the filter.</p> <p>A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'.</p> <p>Each localized message variant is defined in a nested <code>message</code> element with its locale in an <code>xml:lang</code> attribute.</p> 	No.

Element	Description	Required
	<p>To specify a default message for unsupported locales, define a <code>message</code> element with no <code>xml:lang</code> attribute.</p> <ul style="list-style-type: none"> A programmatic filter is a JavaBean that implements the interface TableRefFilter <code>[TableRefFilter]^{API}</code>. It is specified by means of the attribute <code>osd:class</code>; for example: <pre><filter osd:class="com.wombat.MyFilter"></filter></pre> <p>Additional validation messages can be specified during the setup of the programmatic filter using the dedicated methods of the interface TableRefFilterContext <code>[TableRefFilterContext]^{API}</code>.</p> <p>Notes:</p> <ul style="list-style-type: none"> Attribute <code>osd:class</code> and the property <code>predicate</code> cannot be set simultaneously. 	
<code>validationMessage</code>	<p>Specifies localized validation messages for the <code>osd:tableRef</code>.</p> <p>A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'.</p> <p>Each localized message variant is defined in a nested <code>message</code> element with its locale in an <code>xml:lang</code> attribute. To specify a default message for unsupported locales, define a <code>message</code> element with no <code>xml:lang</code> attribute.</p>	No.

Important: When creating a data set, you can create a data set that defines a foreign key to a container that does not exist in the repository. However, the content of this data set will not be available after its creation. After the creation of the container, a data model refresh is required to make the data set available. When creating a data set that refers to a container that does not yet exist, the following limitations apply:

- Triggers defined at the data set level are not executed,
- Default values on fields outside tables are not initialized,
- During an archive import, it is not possible to create a data set that refers to a container that does not exist.

The example below specifies a reference (foreign key) to a record of *catalog* element.

```
<xs:annotation>
  <xs:appinfo>
    <osd:otherFacets>
      <osd:tableRef>
        <tablePath>../catalog</tablePath>
        <display>
          <pattern xml:lang="en-US">Product: ${../ProductID}</pattern>
          <pattern xml:lang="fr-FR">Produit : ${../ProductID}</pattern>
        </display>
        <filter>
          <predicate>type = ${../refType} </predicate>
          <validationMessage>
```

```

<severity>error</severity>
<message>A default error message</message>
<message xml:lang="en-US">A localized error message</message>
<message xml:lang="fr-FR">Un message d'erreur localisé</message>
</validationMessage>
</filter>
</osd:tableRef>
</osd:otherFacets>
</xs:appinfo>
</xs:annotation>

```

Voir aussi :

- [Table definition](#)
- **Primary key syntax** [*PrimaryKey*]^{API}
- [Extraction of foreign keys \(XPath predicate syntax\)](#)

Dynamic constraints

Those facets retain the XML Schema semantics, but the value attribute is replaced by a path attribute that allows fetching the value from another element in the adaptation:

- length
- minLength
- maxLength
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

By means of these facets, the data model may be modified dynamically, in the adaptations.

Example:

```

<xs:element name="amount">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:minInclusive path="/domain/Loan/Pricing/AmountMini/amount" />
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>

```

In this example, the boundary of the facet minInclusive is not statically defined. The value of the boundary comes from the node */domain/Loan/Pricing/AmountMini/amount*

FacetOResource constraint

This facet must be defined for each resource type. It has the following attributes:

- `moduleName`: indicates with an alias which EBX5 module contains the resource. If it is the module itself that contains the resource, the alias must be preceded by "wbp". Otherwise, the alias must be one of the values defined in the element `<dependencies>` of the file 'module.xml';
- `resourceType`: represents the resource type that is one of the following values: "ext-images", "ext-jscripts", "ext-stylesheets", "ext-html";
- `relativePath`: represents the local directory where the resource is located, just under the directory named with the value `resourceType`. In the example below, it is the directory `www/common/images/`. Hence, for this example, the resource is located at `www/common/images/promotion/` where the `www/` directory is at the same level as the `WEB-INF/` directory.

Furthermore, if a resource is defined in a localized directory (`www/en/` for instance), it will be considered only if another resource of the same name is defined in the directory `www/common/`.

This facet has the same behavior as an enumeration facet: the values are collected by recursively listing all the files in 'local path' from the specified directory 'resource type' in the specified 'module'.

Example:

```
<xs:element name="promotion" type="osd:resource">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:FacetOResource osd:moduleName="wbp"
          osd:resourceType="ext-images" osd:relativePath="promotion/" />
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

An overview of the standard directory structure of a module (Java EE web application) for EBX5 is detailed in the section [Modules - Structure](#).

excludeValue constraint

This facet verifies that an instance is different from a specified value.

Example:

```
<xs:element name="roleName">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:excludeValue value="">
          <osd:validationMessage>
            <severity>error</severity>
            <message>Please select address role(s).</message>
          </osd:validationMessage>
        </osd:excludeValue>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

```

    </osd:otherFacets>
  </xs:appinfo>
</xs:annotation>
<xs:simpleType type="xs:string" />
</xs:element>

```

In this example, the empty string is excluded from the possible values.

excludeSegment constraint

This facet verifies that an instance is not included in a segment of values. Boundaries are excluded.

Example:

```

<xs:element name="zipCode">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:excludeSegment minValue="20000" maxValue="20999">
          <osd:validationMessage>
            <severity>error</severity>
            <message>Postal code not valid.</message>
          </osd:validationMessage>
        </osd:excludeSegment>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  <xs:simpleType type="xs:string" />
</xs:element>

```

In this example, values between 20000 and 20999 are excluded.

Enumeration facet filled by another node

By default, an enumeration facet is described statically in XML Schema.

The content of an enumeration facet can be filled dynamically by another node of the data model.

Filling by a list

With the following information, we indicate that the content of an enumeration facet comes from the node *CountryList*.

```

<xs:annotation>
  <xs:appinfo>
    <osd:otherFacets>
      <osd:enumeration osd:path="../../CountryList" />
    </osd:otherFacets>
  </xs:appinfo>
</xs:annotation>

```

The node *CountryList* :

- Must be of the same type as the node that has the enumeration facet
- Must be an aggregated list (maxOccurs > 1).

Example:

```
<xs:element name="FacetEnumBasedOnList">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CountryList" maxOccurs="unbounded">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="DE" osd:label="Germany" />
            <xs:enumeration value="AT" osd:label="Austria" />
            <xs:enumeration value="BE" osd:label="Belgium" />
            <xs:enumeration value="JP" osd:label="Japan" />
            <xs:enumeration value="KR" osd:label="Korea" />
            <xs:enumeration value="CN" osd:label="China" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="CountryChoice" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <osd:otherFacets>
              <osd:enumeration osd:path="../../CountryList" />
            </osd:otherFacets>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Programmatic facets

A programmatic constraint can be added on any XML element declaration that refers to a simple type.

In order to guarantee the conformance to XML Schema, those constraints are specified under the element annotation/appinfo/otherFacets.

Programmatic constraints

A programmatic constraint is defined by a Java class that implements the interface **Constraint** [Constraint]^{API}.

Additional parameters can be defined. Consequently, the implemented Java class must conform to the JavaBean protocol. In the example below, the Java class must define the methods: *getParam1()*, *setParam1(String)*, ..., *getParamX()*, *setParamX(String)*, etc.

Example:

```
<xs:element name="amount">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:constraint class="com.foo.CheckAmount">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraint>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

Voir aussi : **JavaBeans Specifications** [*package-summary*]^{API}

Programmatic enumeration constraints

An enumeration constraint adds to a basic programmatic constraint the definition of an ordered list of values. Such a constraint is defined by a Java class that implements the interface **ConstraintEnumeration** [*ConstraintEnumeration*]^{API}. This facet allows the selection of a value from a list.

Example:

```
<xs:element name="amount">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:constraintEnumeration class="com.foo.CheckAmountInEnumeration">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraintEnumeration>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

Constraint on null value

Sometimes, a value is mandatory only if some conditions are verified (for example if another field has some specific value). In this case, the standard XML Schema attribute `minOccurs` will not help because it is static.

For checking if a value is mandatory or not, according to its context:

1. a programmatic constraint must be defined by a Java class (see above);
2. this class must also implement the interface **ConstraintOnNull** [`ConstraintOnNull`]^{API};
3. XML Schema cardinality attributes must specify that the element is optional (`minOccurs="0"` and `maxOccurs="1"`).

Note

By default, constraint on null value is not checked on user input. In order to enable this check, the ['checkNullInput' property](#) must be set and if the element is terminal, the adaptation instance must also be activated.

Example:

```
<xs:element name="amount" minOccurs="0" maxOccurs="1">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:constraint class="com.foo.CheckIfNull">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraint>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

Constraints on table

A constraint on table is defined by a Java class that implements the interface **ConstraintOnTable** [`ConstraintOnTable`]^{API}.

A constraint on table can only be defined on a table node.

Additional parameters can be defined. Consequently, the implemented Java class must conform to the JavaBean protocol. In the example below, the Java class must define the methods: **getParam1()**, **setParam1(String)**, ..., **getParamX()**, **setParamX(String)**, etc.

Example:

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:appinfo>
      <osd:table>
        <primaryKeys>/key</primaryKeys>
      </osd:table>
      <osd:otherFacets>
        <osd:constraint class="com.foo.checkTable">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraint>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```



```

    </osd:constraint>
  </osd:otherFacets>
</xs:appinfo>
</xs:annotation>
</xs:element>

```

Important: Regarding the validation process, constraints on tables are only checked when calling a validation report of an adaptation instance/table. This means, for performance purposes, these constraints are not checked when updates (such as record insertion, deletion or modification) on tables occur. However the internal incremental validation framework will optimize the validation cost of these constraints if dependencies are defined. For more information, see [performance](#) section.

Voir aussi : **JavaBeans Specifications** [*package-summary*]^{API}

Triggers and functions

Computed values

By default, data is read and persisted in the XML repository. Nevertheless, they may be the result of a specific computation and/or access an external database (a RDBMS, a central system...).

EBX5 allows you to take into account other data in the current adaptation context.

This is made possible by defining *functions*.

A function is specified in the data model by using the `osd:function` element (see example below).

- The value of *class* attribute must be the qualified name of a Java class which implements the Java interface **ValueFunction** [*ValueFunction*]^{API}.
- Additional parameters may be specified at the data model level, in this case the JavaBeans convention is applied.

Example:

```

<xs:element name="computedValue">
  <xs:annotation>
    <xs:appinfo>
      <osd:function class="com.foo.ComputeValue">
        <param1>...</param1>
        <param...n>...</param...n>
      </osd:function>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>

```

Triggers

Instances or table records can be associated with methods that are automatically executed when some operations are performed: creation, update, delete, etc.

In the data model, these triggers must be declared under the `annotation/appinfo` element using the `osd:trigger` tag.

For adaptation instances triggers, we have to define, inside the `osd:trigger` tag, a Java class that extends the abstract class **InstanceTrigger** [InstanceTrigger]^{API}.

Note that in the case of adaptation instance triggers, it is advised to define `annotation/appinfo/osd:trigger` tags just under the root element of the considered data model.

Example:

```
<xs:element name="root" osd:access="--">
  ...
  <xs:annotation>
    <xs:appinfo>
      <osd:trigger class="com.foo.MyInstanceTrigger">
        <param1>...</param1>
        <param...n>...</param...n>
      </osd:trigger>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

For the definition of tables records triggers, we have to define, inside the `osd:trigger` tag, a Java class that extends the abstract class **TableTrigger** [TableTrigger]^{API}.

In the case of table records triggers, it is advised to define the `annotation/appinfo/osd:trigger` tags just under the element describing the considered table or table type.

Examples:

On a table element:

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:appinfo>
      <osd:table>
        <primaryKeys>/key</primaryKeys>
      </osd:table>
      <osd:trigger class="com.foo.MyTableTrigger" />
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

or on a table type element:

```
<xs:complexType name="MyTableType">
  ...
  <xs:annotation>
    <xs:appinfo>
      <osd:trigger class="com.foo.MyTableTrigger">
        <param1>...</param1>
        <param...n>...</param...n>
      </osd:trigger>
    </xs:appinfo>
  </xs:annotation>
</xs:complexType>
```

```

    </xs:appinfo>
  </xs:annotation>
  ...
</xs:complexType>

```

Note that additional parameters can be defined. Consequently, the Java class implemented must conform to the JavaBean protocol. In the example above, the Java class must define the methods: *getParam1()*, *setParam1(String)*, ..., *getParamX()*, *setParamX(String)*, etc.

Auto-incremented values

It is possible to define auto-incremented values. Auto-incremented values are only allowed inside a table, and they must be of type `xs:int` or `xs:integer`.

An auto-increment is specified in the data model by using the `osd:autoIncrement` element under the `annotation/appinfo` element tag.

Example:

```

<xs:element name="autoIncrementedValue" type="xs:int">
  <xs:annotation>
    <xs:appinfo>
      <osd:autoIncrement />
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

There are two optional elements `start` and `step`.

Example:

```

<xs:element name="autoIncrementedValue" type="xs:int">
  <xs:annotation>
    <xs:appinfo>
      <osd:autoIncrement>
        <start>100</start>
        <step>5</step>
      </osd:autoIncrement>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

The `start` attribute specifies the first value for this auto-increment. If this attribute is not specified, then the value 1 is set by default.

The `step` attribute specifies the step for the next value to be generated by the auto-increment. If this attribute is not specified, then the value 1 is set by default.

A field specifying an `osd:autoIncrement` has the following behavior:

- The computation and allocation of the field value are performed whenever a new record is inserted and the field value is still undefined.
- No allocation is performed if a programmatic insertion already specifies a non `null` value. For example, if an archive import or an XML import specifies the value, then it is preserved. Note also that, consequently, the allocation is not performed for a record insertion in occulting or overwriting modes.
- A newly allocated value is, whenever possible, unique in the scope of the repository. More precisely, the uniqueness of the allocation spans over all the data sets of the data model, and it also spans over all the data spaces. The latter case allows the merging of a data space to its parent with a reasonable guarantee that there will not be a conflict if the `osd:autoIncrement` is part of the records' primary key. This principle has a very specific limitation: when a massive update transaction specifying values is performed concurrently to a transaction allocating a value on the same field, it is possible that the latter transaction will allocate a value that will be set in the first transaction (there is no locking between several data spaces).

Internally, the auto-increment value is stored in the 'Auto-increments' table of the repository (in the user interface, it can be accessed from the Administration pane). This field is automatically updated, so that it defines the greatest value ever set on the associated `osd:autoIncrement` field, in any instance and any data space in the repository. This value is computed, taking into account the max value found in the table being updated.

In some particular case, for example when multiple environments have to be managed (development, test, production), each having different auto-increment ranges, it may be required to avoid this "max value" check. This particular behavior can be achieved, thanks to the `disableMaxTableCheck` property:

- locally, by setting a parameter element in auto-increment declaration:
`<disableMaxTableCheck>true</disableMaxTableCheck>` ,
- for the whole data model, by setting `<osd:disableMaxTableCheck="true">` in `xs:appinfo` of data model declaration,
- or globally, by setting property `ebx.autoIncrement.disableMaxTableCheck=true` in main configuration file (`ebx.properties`).

Note that it is generally not recommended to enable this property, as this could generate conflicts in the auto-increment values.

Note: when this option is enabled globally, it becomes possible to create records in the table of auto-increments by import XML or CSV for example. If this option is not selected, the creation of records in the table of auto-increments is prohibited to maintain the integrity of the repository.

User input control

Check null input

According to EBX5 default validation policy, in order to allow a temporary incomplete input, the control whether an element is defined when it is mandatory is not performed on user input, but on adaptation validation only. If the 'mandatory feature' must be checked immediately on user input, the element must additionally specify: `osd:checkNullInput="true"`.

Note: A value is mandatory if the data model specifies that the element is mandatory, either statically (`minOccurs="1"`) or dynamically (by means of a Constraint on null). For terminal elements mandatory values are checked only for an activated adaptation instance. For non-terminal elements the adaptation instance does not need to be activated.

Example:

```
<xs:element name="amount" osd:checkNullInput="true" minOccurs="1">
  ...
</xs:element>
```

Voir aussi :

- [constraint on null](#)
- [Whitespace management](#)
- [Empty string management](#)

EBX5 whitespace management for data types

According to XML Schema (cf. <http://www.w3.org/TR/xmlschema-2/#rf-whiteSpace>), whitespaces handling must follow one of *preserve*, *replace* or *collapse* procedures:

- *preserve*. No normalization is done, the value is not changed
- *replace*. All records of #x9 (tab), #xA (line feed) and #xD (carriage return) are replaced with #x20 (space)
- *collapse*. After the processing implied by replace, contiguous sequences of #x20's are collapsed to a single #x20, and leading and trailing #x20's are removed

EBX5 general management

EBX5 complies with XML Schema recommendation:

- For nodes of type `xs:string` (whether primary key element or not), whitespaces are always preserved and an empty string is never converted to `null`.
- For other nodes (non `xs:string` type), whitespaces are always collapsed and an empty string is converted to `null`.

Exceptions: for nodes of type `osd:html` or `osd:password`, whitespaces are always preserved and an empty string is converted to `null`. For nodes of type `xs:string` defining property `osd:checkNullInput="true"` an empty string is interpreted as `null` at user input in EBX5.

Note

Values that are composed of *only whitespaces* are considered to be empty.

Specific case for handling whitespaces on a primary key of type string

For primary key columns of type `xs:string`, a default EBX5 constraint is defined. This constraint forbids empty strings and non collapsed whitespaces values at validation level.

However, if the primary key node specifies its own `xs:pattern` facet, this last one overrides the default EBX5 constraint. For example, the specific pattern `.*` would accept any strings (note that we do not recommend this).

The default constraint allows the handling of some ambiguities. For example, it would be difficult for a user to distinguish between the following strings: "12 34" and "12 34". For usual values, it does not create conflicts but for primary keys, it could create errors.

Note: EBX5 provides a complete [Tables specification](#), that allows the organization and management of your data in EBX5 as in relational databases.

Empty string management

Default conversion

For nodes of type `xs:string` at user input no distinction is made between an empty string and a null value. That is, an empty string value is automatically converted to `null` at user input.

Distinction between empty string and null value

There are some cases where the distinction is still made between an empty string and a null value. The distinction is still performed in the following cases:

- A primary key defines a pattern allowing empty strings.
- An element defines a foreign key constraint and the target table has a single primary key defining a pattern allowing empty strings.
- An element defines a static enumeration containing an empty string.
- An element defines a dynamic enumeration to another one defining these specific distinction cases.

If the distinction is still made between an empty string and a null value this will imply the following behaviors:

- An empty string will not be converted to `null` at user input,
- Input field on nodes of type `xs:string` will display an additional button for setting to `null` the value of the node,
- At validation time an empty string will be considered as a compliant value regarding to a `minOccurs="1"` property.

Note: in relational mode, the Oracle database does not support the distinction between empty strings and null values, then these specific cases are not supported.

Voir aussi : [*Relational mode*](#)

CHAPITRE 76

Labels and messages

EBX5 allows you to enrich your data models with labels and error messages displayed in the interface.

Label and description

A label and a description can be added to each node in an adaptation model.

In EBX5, each adaptation node is displayed with its label . If no label is defined, the name of the element is used instead.

Two notations are available:

- full notation: label and description are defined respectively by the elements `<osd:label>` and `<osd:description>`.
- simple notation: the label is extracted from the text content up to the first dot (' . ') and/or this extraction is not longer than 60 characters ; the description is the rest of the text.

The description may also have an hyperlink, either a standard HTML `href` to an external document, or a link to another node of the adaptation within EBX5.

- `href` notation, and more generally any html notation must be escaped.
- EBX5 link notation is not escaped and it must specify the path of the target, for example:

```
<osd:link path="../../../misc1">link to another node in the adaptation</osd:link>
```

Example:

```
<xs:element name="misc1" type="xs:string">
  <xs:annotation>
    <xs:documentation>
      Miscellaneous 1. This is the description of miscellaneous element #1.
      Click <a href="http://www.orchestranetworks.com" target="_blank">here</a>
      to learn more.
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

```

<xs:element name="misc2" type="xs:string">
  <xs:annotation>
    <xs:documentation>
      <osd:label>
        Miscellaneous 2
      </osd:label>
      <osd:description>
        This is the miscellaneous element #2 and here is a
        <osd:link path="../misc1"> link to another node in the
        adaptation</osd:link>.
      </osd:description>
    </xs:documentation>
  </xs:annotation>
</xs:element>

```

If a node points to a named type, then the label of the node replaces the label of the named type. The same mechanism applies for the description of the node (`osd:description`).

Note

Regarding whitespace management, the label of a node is always *collapsed* at display (that is, contiguous sequences of blanks are collapsed to a single blank, and leading and trailing blanks are removed). In contrast, for descriptions, the whitespaces are always *preserved*.

Dynamic labels and descriptions

As an alternative to statically defining the localized labels and descriptions for each node, it is possible to specify a Java class that programmatically determines the labels and descriptions for the nodes of the data model. To define the class, include the element `osd:documentation`, with the attribute `class` in the data model. It is possible to pass JavaBean properties using nested parameter elements.

Example:

```

<xs:schema ...>
  <xs:annotation>
    <xs:appinfo>
      <osd:documentation class="com.foo.MySchemaDocumentation">
        <param1>...</param1>
        <param2>...</param2>
      </osd:documentation>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema ...>

```

The labels and descriptions provided programmatically take precedence over any labels and descriptions defined locally on individual nodes.

Voir aussi : **SchemaDocumentation** [*SchemaDocumentation*]^{API}

Enumeration labels

A simple, non-localized label can be added to each enumeration element, by means of the attribute `osd:label`.

Important: Labels defined by an enumeration element are always collapsed at display.

Example:

```
<xs:element name="Service" maxOccurs="unbounded">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="1" osd:label="Blue" />
      <xs:enumeration value="2" osd:label="Red" />
      <xs:enumeration value="3" osd:label="White" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

It is also possible to fully localize the labels by means of the standard `xs:documentation` element.

Example:

```
<xs:element name="access" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="readOnly">
        <xs:annotation>
          <xs:documentation xml:lang="en-US">
            read only
          </xs:documentation>
          <xs:documentation xml:lang="fr-FR">
            lecture seule
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="readWrite">
        <xs:annotation>
          <xs:documentation xml:lang="en-US">
            read/write
          </xs:documentation>
          <xs:documentation xml:lang="fr-FR">
            lecture écriture
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="hidden">
        <xs:annotation>
          <xs:documentation xml:lang="en-US">
            hidden
          </xs:documentation>
          <xs:documentation xml:lang="fr-FR">
            masqué
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Mandatory error message (osd:mandatoryErrorMessage)

If the node specifies the attribute `minOccurs="1"` (default behavior), then a required error message is displayed if the user does not fill the field. This error message can be specific to each node by using the element `osd:mandatoryErrorMessage`.

Example:

```
<xs:element name="birthDate" type="xs:date">
  <xs:annotation>
    <xs:documentation>
      <osd:mandatoryErrorMessage>
        Please give your birth date.
      </osd:mandatoryErrorMessage>
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

The mandatory error message can be localized:

```
<xs:documentation>
  <osd:mandatoryErrorMessage xml:lang="en-US">
    Name is mandatory
  </osd:mandatoryErrorMessage>
  <osd:mandatoryErrorMessage xml:lang="fr-FR">
    Nom est obligatoire
  </osd:mandatoryErrorMessage>
</xs:documentation>
```

Note

Regarding whitespaces management, the enumeration labels are always *collapsed* at display.

Conversion error message

For each predefined XML Schema element, it is possible to define a specific error message if the user entry has an incorrect format.

Example:

```
<xs:element name="email" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <fmt:string>
        <fmt:ConversionErrorMessage xml:lang="en-US">
          Please enter a valid e-mail address.
        </fmt:ConversionErrorMessage>
        <fmt:ConversionErrorMessage xml:lang="fr-FR">
          Saisissez un e-mail valide.
        </fmt:ConversionErrorMessage>
      </fmt:string>
    </xs:appinfo>
  </xs:annotation>
```

```
</xs:element>
```

Validation message with severity associated with a facet

The validation message that is displayed when the value of a field does not comply with a constraint can define a custom severity, a default non-localized message, and localized message variants. If no severity is specified, the default level is *error*. Blocking constraints *must* have the severity *error*.

XML Schema facet (*osd:validationMessage*)

The validation message is described by the element `osd:validationMessage` in `annotation/appinfo` under the definition of the facet.

Example:

```
<xs:element name="zipCode">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <!--facet is not localized, but validation message is localized-->
      <xs:minInclusive value="01000">
        <xs:annotation>
          <xs:appinfo>
            <osd:validationMessage>
              <severity>error</severity>
              <message>Non-localized message.</message>
              <message xml:lang="en-US">English error message.</message>
              <message xml:lang="fr-FR">Message d'erreur en français.</message>
            </osd:validationMessage>
          </xs:appinfo>
        </xs:annotation>
      </xs:minInclusive>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML Schema enumeration facet (*osd:enumerationValidationMessage*)

The validation message is described by the element `osd:enumerationValidationMessage` in `annotation/appinfo` under the definition of the field.

Example:

```
<xs:element name="Gender">
  <xs:annotation>
    <xs:appinfo>
      <osd:enumerationValidationMessage>
        <severity>error</severity>
        <message>Non-localized message.</message>
        <message xml:lang="en-US">English error message.</message>
        <message xml:lang="fr-FR">Message d'erreur en français.</message>
      </osd:enumerationValidationMessage>
    </xs:appinfo>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
```

```

<xs:enumeration value="0" osd:label="male" />
<xs:enumeration value="1" osd:label="female" />
</xs:restriction>
</xs:simpleType>
</xs:element>

```

EBX5 facet (osd:validationMessage)

The validation message is described by the element `osd:validationMessage` under the definition of the facet (which is defined in `annotation/appinfo/otherFacets`).

Example:

```

<xs:element name="price" type="xs:decimal">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:minInclusive path="../../priceMin">
          <osd:validationMessage>
            <severity>error</severity>
            <message>Non-localized message.</message>
            <message xml:lang="en-US">English error message.</message>
            <message xml:lang="fr-FR">Message d'erreur en français.</message>
          </osd:validationMessage>
        </osd:minInclusive>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

CHAPITRE **77**

Additional properties

Default value

It is possible to specify a default value for a field using the attribute `default`. This property is used for assigning a default value when no value is defined on a field.

The default value is displayed in the user input field at creation time. That is, the default value will be displayed when creating a new record or adding a new occurrence to an aggregated list.

Example:

```
<xs:element name="fieldWithDefaultValue" type="xs:string" default="aDefaultValue" />
```

In this example, the element specifies a default string value.

Access properties

The attribute `osd:access` defines the access mode, that is if the data of a particular data model node can be read and written. This attribute must have one of the following values: `RW`, `R-`, `CC` or `--` (see below).

For each node (XML Schema type), three adaptation types are possible:

1. *Adaptation terminal node*

This node is displayed with an associated value in EBX5. Access with the method `Adaptation.get()` uses the adaptation search algorithm.

2. *Adaptation non terminal node*

This node is a complex type that is displayed in EBX5 only if it has one child node that is an adaptation terminal node. It has no value. Access with the method `Adaptation.get()` returns `null`.

3. *Non adaptable node*

This node is not an adaptation terminal node and has no adaptation terminal node child. This node is not displayed in EBX5. Access with the method `Adaptation.get()` returns its default value if defined on `null`.

Access mode	Behavior
RW	<i>Adaptation terminal node</i> : value can be read and written in EBX5
R-	<i>Adaptation terminal node</i> : value can be only read in EBX5
CC	<i>Cut</i> : This is not an adaptation terminal node and no child is an adaptation terminal node. This "instruction" has priority on any child node whatever the value of their access attribute.
--	<p>If the node is a simple type, it is not adaptable. If the node is a complex type, it is not an adaptation terminal node and does not define any child node.</p> <p>The <code>root</code> node of a data model must specify this access mode.</p>
Default	<p>If the <code>access</code> attribute is not defined:</p> <ul style="list-style-type: none"> - If the node is a computed value, it is considered as <code>R-</code> - If the node is a simple type and its value is not computed, it is considered as <code>RW</code> - If the node is an aggregated list, it is then a terminal value and is considered as <code>RW</code> - Otherwise, it is not an adaptation terminal node and it does not define anything about its child nodes.

Example:

```
<xs:element name="proxyIpAddress" type="xs:string" osd:access="RW"/>
```

In this example, the element is adaptable because it is an adaptation terminal node.

Information

The element `osd:information` allows specifying additional information. This information can then be used by the integration code, for any specific purpose, by means of the method **SchemaNode.getInformation()** [SchemaNode]^{API}.

Example:

```
<xs:element name="misc" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:information>
        This is the text information of miscellaneous element.
      </osd:information>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Default view

Hiding a field in the default view

It is possible to specify that a field should be hidden by default in EBX5 using the element `osd:defaultView/hidden`. This property is used for hiding elements within the default view of a data set without defining specific access permissions. That is, if a field is configured as being hidden in the default view of a data set, then it will not be visible in all default forms and views (whether tabular or hierarchical) generated from the structure of the associated data model.

Important:

- If a field is configured to be hidden in the default view of a data set, then the access permissions associated with this field will not be evaluated.
- It is possible to display a field that is hidden in the default view of a data set by defining a custom view. Only in this case will the access permissions associated with this field will be evaluated to determine whether the field will be displayed or not.

Example:

```
<xs:element name="hiddenField" type="xs:string" minOccurs="0"/>
  <xs:annotation>
    <xs:appinfo>
      <osd:defaultView>
        <hidden>true</hidden>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the default view of a data set.

Hiding a field in the text search

To specify whether or not to hide an element in the text search tool, use the element `osd:defaultView/hiddenInSearch`. That is, if a field is configured as being hidden, then it will not be able to be selected in text search of a data set.

Important:

- If a complex field is configured as being hidden in the text search, then all the fields nested under this complex field will not be displayed in the text search.
- If a field is configured as being hidden in the default view of a data set but is not specifically hidden in the text search, then it still will not be available in the text search from the default view of a data set.

Example:

```
<xs:element name="hiddenFieldInSearch" type="xs:string" minOccurs="0"/>
  <xs:annotation>
    <xs:appinfo>
      <osd:defaultView>
        <hiddenInSeach>true</hiddenInSeach>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the text search tool of a data set.

Comparison mode

The attribute `osd:comparison` can be included on terminal node elements in order to set its comparison mode. This mode controls the how its differences are detected in a data set during comparison. The possible values for the attribute are:

default	Element is visible during comparisons of its data.
ignored	No changes will be detected when comparing two versions of modified content (records or data sets). During a merge, the data values of ignored elements are not merged on updated contents, even if the values are different. For new content, the values of ignored elements are merged. During an import of an archive, values of ignored elements are not imported when updating content. For new content, the values of ignored elements are imported.

Important:

- If a complex field is configured as being ignored during comparisons, then all the fields nested under this complex field will also be ignored.
- If a terminal field does not include the attribute `osd:comparison`, then it will be included by default during comparisons.

Restrictions:

- This property cannot be defined on non-terminal fields.
- This property cannot be defined on primary key fields.

Example:

```
<xs:element name="fieldExplicitlyIgnoredInComparison"
  type="xs:string" minOccurs="0" osd:comparison="ignored"/>
<xs:element name="fieldExplicitlyNotIgnoredInComparison"
  type="xs:string" minOccurs="0" osd:comparison="default"/>
```

In this example, the first element is explicitly ignored during comparison, the second element is explicitly included.

Categories

Categories allow the definition of "filters", which can be used to restrict the display of any data model elements that are located in tables.

To create a category, add the attribute `osd:category` to a table node in the data model XSD.

Filters on data

In the example below, the attribute `osd:category` is added to the node in order to create a category named *mycategory*.

```
<xs:element name="rebate" osd:category="mycategory"/>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="label" type="xs:string"/>
      <xs:element name="beginDate" type="xs:date"/>
      <xs:element name="endDate" type="xs:date"/>
      <xs:element name="rate" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

To activate a defined category filter on a data set in the user interface, select **Actions > Categories > *<category name>*** from the navigation pane.

Predefined categories

Two categories with localized labels are predefined:

- Hidden
- An instance node, including a table node itself, is hidden in the default view, but can be revealed by selecting **Actions > Categories > [hidden nodes]** from the navigation pane.
- A table record node is always hidden.
- Constraint (deprecated)

Restriction

Category specification does not apply on table record node, except for category Hidden.

Java reference

CHAPITRE 78

Mapping to Java

How to access data from Java?

Read-access to data is possible from various generic Java classes, mainly **Adaptation** [Adaptation]^{API} and **ValueContext** [ValueContext]^{API}. The getter methods return objects according to the mapping rules described in the next section.

Any update on data is possible only in a well-managed context:

- in the context of a procedure execution, by calling the methods `setValue...` of interface **ValueContextForUpdate** [ValueContextForUpdate]^{API};
- during user input validation, by calling the method `setNewValue` of class **ValueContextForInputValidation** [ValueContextForInputValidation]^{API}.

According to the mapping that is described in the next section, some accessed Java objects are mutable objects: these are instances of `List`, `Date` or any `JavaBean`. Consequently, these objects can be locally modified by means of their own methods. However such modifications will remain local to the returned object unless one of the above setters is invoked (and the current transaction is successfully committed).

Concurrency and isolation levels

Highest isolation level

The highest isolation level in ANSI/ISO SQL is `SERIALIZABLE`. Three execution modes guarantee the `SERIALIZABLE` isolation level, within the scope of a data space:

- if client code is run inside a `Procedure` container; this is the case for every update, XML, CSV or archive export and for data services;
- if client code accesses a data space that has been explicitly locked (see Java class `LockSpec`);
- if client code accesses a data space snapshot.

Default isolation level

If the client code is run outside the contexts enabling `SERIALIZABLE`, its isolation level depends on the persistence mode:

- In semantic mode, the default isolation level is `READ UNCOMMITTED`.
- In relational mode, the default isolation level is the database default isolation level.

Java access specificities

From a Java application, a record is represented by an instance of the class `Adaptation`, this object being initially linked to the corresponding persisted record. Then, unless the client code is executed in a context enabling the [SERIALIZABLE isolation level](#), the object can become "disconnected" from the persisted record: if concurrent updates have occurred they will not be reflected by the `Adaptation` object.

So, it is important for the client code to either be in a `SERIALIZABLE` context, or to regularly look up or refresh the `Adaptation` object.

Voir aussi :

- **`AdaptationHome.findAdaptationOrNull()`** [`AdaptationHome`]^{API}
- **`AdaptationTable.lookupAdaptationByPrimaryKey()`** [`AdaptationTable`]^{API}
- **`Adaptation.getUpToDateInstance()`** [`Adaptation`]^{API}

Mapping of datatypes

Basic rules for simple datatypes

Each XML Schema simple type corresponds to a Java class. The mapping is summarized in the following table.

Schema Datatype	Java class	Comment
xs:string	<code>java.lang.String</code>	
xs:boolean	<code>java.lang.Boolean</code>	
xs:decimal	<code>java.math.BigDecimal</code>	A <code>BigDecimal</code> is converted to double at display. Some precision loss and/or rounding problem can appear if the <code>BigDecimal</code> is made up of more than 15 digits. Moreover, the erroneous displayed value will replace the correct one if the form is submitted.
xs:date	<code>java.util.Date</code>	The returned <code>Date</code> is the beginning moment of each day, that is '00:00:00'.
xs:dateTime	<code>java.util.Date</code>	
xs:time	<code>java.util.Date</code>	The returned <code>Date</code> is always set to the following day: 1970/01/01.
xs:anyURI	<code>java.net.URI</code>	
xs:Name	<code>java.lang.String</code>	
xs:int	<code>java.lang.Integer</code>	
xs:integer	<code>java.lang.Integer</code>	Note that this mapping does not comply with the XML Schema recommendation. Indeed XML Schema specification states that <code>xs:integer</code> have no value space limitations but this value space is restricted by the Java specifications of the <code>java.lang.Integer</code> object.

Multiple cardinality on a simple element

If the attribute `maxOccurs` is greater than 1, then the element is an aggregated list and the corresponding instance in Java is an instance of `java.util.List`.

Elements of the list are instances of the Java class deduced from the rules in the table above.

Complex type definition without `osd:class` declaration

By default, a terminal node on a complex type is instantiated with an EBX5 internal class. This class provides a generic JavaBean implementation, but when Java access is needed, it is recommended to use a specific JavaBean, by using the `osd:class` declaration described in the next section.

Mapping of complex types to specific JavaBeans

It is possible to map a XML Schema complex type to a specific Java class. This is achieved by adding the attribute `osd:class` in the complex node definition. It is also required to specify the attribute `osd:access` (see example), so that the node is considered as *terminal*, unless the element has `xs:maxOccurs > 1` (because in this latter case, it is already considered as terminal).

The Java class must conform to the JavaBean protocol. This means that each child of the complex type must correspond to a JavaBean property of the class. Additionally each JavaBean property must be a read-write property, and its implementation must ensure that the value set by the setter method is returned as-is by the getter method (more particularly, contextual computations are not allowed in these methods).

Example:

```
<xs:element name="customer" osd:access="RW">
  <xs:complexType name="subscriber" osd:class="com.carRental.Customer">
    <xs:sequence>
      <xs:element name="firstName" type="xs:string"/>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In this example, the class Java `com.carRental.Customer` must define the methods `getFirstName()` and `setFirstName(String)`.

A JavaBean may have a specific user interface within EBX5. This is achieved using a **UIBeanEditor** [UIBeanEditor]^{API}.

Multiple cardinality on a complex element

If the attribute `maxOccurs` is greater than 1, then the corresponding instance in Java is:

- either an instance of `java.util.List` for an aggregated list, and every element in the list is an instance of the Java class deduced from the rules in the table above;
- or an instance of **AdaptationTable** [AdaptationTable]^{API}, if the property `osd:table` is specified.

Java bindings

Java bindings allow to generate Java types that reflect the structure of the data model. The Java code generation can be done in EBX5.

Benefits

Ensuring the link between XML Schema structure and Java code provides a number of benefits:

- Development assistance:
automatic completion when writing access path to parameters (if supported by your IDE) ;
- Access code verification:
all accesses to parameters are verified at code compilation ;
- Impact verification:
each modification of the data model impacts the code compilation state ;
- Cross-references:
by using the reference tools of your IDE, it is easy to verify where a parameter is used

Consequently, we strongly recommend to use Java bindings.

XML Declaration

The specification of the Java types to be generated from the data model is included in the main schema.

Each binding element defines a generation target. It must be located at `xs:schema/xs:annotation/xs:appinfo/ebxbnd:binding` (XPath notation) where the prefix `ebxbnd` is a reference to the naming space identified by the URI `urn:ebx-schemas:binding_1.0`. Several binding elements can be defined if you have different generation targets.

The attribute `targetDirectory` of the element `ebxbnd:binding` defines the root directory for Java types generation (generally, it is the directory of the project source codes `"src"`). A relative path is interpreted relatively to the current runtime directory of the VM (not relative to the XML schema).

See the [bindings XML Schema](#).

Example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ebxbnd="urn:ebx-schemas:binding_1.0">
  <xs:annotation>
    <xs:appinfo>
      <!-- The bindings define how this schema will be represented in Java.
      Several <binding> elements may be defined, one for each target. -->
      <ebxbnd:binding
        targetDirectory="../_ebx-demos/src-creditOnLineStruts-1.0/">
        <javaPathConstants typeName="com.creditonline.RulesPaths">
          <nodes root="/rules" prefix="" />
        </javaPathConstants>
        <javaPathConstants typeName="com.creditonline.StylesheetConstants">
          <nodes root="/stylesheet" prefix="" />
        </javaPathConstants>
      </ebxbnd:binding>
```

```
</xs:appinfo>
</xs:annotation>
...
</xs:schema>
```

Therefore, we can define Java constants for XML schema paths. For that, we generate one or several interfaces from a schema node (that can be the root node `" / "`). Hence, in the previous example, we generate two interfaces of java paths constants, one from node `" / rules "` and the other from node `" / stylesheet "` of the considered schema. Interfaces names are described by the tags *javaPathConstants* with the attribute *typeName* and the associated node is described by the tag *nodes* with the attribute *root*.

CHAPITRE 79

Tools for Java developers

EBX5 provides the Java Developer with tools that facilitate usage of the EBX5 API, and integration with development environments:

Activating the development tools

To activate development tools, you need to run EBX5 in *development* mode. It is specified in [ebx.properties](#) with the following property:

```
backend.mode=development
```

Data model refresh tool

EBX5 offers the possibility to refresh XML Schema. This tool can be accessed using the *Actions* button from the *Administration* section.

Typical scenario: the developer modifies his data model (for example, he adds new nodes) by editing the XML Schema file. He wants to see this change in EBX5, without restarting the application server.

Button 'Generate Java'

When accessing an adaptation instance or an data model, the *Generate Java* tool is accessible using the *Actions* button. This tool generates the Java types specified by the [Bindings](#).

Path to a node

The field *Data path* is displayed in the documentation pane. This field indicates the path to the node. It can be useful when writing XPath formula.

Note: this field is always available for an administrator.

